Zusammenfassung / Abstract

This work aims to facilitate the automatic counting of charged particles (ions) measured by a scintillation counter in a synchrotron. The project employs an algorithm to identify clear cases and manually label the remaining ones. The primary focus is to train machine learning networks (specifically Convolutional Neural Networks, or CNNs) to detect instances that were not consistently recognized by the algorithm alone. The work in this project involved manually checking all pile-up cases and peaks below the detection threshold, followed by labeling them for training (the necessary machine-learning code was provided). This process required adjusting various parameters and comparing the results from each model.

The results indicate that using around 100,000 peaks is inadequate for generating a model without some level of overfitting (it memorizes the data instead of learning to recognize patterns). Adjusting hyperparameters like window size increased the number of training windows without needing to acquire more files; this appeared to reduce overfitting by improving validation accuracy two-fold. Future efforts should concentrate on enhancing the peak detection algorithm for better pile-up recognition, enabling the use of a larger training dataset with less effort.

Table of contents

Introduction 5
1. Theoretical Framework 6
1.1 Particle Acceleration (Synchrotron) 6
1.1.1 What is a peak? 7
1.2 Particle counting applications 10
1.3 Convolutional Neural Networks 12
2. Methodology 13
2.1 Data Preparation 13
2.2 Labeling 14
2.2.1 Data Gathering Script 16
3. Experiments 19
3.1 Estimated Minimum Number of Errors 19
3.2 Training Results with 30 Epochs 25
3.3 Performance Metrics at Epoch 30 28
3.3.1 Training with the same data repeated three times 29
3.3.2 Manual and Automatic Labeling in ch1 Files at 150 MeV, 2 MHz 31
3.3.3 Testing Files with Higher Frequencies 36
3.3.4 Testing Files from Experimental Data 43
3.4 Experimenting with Window Sizes 46
3.4.1 Pipeline Control Script 48
3.4.2 Comparison of Model Results 50
4. Discussion 60
4.1 Comparison with Previous Findings 60
4.2 Convolutional Neural Networks vs. Peak Detection Algorithms 61
4.2.1 Potential Algorithm Modifications 62
4.3 Challenges and Limitations 63
Bibliography 64

Introduction

Accurate ion counting using a scintillation counter within a synchrotron is essential for optimizing resource efficiency and guaranteeing the precision of experimental results. Nevertheless, the detection of particles poses challenges due to pile-up events, which occur when multiple particles are detected simultaneously, resulting in overlapping signals. If these events are not appropriately identified and addressed, they may compromise the integrity of the data.

State-of-the-art methods for particle counting encompass Peak Property-Based Counting Algorithms (PPCA), Shape-Based Template Matching Frameworks (STMF), and Convolutional Neural Networks (CNNs). In previous research [1], PPCA demonstrated efficiency and speed, whereas Convolutional Neural Networks (CNNs) have exhibited greater robustness and adaptability, enabling them to handle a broader range of data. However, the challenge remains in precisely identifying various peak types, particularly in pile-up scenarios.

This project seeks to enhance the performance of CNNs in particle counting through the utilization of manually labeled data, specifically aimed at improving peak detection for peaks that are frequently overlooked by automatic algorithms. The primary objective is to reinforce the robustness of the model while reducing the necessity for manual verification in subsequent applications. The ensuing sections provide in-depth information regarding the theoretical framework and methodologies employed. (Please refer to section 1.3 for further details on CNNs.)

Initially, there were deliberations regarding the utilization of third-party labeling applications to assist in the manual identification of peaks. However, this endeavor proved unproductive due to the inherent limitations of those programs. These applications primarily supported common file formats, rendering the WFM format incompatible with any of them during the testing phase. A trial was conducted to convert the WFM format to CSV and JSON. Notwithstanding the fact that most applications were compatible with these formats, issues pertaining to file sizes emerged: the files could potentially encompass over one million lines, yet they were found to exceed three million lines following the conversion. Even attempts to divide the files were found to be ineffective.

A script provided by GSI, utilized to pre-identify peaks and automatically label them, served as the foundation for the development of a machine learning pipeline designed to prepare files for training a peak-identifying network. Furthermore, given that this automatic detection is inadequate for the objectives of the project and considering that this particular task has previously been accomplished [1], a user interface was also integrated to enable users to manually add peaks that were not previously identified. This enhancement is anticipated to yield improved results subsequent to the training of the network.

1. Theoretical Framework

This section illustrates the importance of automating particle counting and outlines the fundamentals of convoluted networks.

1.1 Particle Acceleration (Synchrotron)

According to the thesis upon which this work is based [1], the particle accelerator at GSI (refer to Figure 1) accelerates a beam of particles until the designated speed is attained. Upon reaching this speed, the particles are extracted into the detector and utilized in a predetermined experimental reaction. In this context, the detector is identified as a scintillation counter; however, it may be substituted with an alternate type depending on the specific experiment conducted.

Regarding the detectors mentioned, the 1997 article by P. Forck, T. Hoffmann, and A. Peters titled "Detectors for Slowly Extracted Heavy Ions at the GSI Facility" [2] clarifies that there are three types, each used in different contexts. A scintillation counter operates for ranges up to 10^6 particles per second (pps), while the ionization chamber functions between 10^4 and 10^9 pps, and the secondary electron monitor is applied between 10^6 and 10^{11} pps.

According to the article [2], scintillation counters are also utilized to measure the time structure of the slowly extracted spill, providing a non-integrating, dead-time-free analytical system in the frequency domain.

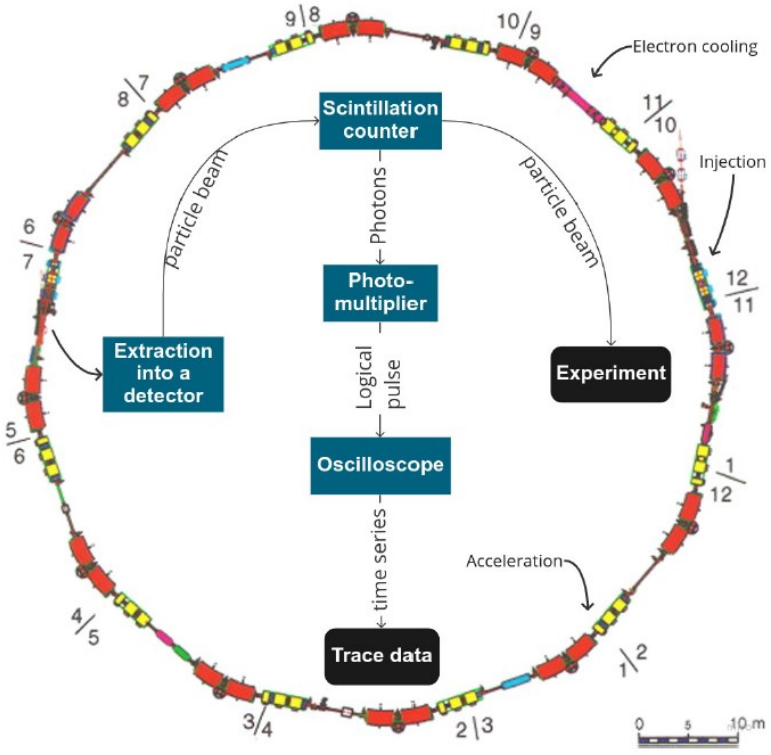


Figure 1: A diagram of particle detection after extraction is enclosed with a diagram of the SIS18 particle accelerator ring [1].

The relationship between the graphs generated by the WFM data and these measurements is straightforward. For each particle that strikes and penetrates the scintillation counter, it releases

a small amount of energy. This energy is then amplified and recorded as time series data, resulting in a signal in which the waves represent changes in energy over time.

Unfortunately, the wave peaks do not represent particles on a one-to-one basis. When multiple particles penetrate the scintillation counter in quick succession or even at the same time, this influences the graph by creating piled-up waves. Instead of appearing neat and symmetrical, the waves become slightly distorted as they merge.

The author in reference [1] further emphasizes the need to accurately account for the particles to enhance resource efficiency at GSI. For example, suppose only fifty percent of the particles are detected during an experimental procedure, and this process must be reiterated until a predetermined quantity of particles is identified. In that case, the accelerator must be activated multiple times until the target objective is accomplished.

Personal observations have indicated that simply adjusting the parameters for pulse width in nanoseconds and lowering the amplitude values significantly enhances detection efficiency compared to the current default settings used by their algorithm. Although few peaks fall below the amplitude (height) threshold and false positives exist above it (high-energy particles produce larger reflections on the graph), their overall significance is minimal. Conversely, some pile-ups are too subtle to be detected by the current algorithm. Being able to identify these should substantially improve its effectiveness.

1.1.1 What is a peak?

A peak is defined as the highest point of a wave in a V-shape (the local maxima that exceeds a predefined amplitude threshold), along with any distortion (pile-up) encountered. Piled-up peaks arise when two particles interact during the detector's dead time, resulting in overlapping waveforms with diminished peak prominence. What does not qualify as a peak (false positive) is its tail-end, the reflection (signal rebound near the noise floor), which rises slightly before transitioning to positive values on the graph.

Sidenote: Files with frequencies of 2, 5, and 10 MHz each have around 25k, 50k, and 90k peaks, respectively. Since all of them have a measurement of 400,000 ns, a peak is found approximately every 20 ns for 2 MHz, every 10 ns for 5 MHz, and every 5 ns for 10 MHz. Since one second equals one billion nanoseconds, 2,500 NPZ files would be needed to measure a single second. With 168 files used for these experiments, about 100,000 peaks in 0.0672 seconds were manually labeled in total.

There are five cases to consider when labeling peaks:

Case 1: Standard Peak

The easiest peak to spot is typically automatically detected when the pulse width variable is set to 15 nanoseconds (Figure 2). These are also evident to the naked eye: if the graph forms a "V" shape below zero, that indicates a peak.

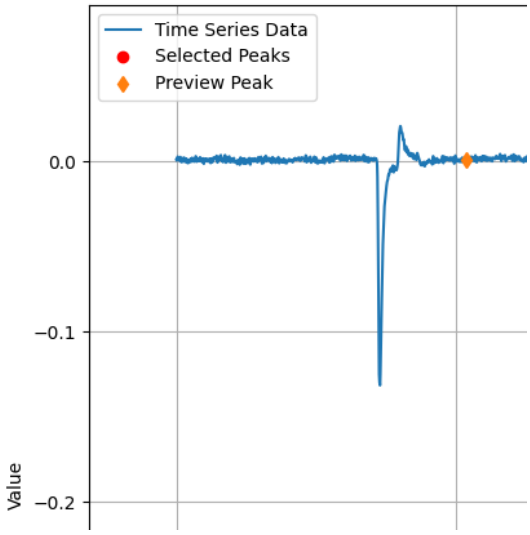


Figure 2: Standard Peak (Case 1) [Labeling UI].

Case 2: Peak is Below the Amplitude Threshold

Any wave resembling case 1 that occurs below the "height' variable (amplitude) threshold is also a peak (Figure 3). However, these can confuse the automatic detection script because reflections (circled in red in Figure 3) also fall below this threshold. The height threshold variable should be set to -0.2 to prevent the detection of reflections. However, this approach overlooks a significant number of smaller peaks.

Setting the variable to -0.1 is usually sufficient for detecting smaller peaks, although there are instances where their height can reach as low as -0.2. However, a height of -0.1 may occasionally identify reflections when the preceding peak is greater than -1.2. Future work should consider the following: if a peak below -0.2 is detected after a large -1.2 peak, it should be disregarded.

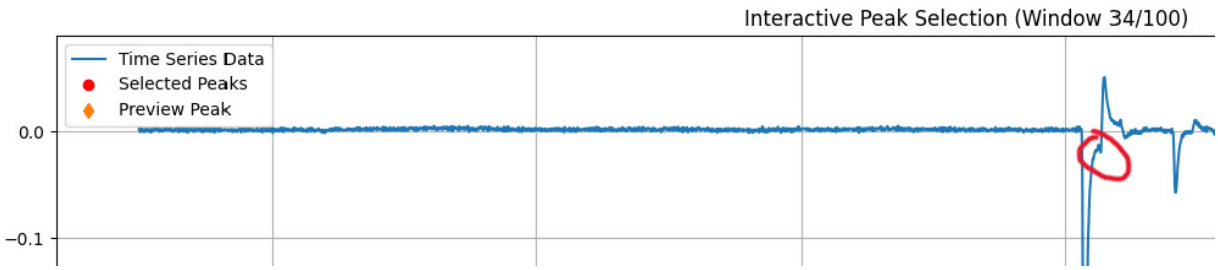


Figure 3: (Case 2) Reflection (in red) and a peak beneath the amplitude threshold [Labeling UI].

Case 3: Clearly Defined Pile-Up

A wave ascending or descending can dip or rise slightly, forming a secondary peak (Figure 4). While these are certainly visible to the naked eye, if the automatic detection has the width variable set to 15 ns, only the main peak (the highest) is detected. However, it is possible to identify them when the width is set to 6 ns; at that setting, whenever a small dip or rise is detected, they are also marked as peaks.

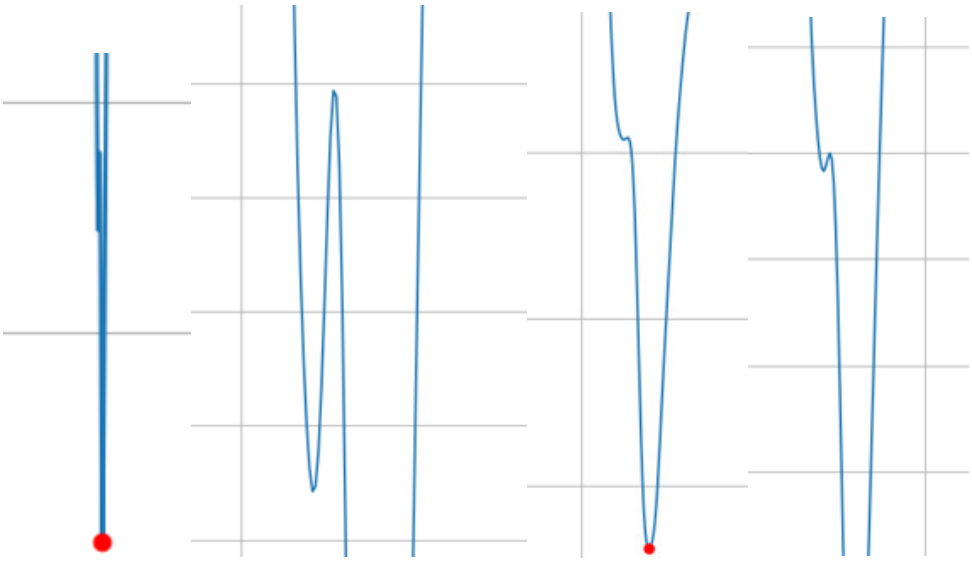


Figure 4: (Case 3) Small peak during ascent [Labeling UI].

Case 4: More Challenging to Identify Pile-ups

Consider Figure 4: a thick line can be seen on the left. Besides the piled-up peak on the left, the upper part of this wave represents only one peak. However, looking at Figures 5 and 6, it's clear that depending on the zoom level, sometimes a piled-up peak is only visible when zooming in closer, which naturally applies to the other cases mentioned in this section.

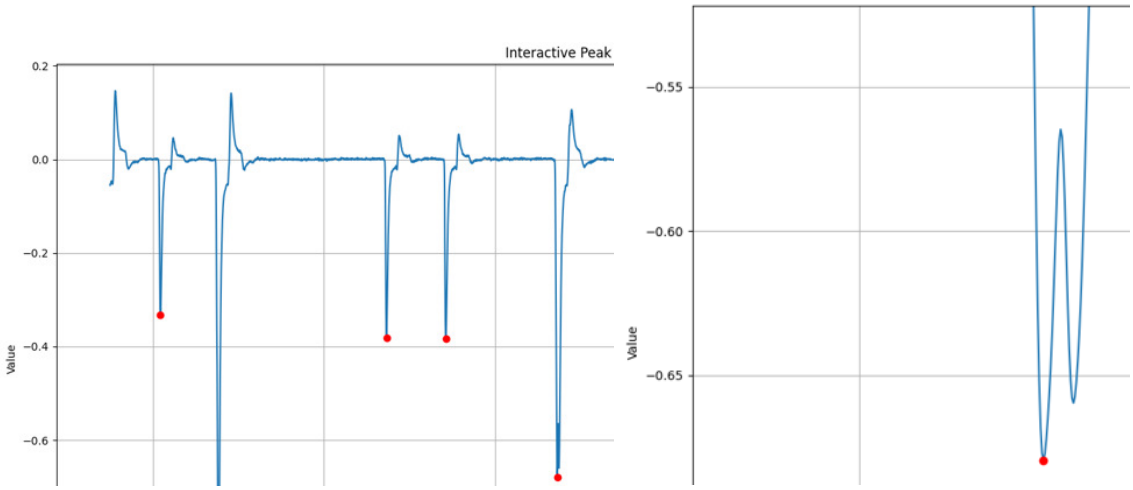


Figure 5: (Case 4) Peak becomes visible when zoomed in [Labeling UI].

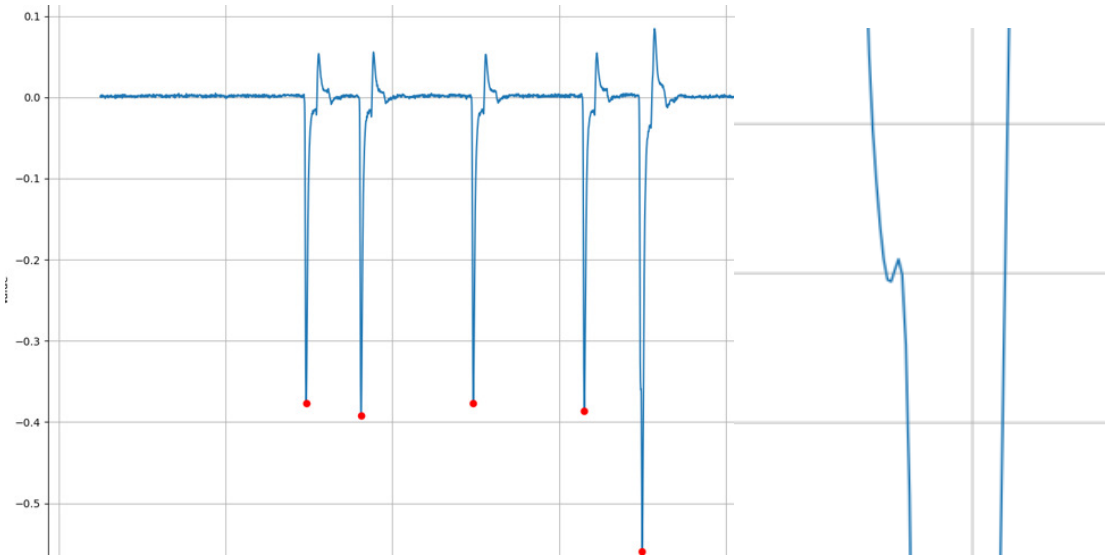


Figure 6: (Case 4) Peak visible upon zooming in [Labeling UI].

Case 5: Minor distortions

Since the machine at the GSI is calibrated sufficiently to ensure that small distortions do not introduce noise into the data, anything that deviates from a straight, slightly diagonal line is also regarded as a peak (Figure 7). A mathematical method to identify these cases could not be established in this instance. Due to the absence of dips or rises in the line, they are challenging to detect both through code and with the naked eye. Even altering the width variable to 0 ns did not allow for the detection of such peaks.

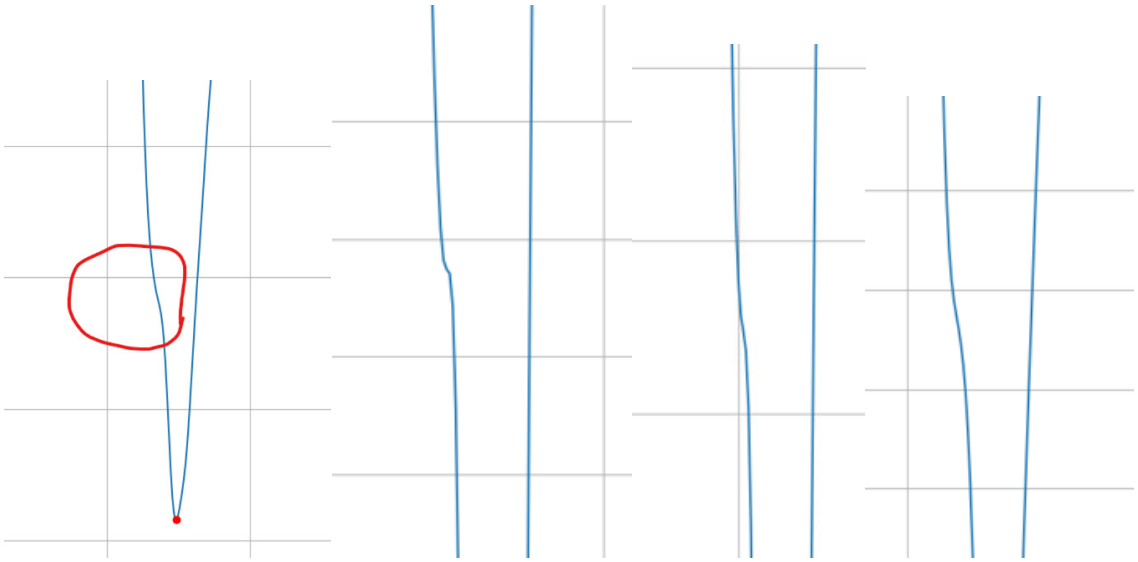


Figure 7: (Case 5) Straight line distortion [Labeling UI].

1.2 Particle counting applications

The article “Software-defined Radio-Based Feedback System for Beam Spill Control in Particle Accelerators” [3] explains the typical operational process of a circular particle accelerator. It describes how it sequentially injects, accelerates, and extracts a beam of particles, which are then delivered to target stations, experiments, or treatment facilities. The authors emphasize

achieving a controlled extraction of particles, known as the “spill,” ensuring a consistent spill rate so that an equal number of particles are released over time.

From the summary above, it is clear why having a method for precise automatic particle counting is beneficial in liberating researchers from the labor-intensive and time-consuming task of manually counting particles to determine if the spill rate is adequate. The reference to treatment rooms underscores the significance of obtaining accurate measurements.

The article “Excitation Signal Optimization for Minimizing Fluctuations in Knock Out Slow Extraction” [4] points out the treatments for which the extracted particles can be used: they can be utilized in hadron cancer therapy. The text discusses enhancing the stability of slow extraction in synchrotrons, which is crucial for consistently delivering particle beams for experiments, medical treatments, and various other accelerators.

Examining four oncology papers and reviewing their abstracts proved enlightening: one of them, “Radiological Use of Fast Protons" [5], appears to be a pioneer in its field, as it states that only electrons have been accelerated to high energies (e.g., with a synchrotron) and used directly in therapy. It is noted that protons, deuterons, and alpha particles had not been used previously due to their very short tissue penetration. This changed with the development of higher-energy machines that enabled ions to be energetic enough to penetrate more tissue. This paper clarifies that, at the time, energies above 125 MeV (million electron volts) were unavailable, and current resources indicate that even 431 MeV is achievable today.

“Radiotherapy with beams of carbon ions” mentions that hadron therapy uses beams ranging from 200 to 250 MeV. In comparison, carbon ions at 400 MeV are employed against radio-resistant tumors, which photon and proton beams have not effectively treated. It also clarifies that protons and carbon ions are more favorable than x-rays (photons) when comparing their dose-depth distribution.

“Carbon ion radiotherapy in Japan: An assessment of 20 years of clinical experience”, published in 2015, discusses the disparity between the numerous photon therapy centers and the limited number of heavy ion centers, even though heavy ion therapies “are much more effective than x-rays owing to the special radiobiological properties of densely ionizing radiation.”

The last paper, “Carbon Ion Therapy: A Modern Review of an Emerging Technology" [8], explains that these unique properties enable carbon ion radiotherapy to precisely target tumors while sparing surrounding tissues. It also discusses how this type of treatment is utilized against:

- Tumors: intracranial, head and neck, lung, cutaneous.
- • Cancer: genitourinary, gastrointestinal, breast, gynecologic, pediatric.
- • Carcinoma: recurrent nasopharyngeal, hepatocellular, adenoid cystic, metastatic prostate, prostate.
- • Also: sarcomas, chordoma, glioma, primary glioblastoma, etc.

Other mentioned applications include recurrent diseases at multiple sites and combination therapy with immunotherapy.

Hopefully, this has highlighted the importance of using machine learning for particle counting: increasing its efficiency can not only save energy and, consequently, money due to a reduced need to repeat experiments to achieve particle capture targets, but it could also potentially save lives, as a more efficient method might lead to broader adoption.

1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are employed in this work because they are well-suited for this type of challenge. CNNs excel at processing spatially structured data with a grid-like topology [9]. This ability is especially pertinent to the provided WFM files, which can be viewed as time-series data. The distinction between CNNs and other machine learning architectures lies in their use of the mathematical operation known as “convolution” instead of general matrix multiplication [9].

The article “Application of Machine Learning Towards Particle Counting and Identification” [10] highlights the motivation for this project: CNNs have proven to be the most effective method regarding accuracy, adaptability, and implementability. Thus, the authors recommend that further in-depth research should be pursued on applying CNNs for particle counting.

Due to time constraints, further suggestions made in the article [10] will not be considered. For reference, it was also noted that since their algorithms were assessed and trained only on low extraction rates, future work should also aim to enable classification at higher extraction rates without compromising accuracy. Additionally, they proposed that an algorithm could be integrated as part of the particle detector attached to the synchrotron using a Field-Programmable Gate Array (FPGA), which could imply its application for real-time analysis [10].

2. Methodology

The first task involved labeling the peaks, training a CNN model, and validating the results. Next, fine-tune the model's hyperparameters and repeat the previous two steps for each experiment in this project. Since no software was available for labeling, an interactive user interface (UI) was developed to facilitate peak labeling.

The user interface employs a secondary script to automatically label peaks that meet the specified width separation and minimum amplitude parameters using the find_peaks() function from the SciPy library. The automatic labeling identifies only the most obvious cases, while the more challenging ones (discussed in section 1.1.1) had to be added primarily by hand. Out of nearly 100,000 peaks in total, between 3,000 and 4,000 were not initially detected. Significant improvements came from adjusting the detection parameters, with only one out of the five cases remaining undetectable under the current conditions.

2.1 Data Preparation

Initial discussions with the GSI Helmholtz Centre for Heavy Ion Research began in early December to establish collaborative expectations and review previous contributions. The collaborators provided historical measurement datasets and a peak detection algorithm for preliminary analysis. After a two-week review period, subsequent efforts focused on evaluating third-party applications for labeling WFM folds, including Label Studio, ELAN, VGG Image Annotator, and MATLAB. Compatibility testing revealed limitations, as most tools could not process WFM files natively. Attempts to convert WFM datasets into CSV and JSON formats for integration with Label Studio were hindered by excessive file sizes, consisting of millions of data points.

A shape-based template matching approach was explored using the VIA. Over 5,200 peak images were manually labeled during the Christmas break to advance the project during the holidays. In January, new measurement data was supplied (including experimental data from channels 2 to 4), and a machine learning training pipeline, which includes a user interface for manual peak marking, was delivered, making the winter work redundant.

The manual labeling and model training workflow consists of three sequential stages to convert raw waveform measurements (WFM) into a dataset suitable for machine learning. Running its scripts necessitates Python 3.10 and a virtual environment to execute the pipeline, which creates a specific folder structure upon the first run of the “ProcessRawWFM” script. Although the WFM files (26 MB) provided at the beginning of the project were incompatible with the updated pipeline, newer measurements (122 MB per file) were successfully converted into ExampleData (NPZ format). Each WFM file is divided into 42 NPZ files (2 ~ 4 MB), which contain 100 discrete graphs per file.

Following this, the ExampleData files undergo manual peak annotation, a process aided by the segmented structure of the NPZ format. The dataset consists of 16,800 graphs (168 NPZ files), each containing between 300 and 1,200 peaks. A total of 99,858 peaks were labeled across the dataset. Manual labeling of ExampleData was prioritized using Channel 1 files due to their higher peak amplitudes and consistent signal rebound patterns near the noise floor, simplifying automatic annotation. In contrast, Channels 2 and 3 exhibit lower amplitudes and irregular post-peak reflections, similar to 5 MHz and 10 MHz WFM files, complicating consistent automatic labeling. Channel 1 becomes less reliable when operating at 5 or 10 MHz as its waveform

characteristics deteriorate to resemble those of Channels 2 and 3. Channel 4 remains unsupported by the pipeline, with its data excluded from conversion.

Subsequently, the labeled ExampleData is organized into a standardized MLDataset for model training. This process includes splitting the NPZ files into training (80%) and validation (20%) subsets, followed by running the "TrainingDataGeneration" script to compile all files into a cohesive MLDataset defined by fixed window sizes and peak counts per example. The provided code implements classification, whereby each window is categorized into a class determined by the number of peaks it contains. The utilization of an inappropriate number of peaks diminishes the model's precision, as the primary objective is to accurately identify pile-up peaks, thus necessitating an optimal count. However, due to the heterogeneous nature of the data, not all classes are adequately represented. Despite the occurrence of twenty peaks within a single window of size 12500, the classification must cease at the maximum count preceding a class with zero examples, as the latter would disrupt the training script.

The final stage utilizes auxiliary scripts ("TrainingExample," "EvaluateDatasetWithModel") to train machine learning models and assess their performance. The evaluation script specifically pinpoints instances of over-detection and under-detection, which later inspired another script that extracts more detailed information from the data.

2.2 Labeling

While labeling is not a complicated task, it is time-consuming and requires concentration. The algorithm initially introduced in December could only identify peaks without their pile-ups and overlooked peaks of smaller amplitudes. The ideal scenario would involve uncovering the mathematics behind all types of peaks to accurately detect and automatically label them.

Currently, changing the variable that detects the width of a wave is sufficient to identify about 90% of all peaks. Another 2% can be attributed to peaks smaller than the amplitude threshold (case 2), while the remaining 8% would be linked to pile-up peaks where the wave is only slightly distorted (case 5).

All 16800 windows were analyzed thrice because of a misunderstanding of how the manual labeling script worked. After flipping like pages through all one hundred windows, it seemed obvious that it was only needed to save the changes at the last graph. This was developed differently: each change had to be saved individually. The confusion stems from the user interface naming each page as a "window," meaning a Windows window has one hundred windows. Consequently, "Save Window" only saves the current portion of the visible graph.

The orange marker that indicates where a peak (red dot) can be added with a double left click (or removed with a double right click) was the focus of the second issue: it disrupted the native zoom of the SciPy library because it updates the graph view as the mouse moves. To address this, a zoom feature was introduced that magnified just the x-axis. Since this zoom was insufficient for accurately detecting peaks due to the distortion from enlarging the graph without preserving its proportions, that weekend was dedicated to creating an Excel table identifying which "windows" from each file contained piled-up peaks. While the WFM processing script appends a number to the end of the NPZ file names, each column represents a file, and each row indicates in which window a pile-up was found (Figure 8).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	5	0	19	5	38	2	0	13	2	0	1	4	1	6	0	3	3	3	10	6	3	10
3	10	6	20	12	60	20	8	16	3	4	6	5	2	7	4	5	10	4	18	7	8	12
4	11	12	21	21	64	24	10	21	5	30	24	6	9	19	21	27	12	7	19	9	10	15
5	13	15	28	23	65	25	11	28	6	33	28	10	11	44	23	32	13	23	32	15	16	35
6	18	16	29	32	75	27	18	37	9	39	32	11	12	45	24	36	16	25	51	23	40	38
7	20	26	37	36	78	32	21	56	10	40	35	16	16	46	29	46	23	31	53	30	44	42
8	22	30	40	46	79	34	23	61	23	54	43	23	20	48	30	75	40	38	61	39	45	48
9	27	37	48	51	83	36	31	86	28	62	45	24	25	49	33	77	43	43	92	43	51	62
10	31	38	50	81		39	33	88	34	63	47	27	31	53	43	79	47	46	98	48	53	64
11	32	39	51	82		42	35	90	35	65	51	36	35	56	50	86	65	51	99	51	56	70
12	35	42	60	90		50	39	91	41	70	70	37	36	65	59	93	67	57		57	57	80
13	38	45	92	94		51	45	98	45	74	82	41	37	71	83		74	63		60	58	84
14	47	50	96	98		52	60		47	75	85	42	38	72	85		75	65		79	67	86
15	49	60				55	69		66	77	87	47	40	78	88		77	72		83	68	89
16	50	62				57	76		75	84	88	49	48	81	96		78	82		87	90	91
17	51	73				60	78		78	94	97	53	49	96	97		85	83		90	99	
18	59	77				63	79		80	97		56	54	97			94	90		91		
19	68	79				76	86		90			57	56				96	93		93		
20	71	85				84	90		94			59	59				99	98		95		
21	72	89				87	93		99			60	65					99		99		
22	75	91				93	96					62	70									
23	77	93										63	73									
24	80	98										66	83									
25	86											69	85									
26	93											74	92									
27	99											77										

Figure 8: Spreadsheet listing locations of piled-up peaks [Excel].

Along with the expected changes that fixed the native zoom, there were also updates that allowed users to modify the values for automatic detection directly from the labeling interface (Figure 9). The automatic labeling script could now run independently rather than being called by the WFM processing script. It would detect and add peaks based on changes in the variables but without removing any peaks that had already been labeled.

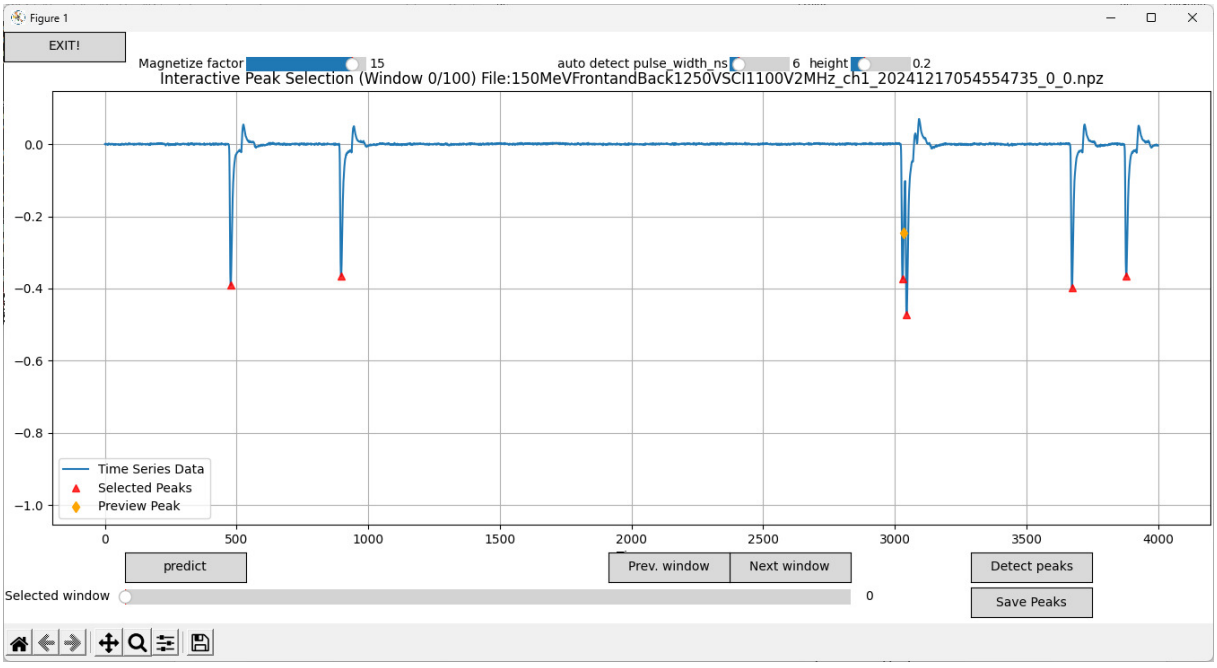


Figure 9: Manual Labeling Interface [Labeling UI].

Another change that was implemented is the “magnetic factor,” which allows the preview marker to adhere to dips in the waves, thus simplifying the labeling of peaks. One reason zooming was essential and problematic from the start is that the preview marker only tracks the mouse on the x-axis. Because peaks can only be located on the y-axis, accurately labeling them within a small window proved challenging.

- Other interface elements include the buttons:
- • “EXIT!”: interrupts the script instead of loading the next file until all files are read.
- • Previous window / Next window: alters the currently viewed graph position.
- • Predict: activates the last trained model and displays its predictions.
- • Detect peaks: utilizes the specified values for width and amplitude above the graph to identify peaks within those parameters.
- • Save peaks: Save changes by overwriting the existing NPZ file.

The “predict” button was deactivated because it invoked the model, even if no model had been trained yet, making it impossible to begin manually labeling the data to train the initial model. The 'Selected window' slider not only indicates which graph position is currently displayed but also facilitates the quick selection of another position. Below, the native buttons from SciPy used during this project pertained to zooming in on the graph and saving the current window as an image.

2.2.1 Data Gathering Script

Even after training the model, the statistics regarding its performance (e.g., accuracy and validation accuracy) are insufficient and can be misleading. While the error rate tends to be inversely proportional to validation accuracy, it is crucial to determine whether, within a specific window, there were either over-detection or under-detection than expected and how many times the model produced correct results (exact matches). Additionally, comparing the labeled and predicted peaks is necessary to assess the model's success. The ideal outcome would involve 100% correctly predicted peaks with 0% errors. If these metrics are not proportional, it is likely that false positives or false negatives are present. To facilitate these evaluations, a new script needed to be developed.

The pipeline already contained a script responsible for evaluating the dataset against the trained model. It examines every file in the folder used for manual labeling, then interacts through all 100 windows, comparing the number of labeled peaks with the number of peaks detected by the model. If the values differ, it plots a graph with the peaks (but without any markings like the red dots used to label peaks in manual labeling) or prints a message if the numbers are the same.

Based on this principle, a new script was developed. In this script, the if-else statements were modified to check if the number of peaks is less than, greater than, or equal to the detected peaks. This adjustment allows for identifying when the model incorrectly predicts the number of peaks in a window and cataloging it appropriately. Each statement functions as follows:

- In the first case, add the difference between both values to a total error count while also adding it to a separate counter that tracks only these additional peaks detected.

```
if amount_of_peaks < predicted_peaks: # Over-detection
    diff = predicted_peaks - amount_of_peaks
    file_extra_peaks += diff
    file_error_quantity += diff
```

- In the second case, increase the general error counter, and also add to another counter for instances when fewer peaks were detected than anticipated.

```
elif amount_of_peaks > predicted_peaks: # Fewer Peaks
    diff = amount_of_peaks - predicted_peaks
    file_fewer_peaks += diff
    file_error_quantity += diff
```

- Finally, if the two numbers are identical, another counter tallies the exact matches found.

```
else: # Exact matches
    file_exact_match_quantity += amount_of_peaks
```

- Two global variables track the total number of labeled peaks and the number of detected peaks. Their values are updated for each new window iterated upon.
- Another variable is utilized for regex pattern matching to display only the last identifying numbers from the files.

```
# Extract the x_y part from the filename
match = re.search(r"(\d+_\d+)\.npz$", example_filepath) # Filter 0_00
# Filter 0_00
x_y_part = match.group(1)
wfm_group = x_y_part.split('_')[0]  # Extract WFM group (e.g., '0' from
'0_123')
```

- A verification for false positives is also conducted:

```
peaks_over_threshold = sum(1 for y in child_example.peaks_y if y <= -1.2)
file_peaks_over_1_2 += peaks_over_threshold
```

- Finally, the statistics are displayed in the command line.

```
print(f"\nFile {x_y_part}:")
print(f"Labeled Peaks = {file_total_labeled}")
print(f"Predicted Peaks = {file_total_predicted}")
print(f"Peak Difference = {abs(file_total_labeled - file_total_predicted)}")
print(f"Total errors: {file_error_quantity}")
print(f"Total of exact matches: {file_exact_match_quantity}")
print(f"EXTRA: {file_extra_peaks}")
print(f"FEWER: {file_fewer_peaks}")
print(f"Peaks over 1.2: {file_peaks_over_1_2}")
```

Since only y-axis values are used for peak identification—although utilizing the x-axis could be beneficial for more challenging cases—it is impossible to determine if the cases identified as exact matches actually represent the same peaks or if they were in different positions but coincidentally had the same quantity in the window. This situation arises because the pipeline employs a classifier: depending on the number of classes established, the model can only count the peaks and classify which window they belong to. This means the location of the peaks doesn't matter, only the quantity found. Considering this situation is relevant, its rare occurrence renders it trivial.

One crucial realization in identifying false positives was tracking how many peaks were above 1.2 (or below -1.2), as that amplitude tends to produce reflections exceeding the height threshold variable. This value indicates the maximum number of reflections that could be inaccurately classified as peaks. Since not all such peaks create this anomaly, this count merely reflects the theoretical maximum, assuming that all 1.2 peaks also resulted in a high reflection that was mistakenly identified as a peak.

Training using various metrics

Adjusting the detection's width and height variables significantly alters the results obtained. Furthermore, window size is another crucial factor. Therefore, three sets of tests need to be conducted:

- Comparing the model created with manually labeled data to those generated through automatically labeled peaks only.
- • Utilizing the best-performing parameters to label files with varying frequencies and to train their models automatically.
- • Varying window size while using the same parameters as the second set of tests.

3. Experiments

For the presentation of data in this section, the raw numbers were converted to percentages for better understanding (where needed); only the number of labeled particles remained unchanged (they would otherwise represent 100%). From this point on, it is important to remember that the total number of labeled peaks varies for each column. Therefore, a direct comparison of values in other rows is not entirely accurate but provides a useful baseline for the results.

Below is an example of a table (Table I) that will be frequently seen. On the left are the evaluated variables, while the right column displays their values. The first three values compare what the algorithm labeled, what the model predicted, and their difference (the absolute value of their subtraction). The number of "Peaks over 1.2" indicates how many waves were high enough to potentially generate a false positive with their reflection. "Total errors" summarizes the values of over-detection and under-detection, while the "Total of exact matches" shows how often the model accurately guessed the correct class for a window.

Table I: Example of common values.

Variables	Total
Labeled Peaks	99858
Predicted Peaks	93480
Peak Difference	6378
Total errors	21120
Total of exact matches	23216
Over-detection	7371
Under-detection	13749
Peaks over 1.2	282

3.1 Estimated Minimum Number of Errors

When using a 15 ns width and 0.2 height, the number of errors in each file containing one hundred windows ranges between one-tenth and one-half of the cases. Below (Figures 10 to 13) is the list of each window where at least one peak was not labeled. Since the zoom function was not accessible then, this data does not include all peaks from cases 4 (which are hard to see) and 5 (which show slight distortions), as they are the only instances where it is not obvious when the occurrences happen.

Since the WFM file names are too lengthy, they will be referred to by numbers 0 to 3, totaling four. Each NPZ file contains one hundred windows, and the peak count ranges from 312 to 1,122, with an average of 587. In total, there are 16,800 windows, where at least 2,801 errors were manually identified out of 99,858 peaks, which means that approximately one error is found for every six windows. Therefore, the average overall error rate is roughly 2.84%, while the success rate is about 97.17% (Table II).

Table II: Minimum errors found.

File #	Errors (≥)	Total Peaks	Average Error Rate
0	802	25,091	3.20%
1	629	23,315	2.70%
2	618	24,161	2.56%
3	752	25,969	2.90%

File #0

With at least 802 errors among 25,091 peaks, the average error rate stands at 3.20%.

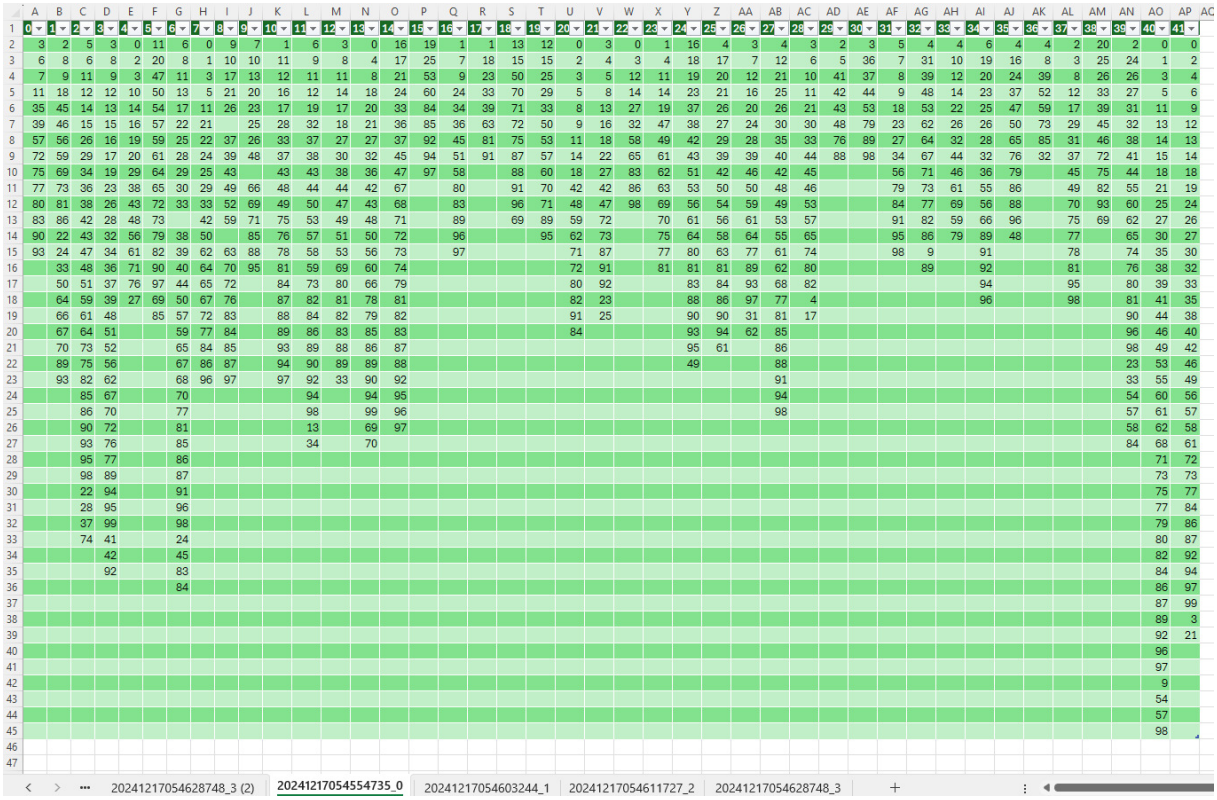


Figure 10: File 20241217054554735_0.wfm [Excel].

The number of errors below (Table III to VI) does not appear to directly correlate with the peak density in the file. The example highlighted in bold illustrates two edge cases where a similar number of peaks displays both higher and lower error percentages.

Table III: Errors in file zero.

File	Peaks	Errors	% Errors
0_00	**531**	**14**	**2.64%**
0_01	**522**	**22**	**4.21%**
0_02	**765**	**32**	**4.18%**
0_03	732	34	4.64%
0_04	550	17	3.09%
0_05	539	18	3.34%
0_06	657	34	5.18%
0_07	616	22	3.57%
0_08	678	20	2.95%
0_09	623	14	2.25%
0_10	679	22	3.24%
0_11	758	26	3.43%
0_12	**801**	**22**	**2.75%**
0_13	686	26	3.79%
0_14	624	25	4.01%
0_15	430	9	2.09%
0_16	381	14	3.67%
0_17	383	8	2.09%
0_18	504	12	2.38%
0_19	596	13	2.18%
0_20	634	19	3.00%

File	Peaks	Errors	% Errors
0_21	531	18	3.39%
0_22	410	11	2.68%
0_23	465	15	3.23%
0_24	592	21	3.55%
0_25	607	20	3.29%
0_26	551	19	3.45%
0_27	675	24	3.56%
0_28	632	18	2.85%
0_29	415	8	1.93%
0_30	432	8	1.85%
0_31	522	14	2.68%
0_32	508	15	2.95%
0_33	625	13	2.08%
0_34	509	17	3.34%
0_35	554	13	2.35%
0_36	525	8	1.52%
0_37	523	17	3.25%
0_38	524	12	2.29%
0_39	687	26	3.79%
0_40	1122	44	3.92%
0_41	993	38	3.83%

File #1

With a total of 629 errors out of 23,315 peaks, the average error rate is 2.70%.

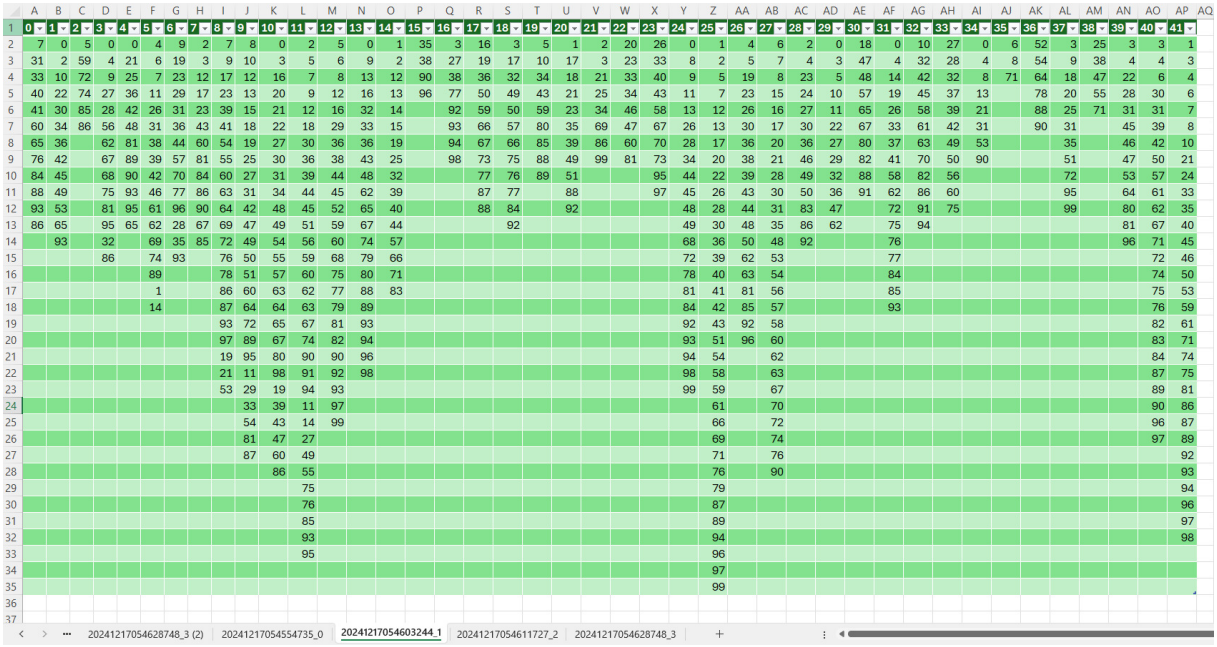


Figure 11: File 20241217054603244_1.wfm [Excel].

Table IV: Errors in file one.

File	Peaks	Errors	% Errors
1_00	524	12	2.29%
1_01	439	13	2.96%
1_02	394	6	1.52%
1_03	435	14	3.22%
1_04	555	12	2.16%
1_05	585	17	2.91%
1_06	472	14	2.97%
1_07	464	13	2.80%
1_08	673	22	3.27%
1_09	785	26	3.31%
1_10	767	27	3.52%
1_11	725	32	4.41%
1_12	713	24	3.37%
1_13	837	21	2.51%
1_14	671	16	2.38%
1_15	438	4	0.91%
1_16	344	8	2.33%
1_17	433	11	2.54%
1_18	417	12	2.88%
1_19	451	9	2.00%
1_20	437	11	2.52%

File	Peaks	Errors	% Errors
1_21	436	8	1.83%
1_22	405	8	1.98%
1_23	419	10	2.39%
1_24	734	22	3.00%
1_25	954	34	3.56%
1_26	775	19	2.45%
1_27	691	27	3.91%
1_28	627	13	2.07%
1_29	535	12	2.24%
1_30	518	10	1.93%
1_31	515	17	3.30%
1_32	599	12	2.00%
1_33	610	11	1.80%
1_34	481	8	1.66%
1_35	326	3	0.92%
1_36	312	6	1.92%
1_37	344	11	3.20%
1_38	385	5	1.30%
1_39	469	13	2.77%
1_40	694	25	3.60%
1_41	927	31	3.34%

File #2

With at least 618 errors out of 24,161 peaks, the average error rate stands at 2.56%.

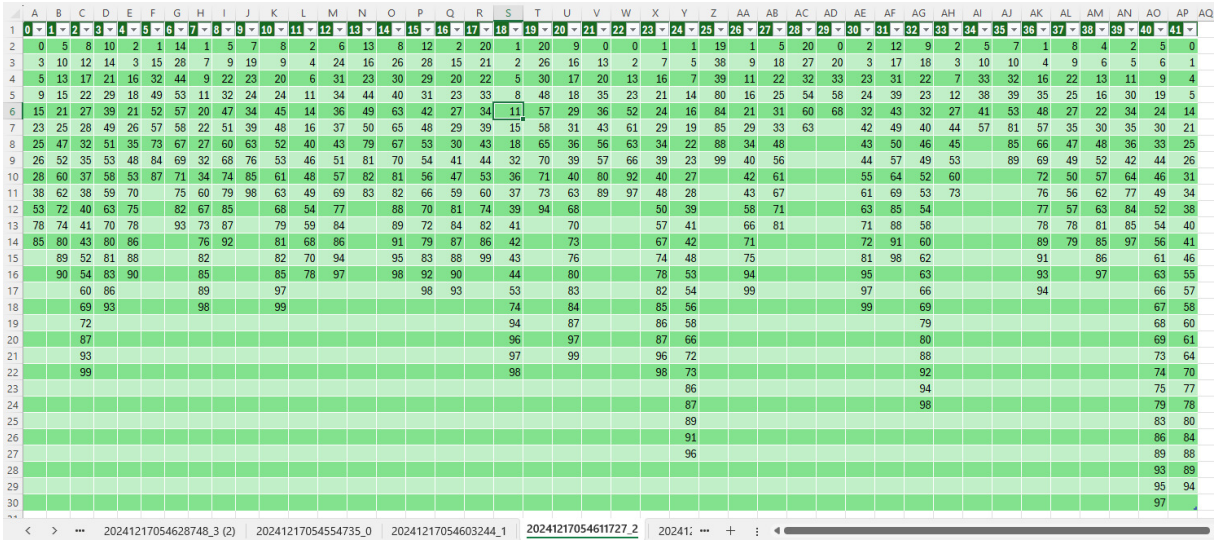


Figure 12: File 20241217054611727_2.wfm [Excel].

Table V: Errors in file two.

File	Peaks	Errors	% Errors
2_00	576	13	2.26%
2_01	700	15	2.14%
2_02	744	21	2.82%
2_03	678	17	2.51%
2_04	607	15	2.47%
2_05	461	9	1.95%
2_06	508	12	2.36%
2_07	644	17	2.64%
2_08	544	13	2.39%
2_09	553	10	1.81%
2_10	677	17	2.51%
2_11	626	15	2.40%
2_12	562	15	2.67%
2_13	451	10	2.22%
2_14	471	15	3.18%
2_15	595	16	2.69%
2_16	689	16	2.32%
2_17	608	14	2.30%
2_18	641	21	3.28%
2_19	594	11	1.85%
2_20	512	20	3.91%

File	Peaks	Errors	% Errors
2_21	409	10	2.44%
2_22	472	10	2.12%
2_23	697	21	3.01%
2_24	787	26	3.30%
2_25	518	8	1.54%
2_26	466	16	3.43%
2_27	502	12	2.39%
2_28	412	6	1.46%
2_29	382	5	1.31%
2_30	435	17	3.91%
2_31	555	14	2.52%
2_32	600	23	3.83%
2_33	486	10	2.06%
2_34	348	6	1.72%
2_35	370	8	2.16%
2_36	558	16	2.87%
2_37	563	13	2.31%
2_38	605	15	2.48%
2_39	768	13	1.69%
2_40	1022	29	2.84%
2_41	765	28	3.66%

File #3

With a minimum of 752 errors out of 25,969 peaks, the average error rate stands at 2.90%.

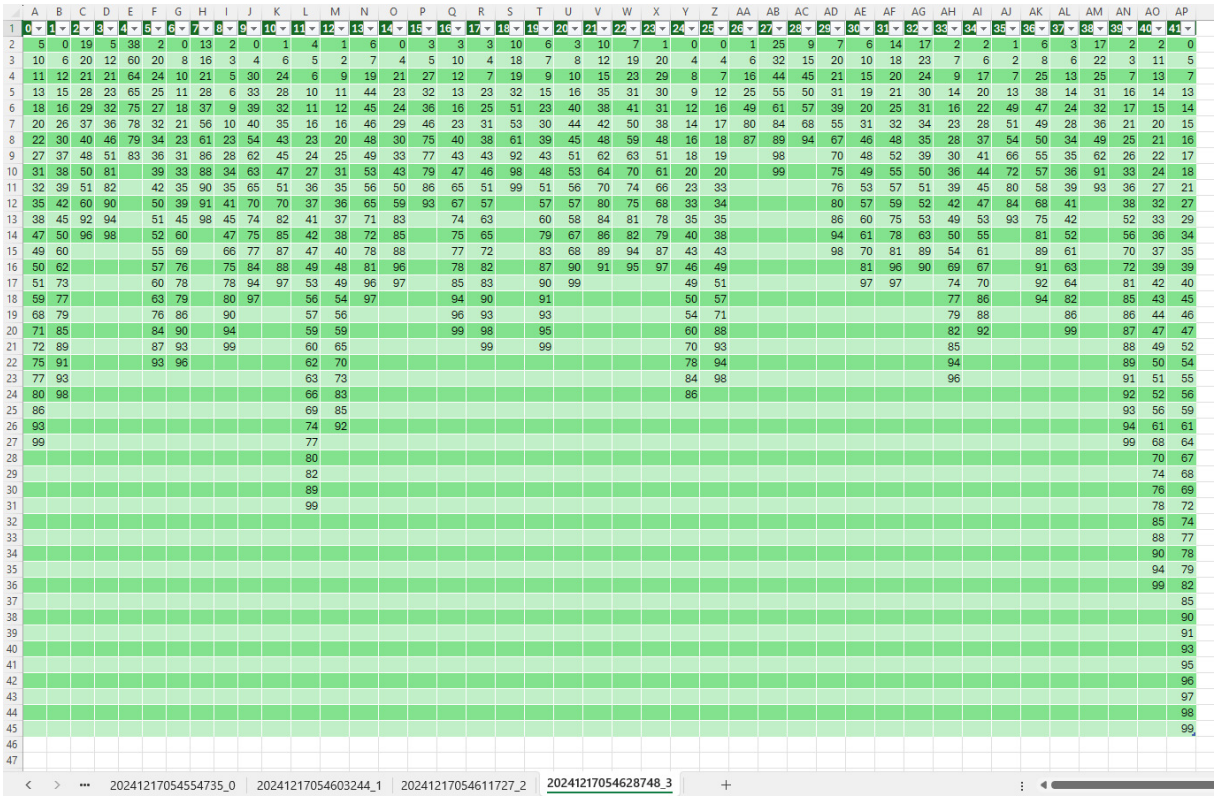


Figure 13: File 20241217054628748_3.wfm [Excel].

Table VI: Errors in file three.

File	Peaks	Errors	% Errors
3_00	746	26	3.49%
3_01	723	23	3.18%
3_02	626	13	2.08%
3_03	488	13	2.66%
3_04	570	8	1.40%
3_05	591	21	3.55%
3_06	542	21	3.87%
3_07	530	12	2.26%
3_08	963	20	2.08%
3_09	1005	17	1.69%
3_10	676	16	2.37%
3_11	605	30	4.96%
3_12	607	25	4.12%
3_13	777	17	2.19%
3_14	745	16	2.15%
3_15	657	11	1.67%
3_16	565	19	3.36%
3_17	461	20	4.34%
3_18	638	10	1.57%
3_19	666	20	3.00%
3_20	503	16	3.18%

File	Peaks	Errors	% Errors
3_21	599	15	2.50%
3_22	557	15	2.69%
3_23	505	15	2.97%
3_24	554	23	4.15%
3_25	586	22	3.76%
3_26	662	7	1.06%
3_27	511	9	1.76%
3_28	435	7	1.61%
3_29	465	14	3.01%
3_30	467	16	3.43%
3_31	500	16	3.20%
3_32	647	15	2.32%
3_33	636	22	3.46%
3_34	648	19	2.93%
3_35	723	12	1.66%
3_36	687	17	2.47%
3_37	568	19	3.35%
3_38	611	10	1.64%
3_39	565	26	4.60%
3_40	608	35	5.76%
3_41	751	44	5.86%

3.2 Training Results with 30 Epochs

After labeling the data, the next step in the pipeline is to generate files used for training the network. The script slices the data, dividing it into examples with a range of peaks from 0 to 13: Thirteen represents the number of classes with which the model could begin training with sufficient examples for a window size of 12500; the number of classes varies according to window size, peak density, and other factors. Once it runs to gather examples for training, it repeats the process for the evaluation set. As mentioned earlier in the Methodology section, the first five columns (excluding “Peaks”) utilized the same four WFM files, while the last column added another eight, being trained with a width of 6 ns and an amplitude threshold of 0.1.

Training Data Slices

The first set of models utilized the same files for training with different parameters, making the example appear connected to the number of peaks in a file. However, even training a model with six times the number of peaks only tripled the number of samples sliced (Table VII).

Table VII: Example slices from the 2 MHz training dataset

Peaks	Manual	w=15ns, h=0.2	w=15ns, h=0.1	w=6ns, h=0.1	w=6ns, h=0.2	12 WFM
0	279	291	287	287	291	**536**
1	821	865	856	837	835	**1836**
2	877	916	911	885	884	**1963**
3	869	918	917	892	887	**1995**
4	886	928	923	896	898	**2069**
5	867	925	938	882	879	**2152**
6	874	910	906	897	894	**2252**
7	871	925	902	873	883	**2406**
8	867	911	903	882	881	**2490**
9	864	905	902	874	871	**2546**
10	744	686	688	726	722	**2560**
11	573	493	495	549	556	**2566**
12	441	345	356	425	412	**2574**
13	435	317	324	402	391	**2580**
TOTAL	**10268**	**10335**	**10308**	**10307**	**10284**	**30525**

Table VIII shows that for files at 2 MHz, each file contributes approximately 10% of its total peaks for sample generation. In contrast, 5 MHz and 10 MHz consist of three files, with approximately 50,000 and 90,000 peaks each, respectively. Therefore, even though 2 MHz, with six files, managed to obtain nearly triple the samples of 5 MHz alone, training both together does not affect the total number of peaks sampled since they simply add up. Regarding 10 MHz, many training slices are unrepresented due to their shorter and more numerous waves, making it impossible to train the network with this current code.

Table VIII: Example slices from training datasets in higher frequencies

Peaks	6 files w=6ns h=0.1 at 2 MHz	3 files w=6ns h=0.1 at 5 MHz	3 files w=6ns h=0.2 at 5 MHz	3 files w=6ns h=0.1 at 10 MHz	w=6ns h=0.1 at 2+5 MHz	w=6ns h=0.1 at 2+10 MHz	w=6ns h=0.1 at 5+10 MHz
0	408	1	1	0	409	409	1
1	1295	6	6	0	1301	1335	6
2	1361	18	19	0	1377	1401	18
3	1367	61	63	0	1407	1412	62
4	1371	141	160	1	1462	1410	142
5	1369	244	278	0	1522	1416	255
6	1367	400	481	6	1597	1414	420
7	1367	584	683	6	1704	1406	605
8	1354	707	798	13	1776	1398	755
9	1330	750	803	76	1804	1371	904
10	1112	761	810	107	1814	1205	928
11	740	771	827	190	1824	926	952
12	535	779	825	290	1825	748	991
13	486	791	828	379	1833	707	1002
TOTAL	**15462**	**6014**	**6582**	**1068**	**21655**	**16558**	**7041**

Evaluation Data Slices

Even though the evaluation data only accounts for 20% of the files, reviewing the total number of files across all frequencies raises questions about what went wrong. The division of training data (Table IX) appeared to capture a proportional amount from each file type, but that was not the case at all.

Table IX: Example slices from the 2MHz evaluation dataset

Peaks	Manual	w=15ns, h=0.2	w=15ns, h=0.1	w=6ns, h=0.1	w=6ns, h=0.2	12 WFM
0	31	31	31	31	31	**0**
1	164	171	169	167	169	**0**
2	206	218	216	208	211	**0**
3	230	229	243	221	234	**0**
4	228	240	234	247	226	**1**
5	230	232	235	232	229	**0**
6	226	247	257	224	228	**6**
7	230	236	232	244	241	**6**
8	234	240	230	225	224	**16**
9	223	233	232	221	222	**69**
10	213	204	198	206	210	**119**
11	167	152	149	161	161	**205**
12	143	81	90	126	124	**315**
13	102	84	89	103	95	**437**
TOTAL	**2627**	**2598**	**2605**	**2616**	**2605**	**1174**

Table X clarifies the previous discussion: since both the 5 and 10 MHz files lack sliced peaks, they do not contribute to an increase in their number. Nevertheless, it is anticipated that the values reflected in the first column will be present. since adding 2 and 5 MHz tends to increase their total slices, it is odd that introducing 10 MHz into the mix causes the process to fall apart.

Table X: Example slices from the evaluation dataset in higher frequencies

Peaks	6 files w=6ns h=0.1 at 2 MHz	3 files w=6ns h=0.1 at 5 MHz	3 files w=6ns h=0.2 at 5 MHz	3 files w=6ns h=0.1 at 10 MHz	w=6ns h=0.1 at 2+5 MHz	w=6ns h=0.1 at 2+10 MHz	w=6ns h=0.1 at 5+10 MHz
0	59	0	0	0	59	154	0
1	283	0	0	0	283	599	0
2	344	2	2	0	346	625	5
3	359	7	11	0	363	627	9
4	354	23	24	0	366	635	25
5	353	50	48	0	382	626	56
6	354	84	108	1	394	628	87
7	346	125	156	2	432	628	143
8	340	160	194	4	443	630	175
9	325	195	202	7	463	588	227
10	295	205	225	26	470	540	235
11	211	212	225	39	477	389	253
12	136	216	229	77	489	306	261
13	146	221	231	109	494	282	270
TOTAL	**3905**	**1500**	**1655**	**265**	**5461**	**7257**	**1746**

An astute reader may have already discerned the issue with the aforementioned approach: the window size remains unchanged. Nevertheless, the peak density has escalated exponentially, and the current amount of classes does not constitute a substantial dataset segment. Previously, a window could accommodate thirteen peaks simultaneously; thus, with a doubling of peaks, this quantity would also double, with a similar outcome for quadrupling the peaks. For a frequency of 5 MHz, an increase in the number of classes to thirty suffices for model training, while for 10 MHz, this requirement rises to 41 (or 40 if training across all frequencies collectively). As this section was intended solely to provide an overview of how the provided code collects examples, additional details pertaining to instances of higher class amounts have been excluded for the sake of conciseness and their lack of relevance.

3.3 Performance Metrics at Epoch 30

After testing various epoch quantities, the standard tests were set to 30 since that is when the validation loss begins to increase. The validation accuracy levels off at 60% when the epoch reaches 45, but this merely reflects the network memorizing the data due to the limited number of files available for experimentation. After 45 epochs, the validation loss spikes, and accuracy ceases to improve, even starting to decline overall. In other words, cross-entropy loss stagnates after 30 epochs, indicating the need for early stopping or hyperparameter tuning. Additionally,

the model demonstrates a generalization gap (training accuracy: 98% vs. validation: 41%), which points to overfitting.

Regardless of how the parameters are changed, the results (Table XI) remain indistinguishable from one another due to the small amount of data used for training.

Table XI: Training results

Metric	Manual	w=15ns, h=0.2	w=15ns, h=0.1	w=6ns, h=0.1	w=6ns, h=0.2	12 WFM
Training Loss	0.0771	0.0467	0.0698	0.0723	0.0630	**0.0881**
Training Accuracy	98.10%	98.81%	98.39%	98.31%	98.26%	**97.33%**
Top-1 Accuracy	98.10%	98.81%	98.39%	98.31%	98.26%	**97.33%**
Top-3 Accuracy	100.00%	99.97%	100.00%	99.95%	99.97%	**99.99%**
Validation Loss	32.286	36.394	30.102	25.673	32.091	**N/A**
Validation Accuracy	41.19%	41.44%	39.95%	42.68%	41.69%	**N/A**
Val Top-1 Accuracy	41.19%	41.44%	39.95%	42.68%	41.69%	**N/A**
Val Top-3 Accuracy	79.16%	79.90%	80.40%	80.65%	79.65%	**N/A**

Due to the unexpected decrease in accuracy when using six times the number of peaks, additional tests were conducted to verify all data types against one another. Since higher-frequency files exhibit an increased peak density, changing the number of classes is necessary. Otherwise, when windows containing a higher quantity of peaks are encountered, they are recorded as an error instead of counted. As a result, the values from training 12 WFM files achieved 100% in Training Accuracy, Top-1 Accuracy, and Top-3 Accuracy while having a Training Loss of 0.0781, even though no validation data was provided.

3.3.1 Training with the same data repeated three times

The manually labeled dataset was used to train three independent models with identical hyperparameters to evaluate model stability. The results indicate that training a network with the same files can yield significant variations, thereby corroborating the earlier assertion regarding the results of different parameters being sufficiently similar to be regarded as identical.

The values in this table indicate the results for each file, with the totals aggregated at the end. Labeled Peaks represent those added either manually or automatically. Predicted Peaks display the network's results, while the difference between the two is presented afterward. The error reflects the number of peaks that were detected beyond what was labeled (Over-detection) or that should have been predicted but were not (Under-detection). When the labeled and predicted peaks match exactly, the peak counts are combined. Lastly, "Peaks over 1.2" signifies the maximum number of potential false positives (reflections at the end of a wave counted as peaks for exceeding the amplitude threshold).

With the tables below (XII to XVI), the labeled peaks are the same for all columns, so they will be mentioned outside of the table.

In WFM Group 0, there were a total of 25848 labeled peaks:

Table XII: Evaluation from file 0 in three separate models trained on the same data

Variables	WFM 0 (%)	WFM 0 (%)	WFM 0 (%)
Predicted Peaks	92,25	92,60	93,48
Peak Difference	7,75	7,40	6,52
Total errors	20,74	21,24	20,60
Total of exact matches	23,02	23,15	23,26
Over-detection	6,49	6,92	7,04
Under-detection	14,24	14,32	13,56
Peaks over 1.2	0,00	0,00	0,30

In WFM Group 1, there were a total of 23467 labeled peaks:

Table XIII: Evaluation from file 1 in three separate models trained on the same data

Variables	WFM 1 (%)	WFM 1 (%)	WFM 1 (%)
Predicted Peaks	92,45	92,87	94,09
Peak Difference	7,55	7,13	5,91
Total errors	21,76	23,02	22,08
Total of exact matches	23,67	22,34	22,83
Over-detection	7,10	7,94	8,09
Under-detection	14,65	15,08	13,99
Peaks over 1.2	0,00	0,00	0,24

In WFM Group 2, there were a total of 24334 labeled peaks:

Table XIV: Evaluation from file 2 in three separate models trained on the same data

Variables	WFM 2 (%)	WFM 2 (%)	WFM 2 (%)
Predicted Peaks	94,07	94,33	95,04
Peak Difference	5,93	5,67	4,96
Total errors	21,28	21,76	20,99
Total of exact matches	23,34	24,13	23,65
Over-detection	7,68	8,05	8,01
Under-detection	13,60	13,72	12,98
Peaks over 1.2	0,00	0,00	0,25

In WFM Group 3, there were a total of 26209 labeled peaks:

Table XV: Evaluation from file 3 in three separate models trained on the same data

Variables	WFM 3 (%)	WFM 3 (%)	WFM 3 (%)
Predicted Peaks	95,57	95,46	95,52
Peak Difference	4,43	4,54	4,48
Total errors	20,89	21,13	20,70
Total of exact matches	23,01	23,20	23,06
Over-detection	8,23	8,30	8,11
Under-detection	12,66	12,84	12,59
Peaks over 1.2	0,00	0,00	0,40

Examining each file individually (tables XII to XV), some numbers show minimal relevant variation, as the totals of exact matches and other values fluctuate unpredictably. In WFM Group 1, for example, the difference between labeled and predicted peaks approaches 1.7% (400 peaks), indicating that the network is merely guessing and did not learn effectively from the provided data and parameters.

A total of 99858 labeled peaks were identified (table XVI):

Table XVI: Evaluation average from all files in three separate models trained on the same data

Variables	Total (%)	Total (%)	Total (%)
Predicted Peaks	93,61	93,84	94,54
Peak Difference	6,39	6,16	5,46
Total errors	21,15	21,76	21,07
Total of exact matches	23,25	23,21	23,20
Over-detection	7,38	7,80	7,81
Under-detection	13,77	13,96	13,26
Peaks over 1.2	0,00	0,00	0,30

When considering all factors, the overall perspective becomes clearer: approximately 6% of the peaks are not accurately predicted. Moreover, easy cases of piled-up peaks are often overlooked. A questionable factor regarding the training's success, which appears improbable, is the count of additional peaks detected. These extra peaks (over-detection) were either missed during labeling or misidentified as reflections (which seems more plausible, as the false positive rate in this context was around 0.3% of all peaks, which could be an acceptable error margin).

3.3.2 Manual and Automatic Labeling in ch1 Files at 150 MeV, 2 MHz

While manual labeling is the gold standard for machine learning training, the tremendous effort required to barely gather enough data for efficient network training is astronomical. Consequently, discovering the right parameters for automation saves dozens of hours that

would otherwise be spent on an incredibly tedious, eye-straining, and carpal tunnel-inducing task.

The manually labeled data across varying widths, amplitudes, and quantities of trained data are compared in the following section. Regrettably, during the testing phase, the default values previously utilized were a width of 15 nanoseconds and an amplitude of 0.2. This particular set of hyperparameters has been demonstrated to yield the poorest performance among all tested permutations. Conversely, the best performance was achieved with a width of 6ns, effectively detecting most piled-up peaks and an amplitude threshold of 0.1 that identifies smaller amplitude peaks. Although this low amplitude adds some false positives, they accounted for less than 0.3% of all labeled peaks, which is an acceptable margin.

Consequently, an application of a width of 6 ns and an amplitude of 0.1 appears to produce nearly optimal results at this juncture. The training pipeline associated with this project does not appear to accommodate substantial volumes of data concurrently, which culminated in diminished performance when employing six times the aggregate number of peaks for training.

Twelve files, including four from all other training, were used for one test. Considering that all twelve have 607,493 peaks and the third used for the rest only amounted to 99,858 peaks, eight files have 500,000 peaks. While about 25,000 peaks each were expected, four of these eight files contain about 90,000 peaks, another three have around 50,000 peaks, and the last two meet prior expectations. Therefore, it is also important to train the network with a different number of classes.

Upon closer examination of the file names, the distinction in the total number of detected peaks is associated with the frequency utilized during the measurement:

- 25,000 peaks indicate 2 MHz
- 50,000 peaks indicate 5 MHz
- 90,000 peaks indicate 10 MHz

Width = 15 ns

A width of 15 ns and an amplitude threshold of 0.2 were the standards used for peak detection, which were applied prior to experimenting with the values. Table XVII shows how one of these changes involved amplitude:

Table XVII: Results for width 15ns at different amplitudes (%)

Variables	Manual – 4 WFM at 2 MHz	w=15ns, h=0.2	w=15ns, h=0.1
Labeled Peaks	99858	96219	96640
Predicted Peaks	97,79	90,07	95,03
Peak Difference	2,21	9,93	4,97
Total errors	20,03	22,68	21,54
Total of exact matches	24,61	23,81	24,27
Over-detection	8,91	6,38	8,29
Under-detection	11,12	16,30	13,25
Peaks over 1.2	0,28	0,29	0,29

As observed in table XVII, the labeled peaks themselves changed little. However, the peak prediction increased significantly from 90.07% to 95.05%. It is easy to assume that some of these peaks might be false positives. They need to be compared with manual labeling to confirm this: since the total number of labeled peaks differs, the percentages do not present the complete picture. For reference, table XVIII shows the values themselves:

Table XVIII: Results for width 15ns at different amplitudes

Variables	Manual – 4 WFM at 2 MHz	w=15ns, h=0.2	w=15ns, h=0.1
Labeled Peaks	99858	96219	96640
Predicted Peaks	97656	86669	91841
Peak Difference	2202	9550	4799
Total errors	20006	21826	20815
Total of exact matches	24573	22913	23458
Over-detection	8902	6138	8008
Under-detection	11104	15688	12807
Peaks over 1.2	282	279	279

The comparisons conducted with the manually labeled data unequivocally indicate that it consistently outperforms the other alternatives. When evaluating both instances at width-15ns, there remains a notable 5% enhancement, as previously observed; however, this is accompanied by a reduction of 2.6% in peak detection. Nevertheless, these figures are still considered acceptable since the focus is on determining the values that contribute to superior performance.

Width = 6 ns

The width reduction (Table XIX) clearly enhances automatic detection's reliability, enabling it to predict 99.12% of the total number of peaks identified manually. The difference in amplitude between the two automatic models is also significant, with both performing slightly better than the manual labels in terms of exact matches.

Interestingly, although the lower amplitude was trained on a more varied range of peaks, including those overlooked by the higher one, the model performed worse in peak detection despite the algorithm's better performance.

Table XIX: Results for width 6ns at different amplitudes

Variables	Manual – 4 WFM at 2 MHz (%)	w=6ns, h=0.1	w=6ns, h=0.2
Labeled Peaks	99858	98976	98549
Predicted Peaks	97,79	91,63	94,42
Peak Difference	2,21	8,37	5,58
Total errors	20,03	21,03	21,60
Total of exact matches	24,61	25,40	24,09
Over-detection	8,91	6,33	8,01
Under-detection	11,12	14,70	13,59
Peaks over 1.2	0,28	0,28	0,28

Height = 0.2

When placed in context, the discrepancy in width significantly impacts both peak detection by the algorithm and the performance of the model. Furthermore, a notable observation in table XX is the frequency with which the model identifies fewer peaks than anticipated. Conversely, it detects additional peaks less frequently.

Table XX: Results for height 0.2 at different widths

Variables	Manual – 4 WFM at 2 MHz	w=15ns, h=0.2	w=6ns, h=0.2
Labeled Peaks	99858	96219	98549
Predicted Peaks	97,79	90,07	94,42
Peak Difference	2,21	9,93	5,58
Total errors	20,03	22,68	21,60
Total of exact matches	24,61	23,81	24,09
Over-detection	8,91	6,38	8,01
Under-detection	11,12	16,30	13,59
Peaks over 1.2	0,28	0,29	0,28

Height = 0.1

Given that the total number of peaks detected with a higher width is significantly lower than those detected with a lower value, the percentage of detected peaks can be misleading (Table XXI), as it predicted 8% fewer peaks than those labeled manually.

Table XXI: Results for height 0.1 at different widths (%)

Variables	Manual – 4 WFM at 2 MHz	w=15ns, h=0.1	w=6ns, h=0.1
Labeled Peaks	99858	96640	98976
Predicted Peaks	97,79	95,03	91,63
Peak Difference	2,21	4,97	8,37
Total errors	20,03	21,54	21,03
Total of exact matches	24,61	24,27	25,40
Over-detection	8,91	8,29	6,33
Under-detection	11,12	13,25	14,70
Peaks over 1.2	0,28	0,29	0,28

By comparing the actual values in Table XXII, it is evident that utilizing the 6ns width generally performs better. Not only are the exact total matches higher but there are also fewer instances of incorrectly labeled peaks in the case of extras. However, considering that the predicted peaks are lower than in other trained models, this issue is somewhat mitigated by the roughly 1% variation in results when training the same model with the same files on different occasions.

Table XXII: Results for height 0.1 at different widths

Variables	Manual – 4 WFM at 2 MHz	w=15ns, h=0.1	w=6ns, h=0.1
Labeled Peaks	99858	96640	98976
Predicted Peaks	97656	91841	90693
Peak Difference	2202	4799	8283
Total errors	20006	20815	20819
Total of exact matches	24573	23458	25141
Over-detection	8902	8008	6268
Under-detection	11104	12807	14551
Peaks over 1.2	282	279	280

Six 2MHz WFM files with about 25,000 peaks at w = 6ns h = 0.1

Increasing the number of files utilized for training by 50% did not result in a significant enhancement in performance. Although this approach yielded improved validation accuracy when compared to manually labeled data, it concurrently led to a decrease in general accuracy by a comparable margin (Table XXIII). Consequently, the overall impact was minimal.

Table XXIII: Comparison of training results between 4 and 6 WFM files

	4 WFM files, 168 NPZ files at 2MHz	4 WFM files, 168 NPZ files at 2MHz	252 NPZ files, 6 WFM at 2MHz
Variables	**manual 30 epochs**	**w=6ns h=0.1 30 epochs**	**w=6ns h=0.1 30 epochs**
loss	0.0771	0.0723	0.1359
accuracy	0.9810	0.9831	0.9598
top_1_accuracy	0.9810	0.9831	0.9598
top_3_accuracy	1.0000	0.9995	0.9996
val_loss	3.2286	2.5673	2.5357
val_accuracy	0.4119	0.4268	0.4654
val_top_1_accuracy	0.4119	0.4268	0.4654
val_top_3_accuracy	0.7916	0.8065	0.8331

In comparing the results of the analysis script (Table XXIV), a noticeable increase in labeled peaks is expected when using more files. However, examining the difference between labeled and predicted peaks shows a performance improvement. The 4-file manual peaks had a detection rate of 94.54%, while the automatic version achieved 91.63% success; the 6-peak detected 96.02%.

Table XXIV: Comparison of evaluation results between 4 and 6 WFM files

Variable	Manual – 4 WFM at 2 MHz (%)	4 WFM at 2MHz w=6ns, h=0.1 (%)	6 WFM at 2 MHz w=6ns h=0.1 (%)
Labeled Peaks	99858	98976	144866
Predicted Peaks	97,79	91.63	96.02
Peak Difference	2,21	8.37	3.98
Total errors	20,03	21.03	18.87
Total exact matches	24,61	25.40	27.98
Over-detection	8,91	6.33	7.45
Under-detection	11,12	14.70	11.42
Peaks over 1.2	0,28	0.28	0.27

Therefore, the total number of errors decreased by more than 2% and consequently, the occurrences of over- and under-detection also declined. However, the cases of over-detection comparing both automatic labeling instances did not improve. The most notable difference between the two is that the cases of under-detection decreased by over 3% when comparing both automatic detections.

3.3.3 Testing Files with Higher Frequencies

Since filenames appear as 150MeVFrontandBack1250VSCI1100V2MHz_ch1_20241217054554735.wfm, some aspects were overlooked during the initial testing. This oversight resulted in mixing all channel 1 files regardless of their frequencies, leading to unusual outcomes. The issue arose because the processing script would fail while attempting to convert all 132 WFM files, which amounted to 5,544 NPZ files. While determining which files were processed correctly, it was observed that only channel 1 files were handled without errors. However, further testing revealed that files with 5 MHz and 10 MHz also produced unexpected results.

Breaking down the file name reveals the following information:

- **150MeV**
- Front and Back
- 1250VSCI
- 1100V
- **2MHz**
- **ch1**
- 20241217054554735

Excluding the final segment, which solely indicates the measurement's timestamp, the only values that demonstrate significant variation are those emphasized in bold. The initial value may be 150 MeV, 251 MeV, 345 MeV, or 430 MeV; the frequencies may be 2 MHz, 5 MHz, or 10 MHz, and the channels can range from 1 to 4. In this particular section, the tests are confined to the 150 MeV and channel 1 files, as these are the only configurations that operate effectively with the provided code. Section 3.4 will examine the remaining files to ascertain whether they function correctly.

Four WFM files vs. Twelve WFM files

Aggregating all Channel 1 files with varying frequencies did not yield the anticipated results (Table XXV). The reasoning behind this observation is now evident:

Table XXV: Comparison of results between 4 and 12 WFM files and 13 classes

Variables	Manual – 4 WFM at 2 MHz	w=6ns, h=0.1	12 WFM w=6ns h=0.1
Labeled Peaks	99858	98976	607493
Predicted Peaks	97,79	91,63	73,94
Peak Difference	2,21	8,37	26,06
Total errors	20,03	21,03	33,24
Total of exact matches	24,61	25,40	10,45
Over-detection	8,91	6,33	3,59
Under-detection	11,12	14,70	29,65
Peaks over 1.2	0,28	0,28	0,10

The percentage of predicted peaks experienced a substantial decline of over 20%, accompanied by a 10% increase in the proportional number of errors, while the precise count of matches diminished by more than 10%. Given that the 10 MHz files account for half of the total peaks, the average peak amplitude has concurrently decreased significantly (higher frequencies display a reduced signal-to-noise ratio attributable to shorter inter-peak intervals). This frequency also contributes to the reduction in the detection of additional peaks. Although this outcome is favorable, it merely reflects a bias associated with high-density files (the identical 4200 windows contain four times the number of peaks). Similarly, the observed increase in instances of fewer detected peaks can be traced back to files operating at a frequency of 5 MHz.

With six files at 2 MHz, three at 5 MHz, and three at 10 MHz, a total of 607,493 peaks were detected. However, this issue arose because the number of classes (maximum peaks) was not adjusted. Table XXVI shows that altering this value has significant implications: a staggering 20% difference in predicted peaks. Unfortunately, the increase in classes appears to confuse the model, resulting in a doubled error rate and a ten-fold increase in false positives, primarily due to reflections at higher frequencies being stronger and more frequent. Finding a way to implement a dynamic amplitude threshold might help resolve this issue.

Table XXVI: Comparison of results between 4 and 12 WFM files and 40 classes

Variables	Manual – 4 WFM at 2 MHz	12 WFM 13 classes	12 WFM 40 classes
Labeled Peaks	99858	607493	607493
Predicted Peaks	97,79	73,94	93,90
Peak Difference	2,21	26,06	6,10
Total errors	20,03	33,24	66,52
Total of exact matches	24,61	10,45	2,43
Over-detection	8,91	3,59	30,21
Under-detection	11,12	29,65	36,31
Peaks over 1.2	0,28	0,10	0,10

Three 5MHz WFM files with about 50,000 peaks at w = 6ns h = 0.1

An unusual occurrence occurred during this dataset's training (Table XXVII): the training concluded in under five seconds with almost no losses, and everything reached 100% accuracy.

Table XXVII: Suspiciously satisfactory results

loss: 0.0014
accuracy: 1.0000
top_1_accuracy: 1.0000
top_3_accuracy: 1.0000

Taking that suspicion into consideration, a file was accessed within the manual labeling interface to examine whether the waves conformed to the expected criteria. It became apparent that even the smaller waves of 0.8 amplitude frequently produced reflections exceeding 0.1 amplitude. Rather than undergoing a complete retraining process with an altered threshold, the script was modified to indicate the frequency of peaks exceeding 0.8 (Table XXVIII), thus allowing for an estimation of the number of false positives present.

For comparison, the script was executed while examining the number of peaks greater than 1.2, and the initial hypothesis that the peaks were smaller than those in the 2 MHz files proved correct. While the four 2 MHz files displayed 52, 57, 72, and 98 peaks of that amplitude or higher, the three 5 MHz files had a comparable count: 30, 56, and 64. Since the 5 MHz files contain double the number of peaks compared to the 2 MHz files, the occurrence of such peaks is about half as frequent.

Table XXVIII: Comparison between datasets trained on 2 and 5 MHz

Variables	Manual – 4 WFM at 2 MHz	3 WFM w=6ns h=0.1 at 5MHz	5 MHz 30 classes	5 MHz 30 classes (2nd run)
Labeled Peaks	99858	167584	167584	167584
Predicted Peaks	97,79	26,26	43,26	33,98
Peak Difference	2,21	73,74	56,74	66,02
Total errors	20,03	73,86	68,80	68,29
Total of exact matches	24,61	0,50	1,94	1,46
Over-detection	8,91	0,06	6,03	1,13
Under-detection	11,12	73,80	62,77	67,15
Peaks over 1.2	0,28	0,09	0,09	0,09
Peaks over 0.8	N/A	6,74	6,74	6,74

Considering how quickly the training concluded, the results outlined above are not surprising: nearly 74% of the windows identified fewer peaks than expected, and approximately 7% of those peaks could be false positives. Even more concerning is that only about 0.5% of the windows had exact matches. While the number of windows with additional detected peaks is small, this could be viewed as a positive—if not for the previously mentioned false positives.

By retraining the model with nearly three times the number of classes, the results improved significantly, although they remained unreliable. After training at 5 MHz with thirty classes for a second time, the predicted peaks decreased notably, even though no other factors changed.

This reinforced the notion that the models are more about guessing than actually learning from this volume of data.

Three 5MHz WFM files with about 50,000 peaks at w = 6ns h = 0.2

The files were regenerated to compare how the amplitude would affect the results (Table XXIX), but this time using 0.2 as a threshold.

Table XXIX: Comparison with datasets trained on different amplitudes with 5 MHz files

Variables	3 WFM at 5MHz w=6ns h=0.1	3 WFM at 5MHz w=6ns h=0.2
Labeled Peaks	167584	155208
Predicted Peaks	26,26	16,33
Peak Difference	73,74	83,67
Total errors	73,86	84,08
Total of exact matches	0,50	0,77
Over-detection	0,06	0,20
Under-detection	73,80	83,88
Peaks over 1.2	0,10	0,10
Peaks over 0.8	6,74	N/A

The initial parameter clearly indicates a notable differentiation in the 5 MHz peaks compared to the 2 MHz peaks. Previously, an alteration in amplitude resulted in an increase of approximately 200 to 400 total peaks detected, which equates to a range of 3 to 5%. In contrast, the current analysis presents a remarkable 10% difference in peak detection. When evaluating this figure alongside the peaks exceeding an amplitude of 0.8, it is reasonable to postulate that the earlier hypothesis may indeed have been accurate, as these peaks constitute 6.74% of the dataset. Nevertheless, while the number of additional peaks detected has risen as anticipated, the significance of this increase remains relatively moderate.

Three 10MHz WFM files with about 90,000 peaks at w = 6ns h = 0.1

Once again, an issue arose that required accessing the manual labeling interface to investigate what was occurring. This made it clear that the amplitude of the peaks at 10 MHz is significantly smaller than at the other tested frequencies, to the extent that out of 267,071 peaks detected, none exceeded an amplitude of 1.2. In contrast, at lower frequencies, an average of about 300 for every 100,000 seems to be expected.

While the average 2 MHz peak reaches an amplitude of 0.4, and a 5 MHz peak has a slightly higher average of 0.5, a 10 MHz peak is around 0.15. The average 10 MHz reflection varies greatly between 0.03 and 0.08. Therefore, a threshold of 0.1 seems appropriate. Although many small distortions are lost this way due to the smaller waves, it becomes increasingly difficult to differentiate them from the usual noise occurring near amplitude zero.

While slicing examples from the training and evaluation data, no windows with peaks between 0 and 5 were found, and only after 10 peaks did a two-digit value appear. A similar occurrence happened when slicing 5 MHz files for training; only the 0 and 1 peaks were missing, and after reaching 3 peaks, the values exceeded two digits already. The problem mentioned earlier is likely related to this low number of sample data, as the training script produces the error "**ValueError: Unexpected result of train_function (Empty logs)**" when attempting to use this data. This may explain why training all frequencies together produced unexpected results.

Even if training a model with 10 MHz is impossible, the analysis script could still gather the peak data identified in these measurements (Table XXX). Here, the captured max-amplitude data was altered. After examining the shapes of the peaks, it appears that lowering the amplitude threshold to 0.05 would yield a similar result as changing the threshold from 0.2 to 0.1. Since peaks higher than 0.2 generate the larger reflections that are expected, that is the new script adjustment that will be implemented below.

Table XXX: Comparison between 2 and 10 MHz results

Variables	Manual – 4 WFM at 2 MHz	3 WFM w=6ns h=0.1 at 10MHz
Labeled Peaks	99858	267071
Predicted Peaks	97,79	3,92
Peak Difference	2,21	96,08
Total errors	20,03	96,08
Total of exact matches	24,61	0,00
Over-detection	8,91	0,00
Under-detection	11,12	96,08
Peaks over 1.2	0,28	0,00
Peaks over 0.2	N/A	47,98

Once more, it is essential that each frequency is accompanied by a dedicated training set to ensure that the network can operate effectively with them. Notably, only 3.92% of the peaks were accurately identified, and there were no precise matches observed. Furthermore, the differences in peaks, the number of errors, and the number of instances in which fewer peaks were detected all remain consistent at 96.08% of the cases, which is diametrically opposed to expectations. Lastly, it is noteworthy that nearly fifty percent of the peaks exceed an amplitude of 0.2, rendering the distinction between reflections and actual peaks at this frequency particularly challenging.

Comparison of 2, 5, and 10 MHz WFM files at w = 6ns h = 0.1

Initially, it should be noted that including 2 MHz files in the dataset facilitates proper model training. This occurs when all files are used collectively, as well as in combinations of 2 and 5 MHz and 2 and 10 MHz. It is not surprising that the simultaneous use of both 5 and 10 MHz results in an error, causing the model to be “trained" for only a few seconds during the entire 30 epochs. Consequently, one can infer that the 10 MHz model is unlikely to initiate training due to the insufficient number of cases presenting minimal or no peaks.

By analyzing all frequencies separately, the table below confirms what was expected: the average amplitude of the waves decreases with each frequency range. This appears to result in fewer exact matches found and more instances where the model over-detects peaks. Conversely, the instances where the model detects fewer peaks than anticipated approach 100%.

Although this was not the case previously, at higher frequencies, most errors appear to stem from an inability to detect enough peaks – the very pile-ups that this project aims to identify. With the increased frequency, a rise in presumed false positives becomes evident: the last three rows of Table XXXI indicate the amplitudes at which the waves of each measured frequency tend to create reflections exceeding the amplitude threshold.

Table XXXI: Comparison between 2, 5, and 10 MHz results

Variables	6 WFM w=6ns h=0.1	3 WFM w=6ns h=0.1 at 5MHz	3 WFM w=6ns h=0.1 at 10MHz
Labeled Peaks	144866	167584	267071
Predicted Peaks	96,02	33,98	3,92
Peak Difference	3,98	66,02	96,08
Total errors	18,87	68,29	96,08
Total of exact matches	27,99	1,46	0,00
Over-detection	7,45	1,13	0,00
Under-detection	11,43	67,15	96,08
Peaks over 1.2	**0,27**	**0,09**	**0,00**
Peaks over 0.8	N/A	**6,74**	N/A
Peaks over 0.2	N/A	N/A	**47,98**
Peaks over 0.25	N/A	N/A	**10,41**
Peaks over 0.3	N/A	N/A	**0,37**

At a frequency of 10 MHz, due to the relatively short wavelengths, an amplitude of 0.20 suffices to produce reflections. However, these reflections constitute approximately half of the total waves, with the remainder being of a lower amplitude. To substantiate this assertion, Table XXXII has been constructed, illustrating that only 10.41% of the waves correspond to an amplitude of 0.25. This indicates that employing standard values of 15 ns and an amplitude of 0.2 for automatic measurements would neglect over 50% of the waves in the MHz range.

Table XXXII: Number of peaks per amplitude threshold in 10 MHz frequency

Peaks over 0.20	47,98
Peaks over 0.21	35,96
Peaks over 0.22	28,97
Peaks over 0.23	22,17
Peaks over 0.24	15.54
Peaks over 0.25	10,41
Peaks over 0.26	6,53
Peaks over 0.27	3,81
Peaks over 0.28	2,03
Peaks over 0.29	0,92
Peaks over 0.30	0,37

Mixing 2 and 5 MHz eight WFM files at w = 6ns h = 0.1

There appears to be a distinct advantage in utilizing training files of varying frequencies. While one might anticipate that the results would cumulatively contribute to one another, it can be posited that the files exhibiting superior visibility facilitated the identification of a greater number of files with higher frequencies. For instance, an accuracy level of approximately 63% for peak prediction was initially projected. However, the accuracy ultimately achieved was 90%. Although the number of exact matches decreased (Table XXXIII), the errors incurred were also lower than anticipated, consistently maintaining their levels at 2 MHz.

Table XXXIII: Results from mixing 2 and 5 MHz frequencies

Variables	6 WFM w=6ns h=0.1 at 2MHz	3 WFM w=6ns h=0.1 at 5MHz	9 WFM w=6ns h=0.1 at 2+5MHz	9 WFM 30 classes
Labeled Peaks	144866	167584	**312450**	312450
Predicted Peaks	96,02	26,26	**87,60**	101,21
Peak Difference	3,98	73,74	**12,40**	1,21
Total errors	18,87	73,86	**22,87**	59,46
Total of exact matches	27,99	0,50	**17,83**	4,84
Over-detection	7,45	0,06	**5,23**	30,33
Under-detection	11,43	73,80	**17,63**	29,12
Peaks over 1.2	0,27	0,09	**0,18**	0,18

However, this can only be said when you compare the same number of classes. By increasing the detected peak quantity per window from 13 to 30, various errors arise, including instances where more peaks are labeled than were supposed to exist. Typically, one would aim for 100% prediction, but only under the condition that the errors are also 0%.

Mixing 2 and 10 MHz eight WFM files at w = 6ns h = 0.1

The same result applies to mixing 2 and 10-MHz files (Table XXXIV). However, while the positive impact of 2 MHz is evident to anyone, the adverse influence of 10 MHz cannot be entirely disregarded in this case. Therefore, training enough 2 MHz files to achieve 100% validation accuracy may assist a model in identifying 5 MHz files. Still, it appears that 10 MHz necessitates its own labeling to help the model distinguish between the waves. Additionally, transitioning from 13 to 40 classes did not yield significant improvement: since the error rate remained unchanged and new errors were introduced, the previous statement still stands.

Table XXXIV: Results from mixing 2 and 10 MHz frequencies

Variables	6 WFM w=6ns h=0.1 at 2MHz	3 WFM w=6ns h=0.1 at 10MHz	9 WFM w=6ns h=0.1 at 2+10 MHz	40 classes
Labeled Peaks	144866	267071	**411937**	411937
Predicted Peaks	96,02	3,92	**68,04**	80,08
Peak Difference	3,98	96,08	**31,96**	19,92
Total errors	18,87	96,08	**37,25**	35,80
Total of exact matches	27,99	0,00	**10,64**	6,16
Over-detection	7,45	0,00	**2,65**	7,94
Under-detection	11,43	96,08	**34,60**	27,86
Peaks over 1.2	0,27	0,92	**0,10**	0,10

Mixing 5 and 10 MHz six WFM files at w = 6ns h = 0.1 and h = 0.2

This test exhibits a degree of redundancy; although certain improvements are discernible, the persistent patterns of zero exact matches alongside the consistent occurrence of errors remain evident (Table XXXV). The increase in classes from 13 to 40 exacerbates the issue, leading to the labeling of 538,615 peaks, despite the actual existence of only 434,655. Modifying the amplitude

threshold to 0.2 yields enhanced results; however, nearly half of the classes must now be utilized. Nonetheless, despite achieving a 39.79% error rate, the reported 96.17% accuracy still suggests a considerable presence of both false positives and false negatives.

Table XXXV: Results from mixing 5 and 10 MHz frequencies

Variables	3 WFM w=6ns h=0.1 at 5MHz	3 WFM w=6ns h=0.1 at 10MHz	6 WFM w=6ns h=0.1 at 5+10MHz	40 classes	Height = 0.2 26 classes
Labeled Peaks	167584	267071	**434655**	434655	178487
Predicted Peaks	26,26	3,92	**52,05**	123,92	96,17
Peak Difference	73,74	96,08	**47,95**	23,92	3,83
Total errors	73,86	96,08	**49,55**	60,07	39,79
Total of exact matches	0,50	0,00	**1,74**	2,69	6,84
Over-detection	0,06	0,00	**0,80**	42,00	17,98
Under-detection	73,80	96,08	**48,75**	18,08	21,81
Peaks over 1.2	0,09	0,92	**0,03**	0,03	2,33
Peaks over 0.8	N/A	N/A	**2,6**	2,6	178487

3.3.4 Testing Files from Experimental Data

This section is dedicated to elucidating the types of files compatible with the provided code and the specific conditions they must fulfill. Previously, this project focused on files from Channel 1 (Ch1) at 150 MeV, which are regarded as the standard files for measurements. Channels 2, 3, and 4 are classified as experimental files. While this section does not significantly contribute to the results concerning the target data, it may serve as a reference for future endeavors. It remains uncertain whether higher voltages should also be classified as experimental; however, circumstantial evidence suggests this possibility.

This analysis examines all remaining channels for 150 MeV voltage and all channels for 251, 345, and 430 MeV.

Since the results at each voltage only showed sufficiently relevant differences when comparing the channels with one another, they will be presented accordingly. Additionally, it's important to note that only 150 and 430 MeV had the full range of frequencies, while the others had only two available: 345 MeV had 5 and 10 MHz, while 251 MeV had 10 MHz and an unknown frequency (the file name did not provide that information). For clarification, instead of testing all 132 available files, one was tested for each frequency, totaling about 37 files.

Channel 1

The peaks in ch1 at higher voltages appear well-defined but have an average amplitude of -0.15 instead of -0.4, as seen in Figure 14.

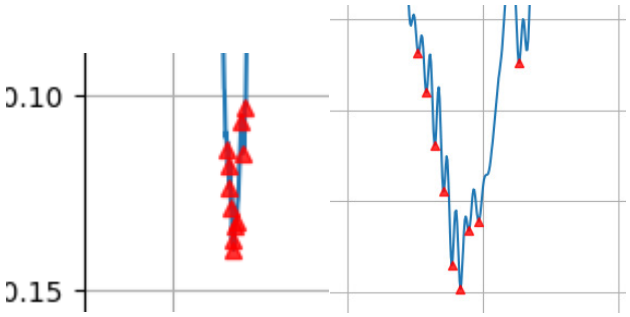


Figure 14: Peaks in Other Voltages [Labeling UI].

Given that this characteristic bears similarities to the 10 MHz feature observed at 150 MeV, it may present certain challenges during the training process. Specifically, an error occurs when attempting to initiate the training of a model using 350 MeV, preventing the process from commencing. Conversely, training at 430 MeV proceeds akin to the 5 MHz data from 150 MeV (despite a 2 MHz dataset being utilized in this instance), resulting in the completion of all 30 epochs in an impressively swift manner.

Channels 2 and 3

For channels two and three across all frequencies, the peaks appear to be approximately 0.002 in amplitude (Figure 15), thereby complicating the differentiation between noise and actual peaks. Automatic labeling is likely to present significant challenges due to this issue. Furthermore, higher peaks, ranging from 0.1 to 0.4 in amplitude, may occasionally be detected, albeit with a low frequency of occurrence.

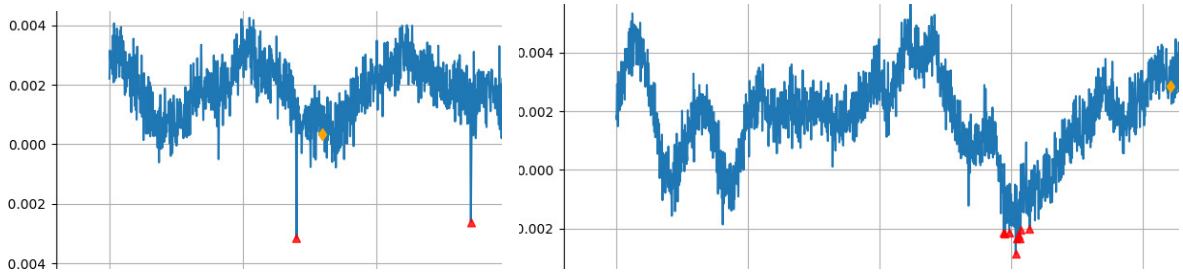


Figure 15: Peaks of ch2 and ch3 [Labeling UI].

Channel 4

With respect to the ch4 files, it is confirmed that none of them were capable of processing raw WFM files, as anticipated (Figure 16).

```
Traceback (most recent call last):
  File "C:\Users\allan\Desktop\BA\particle-counting-re\src\ProcessRawWFM.py", li
ne 253, in <module>
    print("all_peaks", np.shape(all_peaks), np.min(all_peaks), np.max(all_peaks)
)
  File "<__array_function__ internals>", line 180, in amin
  File "C:\Users\allan\AppData\Local\Programs\Python\Python310\lib\site-packages
\numpy\core\fromnumeric.py", line 2916, in amin
    return _wrapreduction(a, np.minimum, 'min', axis, None, out,
  File "C:\Users\allan\AppData\Local\Programs\Python\Python310\lib\site-packages
\numpy\core\fromnumeric.py", line 86, in _wrapreduction
    return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
ValueError: zero-size array to reduction operation minimum which has no identity
```

Figure 16: Error when processing [Command line].

Since the files cannot be processed, verifying how they were meant to appear is impossible, as processing them is a prerequisite for accessing the manual labeling interface and visualizing the waves.

Training all ch1 files together

Once all Channel 1 files were deemed suitable for training, the hypothesis positing that an increased volume of data could enhance the model's performance was rigorously evaluated (Table XXXVI). Given that most of the files exhibited characteristics akin to a poorly performing model, the results appeared unpromising. Ultimately, the outcome was nearly identical to that of the model trained exclusively with 5 and 10 MHz files at Channel 1 and 150 MeV when utilizing 13 classes. Nevertheless, this quantity was also analyzed, as it had been observed that utilizing 40 files was feasible with 10 MHz.

It is noteworthy that the quantity of examples gathered for higher peak counts was considerable; thus, it is reasonable to infer that the highest class resides within the range of 50 to 60. Nevertheless, given that the peak prediction escalated while the error rate remained virtually unchanged, undertaking the entire two-hour procedure of regenerating the files with an amplitude of 0.2, producing training data, and conducting the training itself, along with the extensive evaluation process, does not appear promising, particularly in light of the most recent test results obtained prior to this assessment.

Table XXXVI: Results from training all Channel 1 files (%)

Variables	12 WFM w=6ns h=0.1	6 WFM w=6ns h=0.1 at 5+10MHz	33 WFM ch1	33 WFM ch1 - 40 classes
Labeled Peaks	607493	434655	2506767	2506767
Predicted Peaks	93,90	52,05	49,49	108,74
Peak Difference	6,10	47,95	50,51	8,74
Total errors	66,52	49,55	52,25	43,59
Total of exact matches	2,43	1,74	2,82	4,93
Over-detection	30,21	0,80	0,87	26,17
Under-detection	36,31	48,75	51,38	17,42
Peaks over 1.2	0,10	0,03	0,05	0,05
Peaks over 0.8			0,99	0,99
Peaks over 0.25			42,71	42,71

In light of the considerable volume of data utilized, the comprehensive figures can be found in Table XXXVII. What's surprising is that with just under three times the amount of data as before, this model could label more than four times the number of peaks. The label height data shows that very few reached high values while most remained low. This reinforces the need to train the model on different frequencies separately.

Table XXXVII: Results from training all Channel 1 files

Variables	12 WFM w=6ns h=0.1	6 WFM w=6ns h=0.1 at 5+10MHz	33 WFM ch1	33 WFM ch1 - 40 classes
Labeled Peaks	607493	434655	2506767	2506767
Predicted Peaks	570428	226221	1240652	2725972
Peak Difference	37065	208434	1266115	219205
Total errors	404093	215370	1309871	1092717
Total of exact matches	14787	7582	70787	123569
Over-detection	183514	3468	21878	655961
Under-detection	220579	211902	1287993	436756
Peaks over 1.2	620	150	1163	1163
Peaks over 0.8			24845	24845
Peaks over 0.25			1070735	1070735

3.4 Experimenting with Window Sizes

Until now, modifications have been made regarding the files utilized and the optimal parameters for implementing automatic labeling. However, before the completion of this project, there remains another avenue to explore: assessing whether alterations in window sizes affect the performance of peak detection.

Most scripts in the pipeline reference window size, maximum peaks, width, and height. Therefore, it was impractical to track which values were changed and their current states. Given the number of tests to run, creating a new script to manage these variable changes automatically—only as needed—was easier than altering the values directly in the script.

The inception of the "pipeline control" script was derived from the necessity to modify script values and access them efficiently. This utility facilitates the creation of NPZ files, activates manual labeling, and establishes width and height parameters for automatic labeling. Furthermore, it generates training data and subsequently trains a model based on the revised window dimensions. The script also invokes the evaluation module, conducts a statistical analysis, and ultimately adjusts the values before making a definitive selection. At each stage of the procedure, the current values, along with their corresponding types, are displayed to enhance the debugging process.

The script uses the default values (Table XXXVIII) that are already implemented elsewhere:

Table XXXVIII: Default values for the parameters used in training models

window_size = 12500
max_peaks = 13
window_amount = 100
pulse_width_ns = 15
height = 0.2

These tests will be limited to using the WFM files designated for manual labeling since it is already established that they are effective and what results can be anticipated from them. Altering the variable would mean that instead of averaging six hundred peaks from an NPZ file

divided into one hundred segments, they would be divided by the new value. Regrettably, the correlation is not as unequivocal as one might anticipate. For instance, while a window size of For example, while a window size of 12500 is needed for a hundred of them, the values in Table XXXIX are required for other cases:

Table XXXIX: Equivalency between window quantities, sizes, and their max peaks

Window quantity	WINDOW_SIZE	Assumed max_peak
16	78125	38~76
32	39062	19~38
64	19531	10~20
128	9765	5~10
256	4882	3~6
512	2441	2~4
1024	1220	1~2
2048	610	1~2
4096	305	1~2

If peaks were to be divided equally among all the windows, they would be represented by the "max_peak" value in the table. However, that value would be sixty for one hundred windows in an average file with six hundred peaks. These values are only theoretical, as this amount ranges from 300 to 1200 and may be spaced unevenly rather than proportionally distributed. In the analysis script, a counter was added to display the current window quantity (with a maximum of 16,800) and a check that compares the current saved value to the number of peaks found in the current window, ultimately showing the highest value at the end. When using standard configurations, the maximum value was 20. For clarification, this number also indicates how many classes are available for training.

The values corresponding to powers of two will be utilized for the number of windows into which the peaks are divided, commencing with a value of 16. It is anticipated that isolating the peaks will enhance the model's comprehension of the expectations placed upon it. Nonetheless, owing to the previously mentioned ambiguity, two sets of tests will be conducted: one set will involve varying the window quantities for each NPZ file, beginning with 16, designated as window size 78125. The second set will focus on the window sizes themselves adhering to this rule, which will result in nearly inverse quantities, fluctuating by a few thousand (for instance, a window size of 16 would correspond approximately to an amount of 70,000 windows).

For this second test, changes were made to the pipeline, altering many aspects of its functionality. However, since it was provided extremely late in the development process, remaking all previous tests with it would not be feasible. Therefore, the next set of tests will utilize the older pipeline to evaluate all models trained while also presenting these new types of results and comparing both. The older method will be referred to as "method one," and the newer one as "method two."

It can be anticipated that the utilization of manually labeled peaks resulted in the old method identifying approximately 93,000 predicted peaks. In contrast, training the new pipeline with these files yielded 102,000 predicted peaks out of 98,000, indicating a significant difference. Furthermore, the previous evaluation method assessed each individual window, which is highly impractical when evaluating tens of thousands of windows, as it requires several hours for a

complete review. Conversely, the new method utilized within the pipeline employs only the minimum number of available validation examples. For instance, with a window size of 32, despite containing over one million examples of zero peaks and over 20,000 examples of one peak, the method applies the same value of 418 for all classes. Given that higher classes may have as few as single-digit examples, the results appear to be rather constrained, to say the least.

3.4.1 Pipeline Control Script

The necessity to modify several variables across multiple scripts proved to be an inconvenience. The aforementioned "pipeline control" was refined over a period of two weeks, and the two most significant menus for your review are listed below.

The starting menu (Figure 17) reads the values of the variables saved in a text file and displays them to the user, who must then decide whether to keep them, revert to the default values, or select them manually.

```
Values from the file:
    - window_size: 12500
    - max_peaks: 13
    - window_amount: 100
    - pulse_width_ns: 6
    - height: 0.1

What would you like to do?

1) Use SAVED values
2) Use DEFAULT values
3) CHANGE the values

The default values are:
    - window_size: 12500, max_peaks: 13, window_amount: 100, pulse_width_ns: 15, height: 0.2

Input:
```

Figure 17: Starting Menu [Command line].

When the user wishes to change values, the next menu appears (Figure 18). Here, they can select the WINDOW_SIZE value by inputting the number of windows they want to view when opening the NPZ file. Suppose any value that is a power of two between 16 and 4096 is provided. In that case, a match-case statement predefines the number of "max_peaks" based on the test when generating data with all the specified window quantities.

Later in development, it was decided to also include the option for users to input their own values for testing purposes. With ***window_size = max(1, (12500 * 100) // int(temp_input))***, the script calculates the value necessary for it to integrate with the rest of the pipeline. For example, if 4096 were not predetermined, it would yield 305 after the calculation. Regarding the assessment of raw window sizes in place of window quantities, the match-case statements were commented out, and it was ensured that the input received from the prompt was accurately cast into an integer.

One issue that consumed an entire week was figuring out why changing WINDOW_SIZE in Script B (from Script A) had no effect when accessed in Script C – the script continued to use the hardcoded value instead of the updated one. The solution was to find and replace the value directly in the script; this way, the value would always be correct. However, this led to the

secondary effect that now the script for manual labeling displays the window size of the opened data based on this value, so it needs to be adjusted according to what you want to see.

When it comes to pulse width in nanoseconds and amplitude threshold, the user has four options featuring permutations of 6 and 15 for width and 0.1 and 0.2 for amplitude. Additionally, there is an option to manually select other values. The final option, apart from using the defaults or proceeding, is designed to assist with testing. Even though data generation often finds examples for a specific number of peaks, it is necessary to use a lower value to allow training to commence without errors. Above the input prompt, the value for AMOUNT_OF_CLASSES from the most recently trained model is displayed. This value is detected automatically and reflects "max_peaks + 1" (to account for example slices with no peaks).

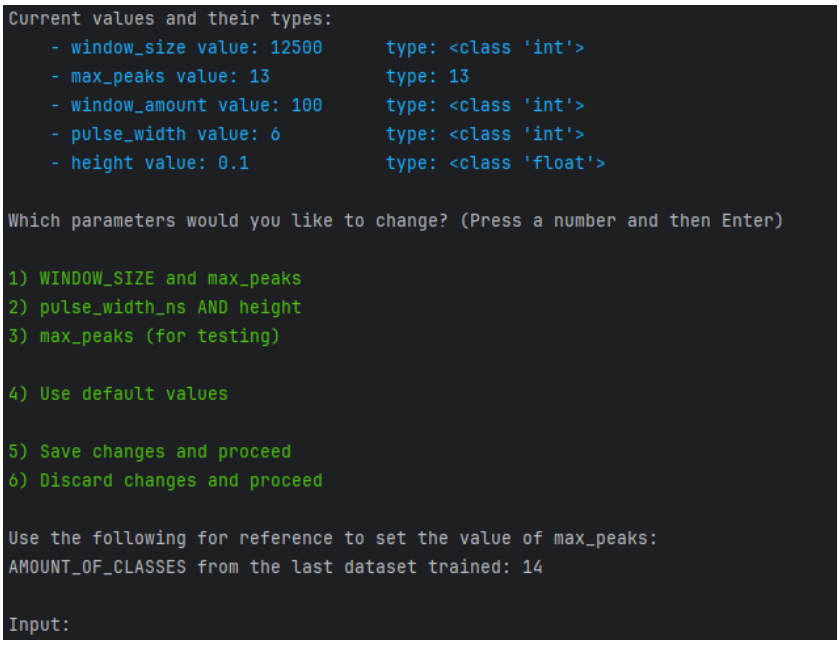


Figure 18: Menu for Changing Values [Command line].

Regarding the main menu (Figure 19), since the variables have already been written to the text files and assigned in the script itself, the values displayed are not from the file but from the script, which makes it possible to identify their types. Early in development, there were errors when Python was not inferring the correct types, which resulted in printing everything for debugging purposes.

Running the script without interruption while using cached data for WFM file processing became problematic after generating training data and during the training itself. For example, if one creates new NPZ files while overwriting the old ones, choosing the option to generate training data will somehow use the old, already overwritten files, causing confusion and delaying development.

Since manually opening the same three folders and deleting all files multiple times in a row quickly became tedious, automating the process by providing the file paths for deleting NPZ data seemed more efficient. Additionally, a new option for utilities was to split the files into 80% for training input data and 20% for validation input data. This method uses the index of the files in the folders to perform the calculation, so it simply involves assigning the first segment of the converted WFM files to one folder and the second segment to another.

```
Current values and their types:
    - window_size value: 12500      type: <class 'int'>
    - max_peaks value: 13           type: 13
    - window_amount value: 100      type: <class 'int'>
    - pulse_width value: 6          type: <class 'int'>
    - height value: 0.1             type: <class 'float'>

Which script would you like to run? (Press a number and then Enter)
    NOTE: Options 1, 4, 5 end the process to avoid using cached data on 4, 5, 6, 7

- DATA PREPARATION
    1) ProcessRawWFM.py
    2) ManualLabeling.py
    3) AutomaticLabeling.py
- MODEL TRAINING
    4) TrainingDataGeneration.py
    5) TrainingExample.py
- DATA ANALYSIS
    6) EvaluateDatasetWithModel.py
    7) Stats.py
- UTILITIES
    8) CHANGE TRAINING VALUES
    9) Show current parameter values
    10) Split .NPZ files 80/20 for training
    11) DELETE ALL .NPZ FILES

(Choose 0 to exit the program during choices)
Input:
```

Figure 19: Main Menu [Command line].

3.4.2 Comparison of Model Results

The idea of comparing different window sizes and quantities is that each size accommodates a different number of peaks. With 16 windows, there are 49 peaks on the same graph. However, after 1024 windows, only two peaks are present in the same window, with the other two in the next one. On 4096 windows, there is already an empty window between the same four peaks. For illustrative purposes, Figures 20 to 22 demonstrate the presentation of the same sequence of peaks across various window sizes.

With the baseline 100 showing seven peaks within the same window, starting from 16 and reaching 49 peaks, the following increments are: 20, 10, 6, 4, 4, 2, 2, and 2. When compared to the theoretical numbers of 81, 41, 20, 10, 5, 3, 1, 1, and 1, finding examples for the higher total peaks in a single window can be challenging. While slicing an example for training data generation, the script found only one instance with 78 peaks in a single window.

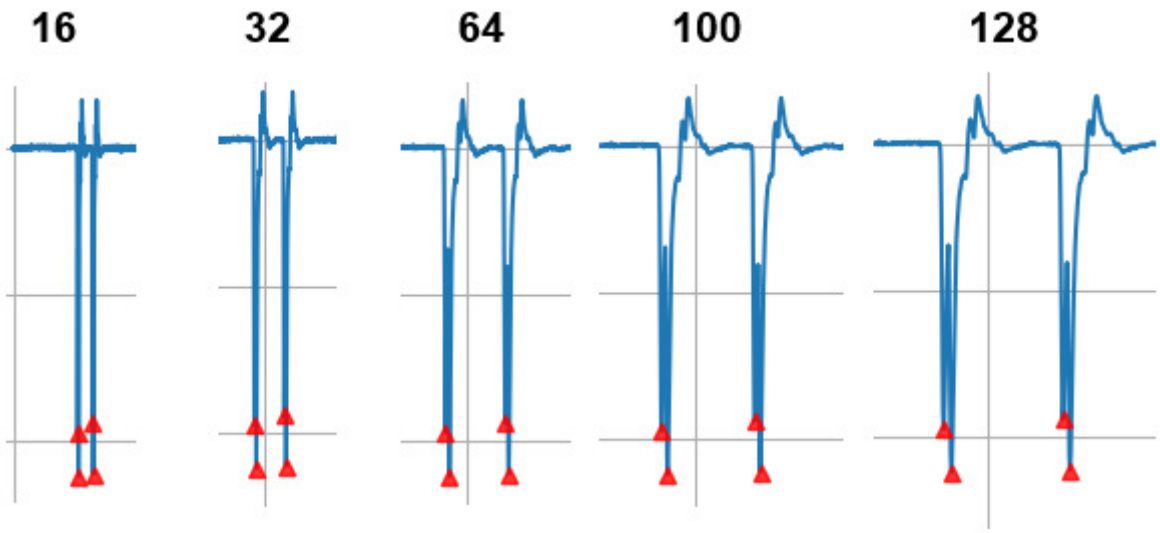


Figure 20: 16, 32, 64, 100, and 128 windows [Labeling UI].

After generating training data, max_peaks would always be adjusted to determine which value would yield the best model performance during training. One factor to consider is the number of training steps each of the 30 epochs presented when changing that variable. Since the files with manually labeled data contained about thirty steps, the highest value with comparable results was examined. Even though the hypothetical numbers mentioned earlier are close to the actual values, they do not always have examples for them, or if they do, the training would not commence.

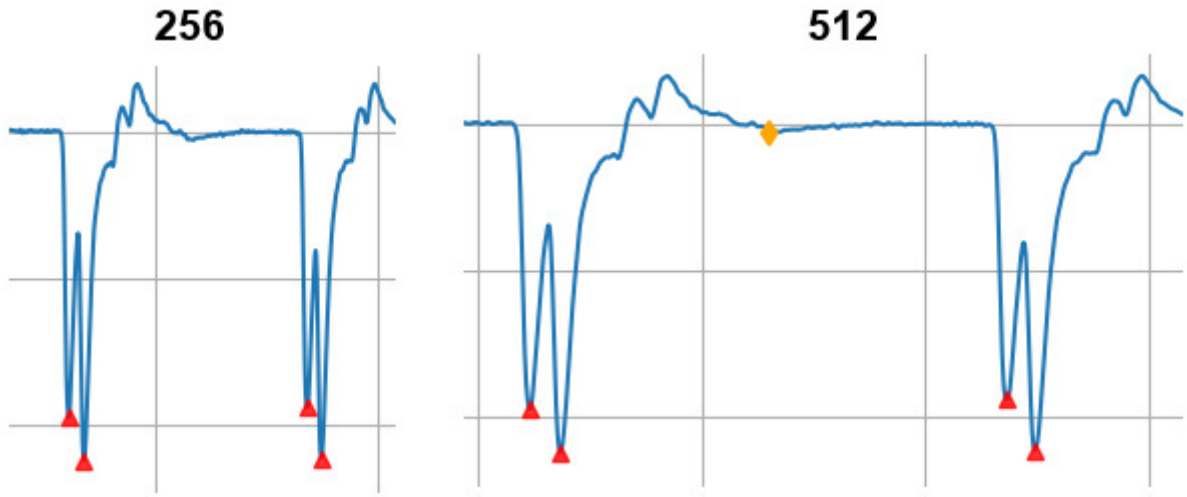


Figure 21: 256 and 512 windows [Labeling UI].

The default parameter of “max_peaks = 13" is utilized for 100 windows within a singular NPZ file. The values employed, commencing from 16 and ascending to 4096, included: 78, 40, 20, 12, 8, 5, 3, 2, and 2.

That may not be the case, but coincidentally or not, the term "window size" appears to have been used in a different context in [1]: the duration in nanoseconds displayed in the plotted graph. This conjecture arises from running the script provided by GSI at the outset of the project, which plotted images of singular peaks within 200 ns. According to his project, he employed a

size of 200 ns while training his CNNs. The default value that this project's pipeline started with was 12.500, which displays 4.000 ns in one graph. For reference, 200 ns would have 2048 windows in one NPZ file, while 25.000 ns would have 16 windows in a single file. As shown below, with 4096, the samples begin to resemble noise since the wave descends more diagonally compared to other sizes.

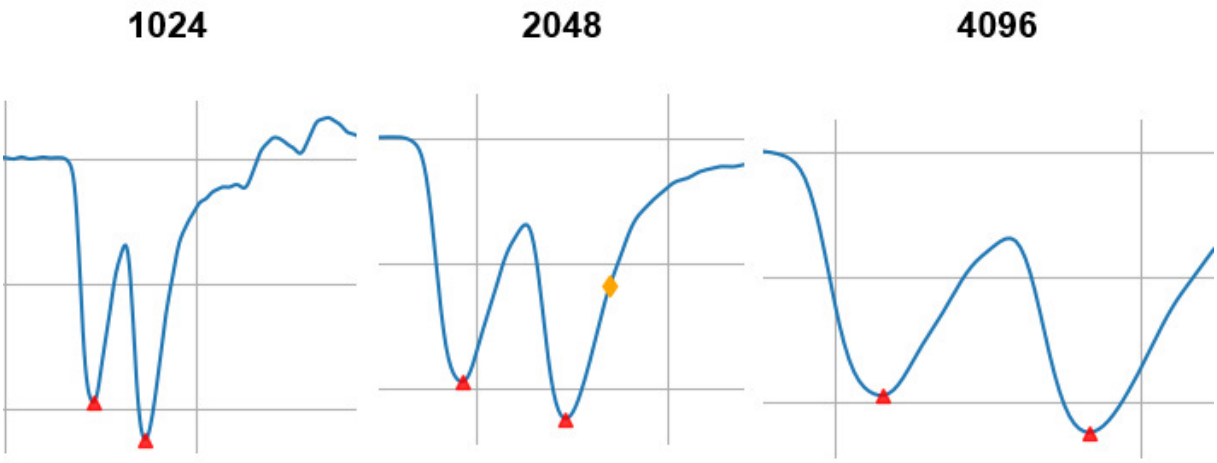


Figure 22: 1024, 2048, and 4096 windows [Labeling UI].

Training Performance (window quantity)

The table below summarizes the results of all window quantities along with their maximum number of peaks found in one window and the training steps required for that result. In red, there is an exception: even though 64 windows were able to train a model, all tests with max_peaks between 10 and 20 resulted in about four steps, while a lower number of max_peaks also decreased the steps taken, which is usually the opposite of what happens.

For instance, with 128 windows between 10 and 13, the max peaks were 69, 45, 25, and 13 consecutively. Using 100 windows as a baseline with about 30 steps, selecting 25 yields the best results because it is not only closest to 30 compared to 45, but it also features a higher class count, which is desirable.

The results for the values of 16 and 32 are absent in Table XL due to the occurrence of a "math domain error" during the model training process. Upon further investigation, it was determined that the error originated from the library utilized, which performs operations involving the logarithm base ten. Although a value of 64 was processed successfully, albeit with difficulty, there was minimal motivation to devote additional time to rectifying the error (one week was deemed sufficient). Additionally, the failure to train with the value of 16 is attributable to the same issue experienced with the 10 MHz files: within the range of 0 to 6-peak windows, no examples are available, which alone is sufficient to disrupt the training script.

As a refresher, the window sizes that correspond to the window quantities inside the NPZ file are: 19531 (64), 9765 (128), 4882 (256), 2441 (512), 1220 (1024), 610 (2048), and 305 (4096).

Table XL: Training results in different window quantities

Window Quantity	Max Peak	Training Steps	Accuracy	Loss	Top 3 Accuracy	Val Accuracy	Val Loss	Val Top 3 Acc
Manual	13	30	98.10%	0.0771	100.00%	**41.19%**	32.286	79.16%
100	13	29	98.81%	0.0467	99.97%	**41.44%**	36.394	79.90%
64	20	4	100.00%	0.0110	100.00%	**11.67%**	39.070	36.67%
128	12	25	97.82%	0.0939	100.00%	**45.09%**	27.255	84.13%
256	8	20	89.52%	0.2991	99.73%	**67.72%**	0.8194	98.51%
512	5	27	94.17%	0.1605	100.00%	**84.42%**	0.3974	99.89%
1024	3	63	99.10%	0.0235	100.00%	**98.60%**	0.0351	100.00%
2048	2	81	99.99%	0.00048	100.00%	**99.97%**	0.00086	100.00%
4096	2	81	100.00%	0.0000153	100.00%	**99.94%**	0.00093	100.00%

Each time a model is trained, the values can change by a few percentage points. Since the command line output for the results appears as follows: “accuracy: 0.9810 - loss: 0.0771 - top_1_accuracy: 0.9810 - top_3_accuracy: 1.0000 - val_accuracy: 0.4119 - val_loss: 3.2286 - val_top_1_accuracy: 0.4119 - val_top_3_accuracy: 0.7916” (values from the manual training), the table was modified to adjust percentages for better readability. The values for top 1 accuracy and top 1 validation accuracy were excluded to ensure the table fits the page, as they are identical to “accuracy” and “validation accuracy.”

Table XL indicates that validation accuracy improves when the number of peaks is represented in a single graph. Fewer classes (max_peaks + 1) enhance overall accuracy, and the loss during training also decreases significantly.

Given the nature of the search for piled-up peaks, the more classes there are, the better; thus, 512 windows per file seemed to offer one of the best performances in the given circumstances. In an ideal scenario, with sufficient data available, one can assume that a file with 4096 windows could potentially have about six classes. This number should increase significantly if higher voltages (Figure 23) are adapted to work with this project’s pipeline in the future.

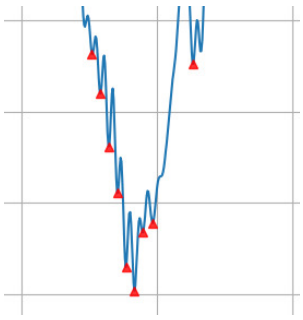


Figure 23: High voltage wave [Labeling UI].

Another important point to consider regarding the reduced number of classes leading to better results is that the number of available examples increases exponentially. The values for the number of examples from 16 to 4096 (e.g., 100 had 435) were: 1, 12, 198, 258, 306, 190, 2686, 5156, and 5157. For comparison, let us show the same class for all tested values: 0, 0, 0, 821, 1333, 4352, 14129, 36740, 67718, and 67774 (the first non-zero values come from manual testing). The analysis of the information suggests that the manipulation of data, conducted in a

manner that selectively filters the requisite learning components for the model, significantly enhances its effectiveness.

It was noted in Table XLI that the algorithm responsible for detecting peaks remained unaffected by the alteration in size. Conversely, augmenting the number of windows, which entails a reduction in window size, exhibited both advantageous and detrimental effects within the model, although the majority of these impacts were negative.

The positives for 1024, 2048, and 4096 are related to the exact matches (Table XLI). Since the model handles peaks better without pile-ups, the fact that there are fewer windows containing them, as they are more spread apart, causes this value to increase. This is inferred from Table XL, which indicates it is equivalent to the validation. Another positive trait (Table XLI) is the reduction in error from over-detection, which again results from having fewer of them together. Their negatives include the significant rise in errors, reinforcing the belief that the models struggle to detect pile-ups. Since those windows are primarily made up of single waves, each time a pile-up occurs and goes unrecognized, it is classified as an error because there are fewer peaks than labeled.

Table XLI: Evaluation results for different window quantities

Variables / Window quantities	Manual	100	64	128	256	512	1024	2048	4096
Labeled Peaks	99858	98976	98975	98966	98955	98955	98913	98963	98963
Predicted Peaks %	97,79	91,63	82,39	99,34	96,17	94,17	**77,31**	**44,67**	**64,89**
Peak Difference %	2,21	8,37	17,61	0,66	3,83	5,83	22,69	55,33	35,11
Total errors %	20,03	21,03	41,96	22,64	22,41	20,66	**31,01**	**58,39**	**39,97**
Total of exact matches %	24,61	25,40	6,61	28,30	**45,17**	**59,16**	**57,96**	**39,59**	**57,79**
Over-detection %	8,91	6,33	12,17	10,99	9,29	7,41	**4,16**	**1,53**	**2,43**
Under-detection %	11,12	14,70	29,78	11,65	13,12	13,25	**26,85**	**56,86**	**37,54**

Overall, the 512 model appears to be the best performer: it not only matched the manually labeled peaks but nearly tripled the number of exact matches while keeping the same overall error rate. Though the 128 model can also be considered the best since it achieved 5% more predicted peaks than the manual count, the other values did not show significant changes compared to 512. For reference, the 512 model has a window size of 2441, while the 128 model has 9765.

Training Performance (window size)

Historically, the minimum window size was established at 305; however, this measurement is anticipated to diminish further. It is evident that reducing the window size to such an extent, whereby the peaks fail to fit within a single window, poses significant challenges. For instance, a window size of 16 (Figure 24) exemplifies the manual labeling graph, necessitating the use of three windows to accurately represent one peak.

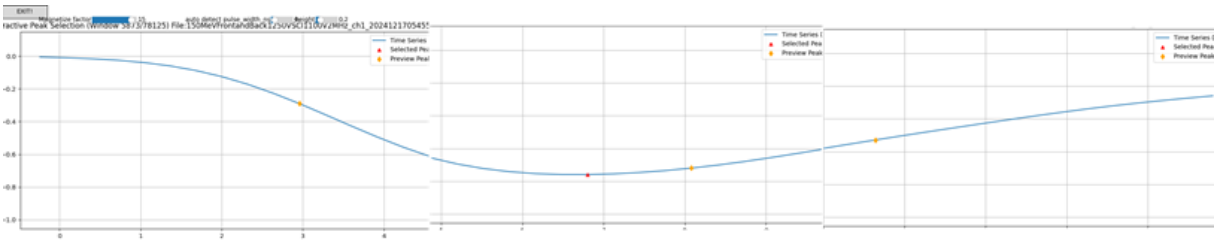


Figure 24: Three windows for one peak, size 16 [Labeling UI].

The methodology section indicated that a new pipeline version is also being utilized for the current tests. While testing this version with the default values, some discrepancies were discovered that raised concerns about potential bugs introduced with the changes. Therefore, although both results are provided, the old values offer a more accurate representation.

For the tests in Table XLII, the new pipeline initially ran with the suggested values from the provided code (window sizes from 16 to 2048, with maximum peaks of 1, 2, 2, 2, 2, 2, 3, and 4, respectively). Then, it operated with the maximum number of peaks that the model could train for comparison (1, 2, 2, 3, 3, 3, 4, 5). Afterward, the default pipeline was used with the same max_peaks. The rationale is that since the scope of this project is to use machine learning to help identify piled-up particles, disregarding examples where they occur would contradict that purpose. However, there are instances when one must resort to lower numbers because the model would refuse to train otherwise; for example, there were instances in the training data but none in the evaluation data. The higher the value, the fewer samples are obtained, which in turn decreases the model's effectiveness due to inadequate training with few examples. Thus, the conclusion reaffirms that using fewer classes, even when more data is available, artificially increases the model's accuracy.

The data preparation script for this size indicates that over a million examples for zero peaks were found, nearly eighty thousand for one peak, but only 53 examples for two peaks in a window. Using method one, training the model to detect only one peak resulted in it predicting over 1,200 peaks in a file containing about 500 of them, whereas training it to predict two peaks led to the model predicting none. Running the new evaluation script from method two shows that 100% of the cases are supposedly identified correctly for one peak detection. Training it with two peaks also disrupts this pipeline, resulting in 0% detection.

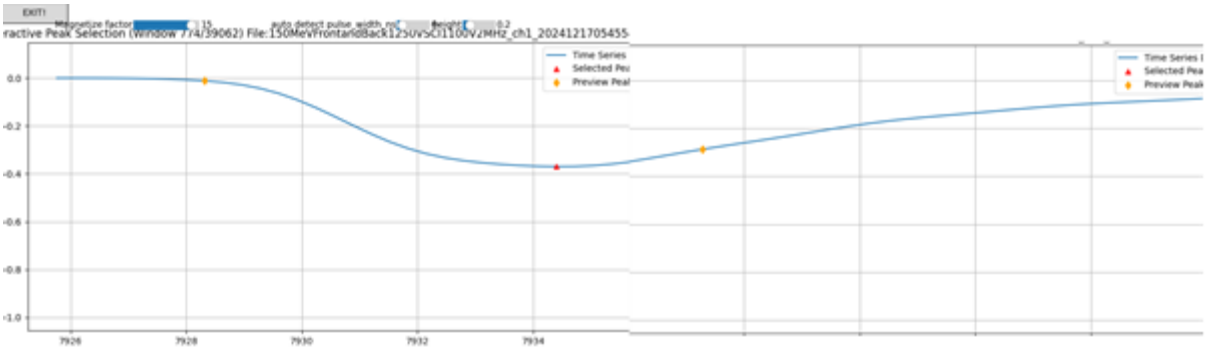


Figure 25: One peak in two windows, size 32 [Labeling UI].

With a window size of 32 (Figure 25), having peaks divided into two windows seems to have resulted in many of them being double-identified at the cut points. Using method one, over 700 peaks were identified, even though the first evaluated file had around 500. Since reviewing all files takes almost an hour, it's necessary to check the results of the first file to determine if they

are acceptable before allowing the full run. For method two, 91.55% of one-peak classes were identified correctly, while two-peak classes reached 100%, resulting in a total of 97.18% exact matches. However, considering that only 284 out of 20,092 one-peak examples were checked, these results appear random. In this method, both sizes 16 and 32 showed zero single-digit numbers of windows where peaks were either over- or under-detected.

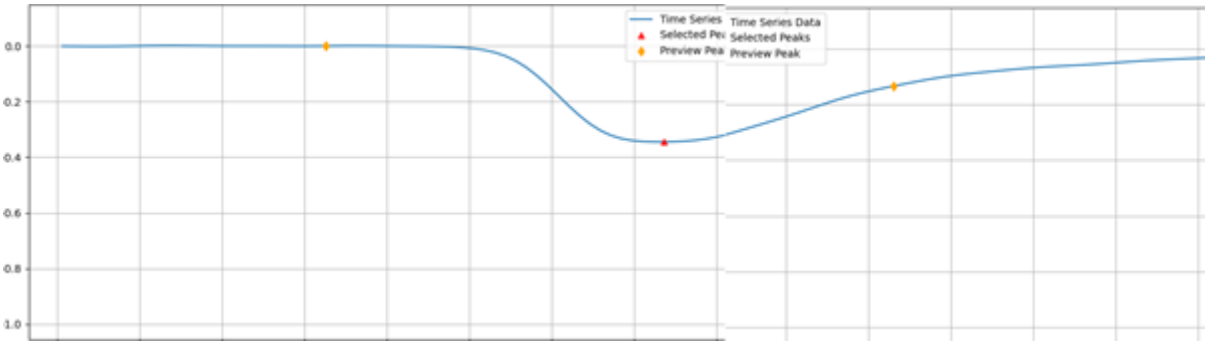


Figure 26: Peak in two windows, size 64 [Labeling UI].

With size 64, positive changes can already be observed, as method one evaluation allows for the detection of sensible values. As shown in Figure 26, waves can still be divided into two windows, but now the peak is pronounced enough that the second window consists solely of the wave's descent, which increases accuracy to some extent. In method two, the only significant value is that for windows with one peak, 71 out of 1616 were over-detected, meaning there were more predictions than actual labels.

Table XLII: Training results in method two with different max peaks

Window Size	Test	Train Acc	Train Loss	Train Top3	Val Acc	Val Loss	Val Top3
16	Test One	99.82%	0.0072	100.00%	99.13%	0.0238	100.00%
	Test Two	99.71%	0.0109	100.00%	99.42%	0.0259	100.00%
32	Test One	97.50%	0.0862	100.00%	96.83%	0.0993	100.00%
	Test Two	97.12%	0.0895	100.00%	98.33%	0.0782	100.00%
64	Test One	98.14%	0.0533	100.00%	98.23%	0.0525	100.00%
	Test Two	98.23%	0.0528	100.00%	98.31%	0.0554	100.00%
128	Test One	98.59%	0.0366	100.00%	99.06%	0.0348	100.00%
	Test Two	**91.73%**	**0.2132**	**100.00%**	**93.66%**	**0.2148**	**100.00%**
256	Test One	98.57%	0.0399	100.00%	99.06%	0.0344	100.00%
	Test Two	**92.43%**	**0.2315**	**100.00%**	**92.96%**	**0.2599**	**100.00%**
512	Test One	99.04%	0.0306	100.00%	99.04%	0.0332	100.00%
	Test Two	**88.57%**	**0.3035**	**100.00%**	**94.37%**	**0.2709**	**100.00%**
1024	Test One	97.28%	0.0722	100.00%	97.38%	0.0884	100.00%
	Test Two	**91.84%**	**0.2008**	**100.00%**	**88.74%**	**0.2826**	**100.00%**
2048	Test One	97.08%	0.0779	100.00%	94.73%	0.1776	99.88%
	Test Two	**90.77%**	**0.2348**	**99.99%**	**85.26%**	**0.3952**	**99.56%**

In the initial three tests (Table XLII), the proposed values were found to be the upper limits, ensuring that the results were comprehensive and that the model functioned adequately. By increasing the window size from 128 to 2048, it became feasible to augment the number of

classes by one. This modification subsequently led to a decline in accuracy, varying between 8 and 13%. This further substantiates the notion that models incorporating a greater number of classes alongside a limited number of examples tend to induce confusion in the model.

Because classes with more peaks have fewer examples, method two also becomes less reliable when using them. When moving from 128 to 2048 with max peaks of 2, 2, 2, 3, and 4, there were about 1550 examples for class 2 and about 800 examples for classes 3 and 4. However, the model with max peaks of 3, 3, 3, 4, and 5 had only about 80 examples for classes 3 and 4 and 190 for class 5.

According to the evaluation results of method two, Tables XLIII and XLIV can be extracted:

Table XLIII: Relevant errors found with method two

Window Size	Max Peaks	Total Examples	Over Detected	Under Detected
512	2	71	6	1
512	3	71	0	7
1024	3	87	6	8
2048	3	190	38	40
2048	4	190	42	17
2048	5	190	0	28

Table XLIII illustrates the instances of over-detection and under-detection (extra and fewer peaks) using an average rather than a case-by-case analysis. It is evident that even when compared to others, windows with more peaks are more likely to be incorrectly predicted by the model.

Table XLIV: Relevant errors in method two with different max peaks

Window Size	Class	Test	Over (%)	Under (%)
512	**2**	Test One	0%	0.45%
		Test Two	**8.45%**	**1.41%**
1024	**3**	Test One	0%	3.66%
		Test Two	**6.9%**	**9.2%**
2048	**3**	Test One	8.58%	5.32%
		Test Two	**20.0%**	**21.1%**
2048	**4**	Test One	0%	4.47%
		Test Two	**22.1%**	**8.9%**

Table XLIV uses the values from the first and compares their equivalents on the first test if they exist. Window sizes 512 and 2048, with classes 3 and 5, had 9.86% and 14.7% fewer detected peaks than those labeled by the model (both had 0% over-detection). The table also indicates that errors increase two to four times with an increase in total class number. While under-prediction appears prevalent in all tests, over-detection increases from zero to several due to this “small” change.

Returning to the analysis conducted on Method One, a window size of 128 is once again partitioned into multiple windows (Figure 27). Furthermore, the generation of training data experiences an acceleration as the quantity of windows reduces from 78,000 to 10,000. Larger window sizes predominantly fit within a single graph; however, there remain instances where they are divided into two.

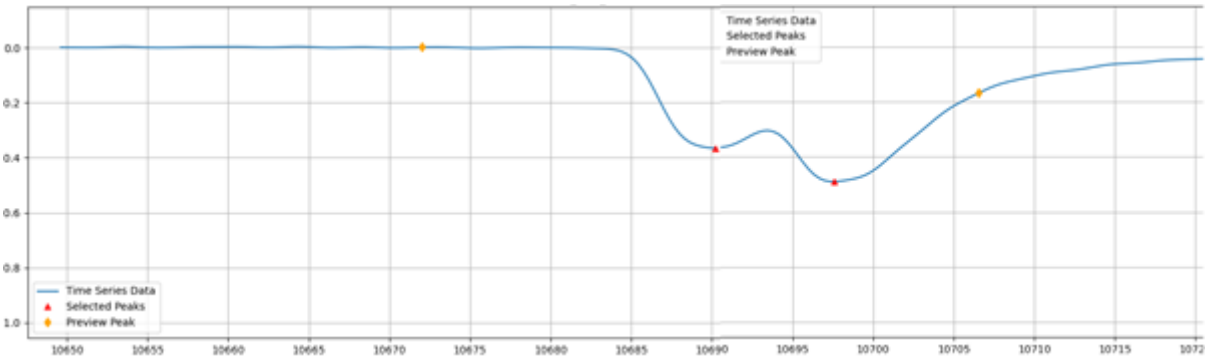


Figure 27: One peak into two windows, size 128 [Labeling UI].

The test utilizing the suggested variables (test one) for maximum peaks yielded Table XLV. Here, 512 and 1024 exhibited the worst performance, while 128 and 256 achieved the highest prediction rates, maintaining the overall total error rate from manual training and significantly increasing the number of exact matches. One might expect over-detection to be considerably higher, given that the models did not use examples with a greater peak count in their windows; however, the effect was the opposite.

Table XLV: Evaluation results for test one, with lower class amount

Variables	**Manual**	**64**	**128**	**256**	**512**	**1024**	**2048**
Max Peak	**13**	**2**	**2**	**2**	**2**	**3**	**4**
Labeled Peaks	99858	98975	98966	98955	98955	98913	98913
Predicted Peaks	97,79	81,58	89,16	89,45	**53,11**	**76,60**	81,06
Peak Difference	2,21	18,42	10,84	10,55	46,89	23,40	18,94
Total errors	20,03	39,15	**22,30**	**17,22**	50,12	32,96	25,87
Total of exact matches	24,61	**67,48**	**78,17**	**80,07**	**47,15**	**59,01**	**58,79**
Over-detection	8,91	10,36	5,73	3,34	1,62	4,78	3,47
Under-detection	11,12	28,78	16,57	13,89	48,50	28,18	22,40

Test two, using the maximum number of peaks allowed by the model for training, yielded the following results (Table XLVI). Here, 512 and 1024 continue to be the worst performers, but all of them experience a significant drop in predictions. Conversely, except for 512, all window sizes had a higher number of exact matches compared to the model trained with manual labels. The 128 window size appears to have performed the best across both tests. However, 256 performed worse with an increased number of peaks for training.

Table XLVI: Evaluation results for test one, with a higher class amount

Variables	**Manual**	**64**	**128**	**256**	**512**	**1024**	**2048**
Max Peak	**13**	**2**	**3**	**3**	**3**	**4**	**5**
Labeled Peaks	99858	98975	98966	98955	98955	98913	98913
Predicted Peaks	97,79	81,58	81,35	**55,98**	**29,88**	**53,05**	**60,63**
Peak Difference	2,21	18,42	18,65	44,02	70,12	46,95	39,37
Total errors	20,03	39,15	54,01	57,63	81,37	60,27	51,16
Total of exact matches	24,61	**67,48**	**48,58**	**43,71**	19,19	**34,20**	**36,78**
Over-detection	8,91	10,36	17,68	6,80	5,63	6,66	5,90
Under-detection	11,12	28,78	36,33	50,83	75,74	53,61	45,27

Tables XLVII and XLVIII compare the accuracy predictions from both methods.

Table XLVII: Evaluation results comparing predicted peaks in both methods, less classes

Variables	**Manual**	**64**	**128**	**128**	**256**	**256**
Max Peak	**13**	**2**	**2**	**3**	**2**	**3**
Labeled Peaks	99858	98975	98966	98966	98955	98955
Method one %	97,79	81,58	**89,16**	81,35	89,45	**55,98**
Method two %	-	98,14	**98,59**	91,73	98,57	92,43

In table XLVII, it is evident that, generally, reviewing the entire dataset reveals more errors than analyzing a small sample size. This is apparent, beginning at 256 and continuing through table XLVIII.

Table XLVIII: Evaluation results comparing predicted peaks in both methods, more classes

Variables	**512**	**512**	**1024**	**1024**	**2048**	**2048**
Max Peak	**2**	**3**	**3**	**4**	**4**	**5**
Labeled Peaks	98955	98955	98913	98913	98913	98913
Method one %	53,11	**29,88**	76,6	53,05	81,06	60,63
Method two %	99,04	88,57	97,28	91,84	97,08	90,77

While the sample size indicates that the model is performing exceptionally well, the actuality diverges: most instances yield results at least 10% inferior, while others achieve only one-third of the anticipated performance. Furthermore, a window size of 64 yielded the most favorable outcomes in these evaluations. Nevertheless, a window size of 128 may be deemed more appropriate for the top position in this context, primarily because obtaining a comparable result with half the number of windows per file significantly reduces both data generation and evaluation runtime.

4. Discussion

Following comprehensive research on labeling applications, manual data annotation, hyperparameter tuning, and training iterations, it has become evident that although parameter adjustments can enhance model accuracy and mitigate overfitting, the impact remains limited by data constraints.

Manual labeling has been demonstrated to be the most effective method for enhancing model performance, particularly when utilized in conjunction with optimized data generation and training parameters. Nevertheless, it remains impractical to attain 100% training and validation accuracy with a CNN due to the significant labor involved; specifically, the manual annotation of slightly more than half a second of data requires approximately two full workdays. Furthermore, augmenting the dataset by 50% did not result in enhancements beyond the model's intrinsic error rate.

An effective strategy involves refining the detection algorithm, which currently identifies a near-optimal number of peaks. With properly calibrated hyperparameters and logic, automated detection could enable the creation of a sufficiently large, labeled dataset for CNN training. At present, 100,000 labeled peaks have proven insufficient.

Furthermore, higher frequencies (5 MHz and 10 MHz) pose additional challenges, as their peak reflections are less consistent than those in 2 MHz datasets, which makes algorithmic detection less reliable. As a result, these frequencies would require manual labeling to facilitate effective CNN training.

4.1 Comparison with Previous Findings

Upon the completion of testing, it is imperative to draw comparisons with the previous work [1]. Beginning with the parameters sourced directly from his research (Figure 28), modifications were implemented, specifically the omission of pruning, to ensure that no peaks were overlooked during the manual labeling process. In addition, the window size was predetermined in a different context, rendering the corresponding value in this instance unnecessary.

```
default_parameters = {
                    "window-size": 200,
                    "prune": True,
                    "prune-factor": 10,
                    "filters-count": 8,
                    "kernel-size": 2,
                    "pool-size": 2,
                    "dropout-rate": 0.5
                }
```

```
parameters = {
    "window-size": 200,
    "filters-count": 8,
    "kernel-size": 2,
    "pool-size": 2,
    "dropout-rate": 0.5
}
```

Figure 28: Parameter Comparison (Left: Code from [1]; Right: Provided code [PyCharm IDE]).

The document [1] asserts that a window size of 15 results in the highest performance. Concurrently, the other parameters do not affect the accuracy of the model but solely its processing speed. Presented below (Table XLIX) are the results he cites regarding different window sizes:

Table XLIX: Results with different window sizes [1]

Window size	800	400	200	100	50	25	15
Accuracy (%)	92.32	95.97	97.93	98.95	99.47	99.74	99.84

Upon comparison of the results for windows of comparable dimensions (Tables L and LI), it becomes apparent that they exhibit a remarkable similarity, with a margin of error of approximately 2% (as referenced in a preceding section).

Table L: Equivalent results from this project to compare with [1]

Window Size	2048	1024	512	256	128	100	64	32	16
Train Acc %	97.08	97.28	99.04	98.57	98.59	98.81	98.23	97.50	99.82

The larger window sizes clearly demonstrate a significant variation. Given that reference [1] appears not to have employed a classification analogous to that utilized in this project for maximum peaks, it is viable to compare those results with the testing conducted using the threshold of the higher values that the models would be trained on.

Table LI: Second part of the previous comparison

Window Size	2048	1024	512	256	128
Train Acc %	90.77	91.84	88.57	92.43	91.73

Upon comparing the values from tables XLIX to LI, it was observed that the increase in window size correlates with a trend of diminishing returns in the gradual decrease of accuracy during the training process. The sole outliers are sizes 256 and 512, which diverge significantly from the anticipated results.

The author [1] does not specify the precise values utilized for width and amplitude in the context of peak detection. Nevertheless, as he expresses a preference for window size 15—typically encompassing a singular wave—it may be inferred that a width of 15 ns and amplitude of 0.2, as employed by the GSI detection algorithm, constituted the standard parameters. In light of the fact that width 6 ns and amplitude 0.1 were implemented, this should be duly considered when evaluating the results.

4.2 Convolutional Neural Networks vs. Peak Detection Algorithms

A question that needs to be addressed is whether CNNs were more effective than merely relying on the peak detection algorithm. Table LII aims to clarify this:

Table LII: Labeled vs. predicted peaks with different widths and amplitudes

Variables	Manual	w=15ns, h=0.2	w=15ns, h=0.1	w=6ns, h=0.1	w=6ns, h=0.2
Labeled Peaks	99858	96219	96640	98976	98549
Predicted Peaks	97656	86669	91841	90693	93049

From the first set of tests, “labeled data” indicates the algorithm’s performance, while “predicted peaks" reflect the CNN's effectiveness. The conclusion is that the CNN performed best with manually labeled data, while the optimal algorithm parameters were a width of 6ns and an amplitude of 0.1.

Table LIII: Labeled vs. predicted peaks with different window amounts/sizes

Window size	12500	19531	9765	4882	2441	1220	610	305
Labeled Peaks	98976	98975	98966	98955	98955	98913	98963	98963
Predicted Peaks	90693	81544	**98308**	95166	93182	76472	44204	64220

The subsequent tests (Table LIII) employ these values for automatic labeling while only altering the window size. From this point on, all labeled peaks demonstrate minimal significant differences, while the window size clearly indicates how well each model performs with the changes.

Table LIV: Labeled vs. predicted peaks with different window sizes (power of 2)

Window size	64	128	128	256		512		1024		2048	
max peaks	2	2	3	2	3	2	3	3	4	4	5
Labeled Peaks	98975	98966	98966	98955	98955	98955	98955	98913	98913	98913	98913
Predicted Peaks	80744	88239	80509	88519	55393	52557	29572	75770	52471	80181	59969

In the recent tests (Table LIV), performance was not necessarily optimal with smaller window sizes. Among all the tests, a window size of 9765 with automatic labeling at a width of 6ns and amplitude of 0.1 produced the best results, even surpassing the model with manually labeled data. However, results varied significantly across training sessions, and even the model with manual labels exhibited a peak detection performance as low as 94%. Therefore, there is a range of window sizes and parameters that yield similar performance results when taking error rates into account.

4.2.1 Potential Algorithm Modifications

In addition to other lines of code that assist in organizing the peak findings, the script for automatic labeling employs the following methods to identify peaks (Figure 29):

```
# Detect initial peaks
peaks, _ = find_peaks(-1 * data.y, height=height, threshold=threshold, distance=distance_param)
```

Figure 29: find_peaks() [PyCharm IDE].

This is the find_peaks() function from the scipy.signal Python library. When reviewing the documentation, it states that the function allows for prominence detection, or "how wide" a wave is. Another option is using the width filter to determine if it helps improve peak detection. Subsequent research may investigate these values to ascertain whether they facilitate the identification of pile-ups through coding, particularly given that they are also challenging to detect visually.

Waves could also undergo further processing rather than being accepted as they are. Other functional options available in the library's documentation include smoothing the waves with

the savgol_filter() and using np.gradient() to ideally sharpen the edges where the peaks occur. Making the amplitude dynamic instead of fixed could also help eliminate the frequent false positives found in reflections after high amplitude waves; for 2 MHz files, the default amplitude could be set to 0.1, which would increase to 0.2 after a wave of 1.2 amplitude.

If those changes do not lead to any improvement, other libraries could be explored besides SciPy. PeakUtils, Photutils, Lmfit, and PyWavelets might be relevant alternatives to consider, with their functions being indexes(), deblend_sources(), fit(), and cwt().

4.3 Challenges and Limitations

This is a summary of all the problems encountered during the project. These challenges can be classified into four distinct categories: data labeling challenges, limitations in model training, issues related to high-frequency and multi-channel data, and complications associated with pipelines and scripts.

Labeling was deemed impracticable with third-party tools owing to their incompatibility with WFM files, resulting in a delay of nearly two months in the commencement of the project. Additionally, the user interface introduced inefficiencies, as each modification necessitated individual saving rather than batch processing. Furthermore, the malfunctioning default zoom function further impeded progress until rectified. Consequently, labeling 16,800 windows necessitated three complete passes.

Training CNNs highlighted limitations in data sufficiency and model robustness. Notwithstanding the utilization of approximately 100,000 labeled peaks, certain models demonstrated overfitting, as evidenced by validation accuracy plateauing at approximately 40%, in contrast to a training accuracy of 98%. This phenomenon suggests a propensity for memorization rather than generalization.

Higher frequencies, such as 5 MHz and 10 MHz, resulted in shorter peak intervals and diminished signal strength, consequently leading to false positives, as peak reflections frequently exceeded the amplitude detection threshold. Specific challenges related to each channel included low-amplitude peaks (approximately 0.002) being indistinguishable from actual peaks, along with continual formatting errors present in Channel 4. Additionally, the sparse training slices at 10 MHz have often resulted in failures during dataset generation, as the model cannot train in classes that have zero examples.

Regarding the programming aspect, the implementation of the pipeline utilizing classifiers resulted in a technical bottleneck that adversely affected script performance during window size testing. The generation of millions of windows extended the duration of each procedural step, with each test involving varying parameters requiring several hours for completion in terms of training and evaluation. A significant portion of this time was inefficiently spent on hypothesizing the appropriate number of classes for each dataset. Furthermore, the lack of training in Python and practical machine learning posed significant challenges, as progress would be impeded until the intricacies of the language were overcome.

Bibliography

[1] S. Engel, “Application of machine learning towards particle counting and identification,” 2022. [Unpublished bachelor's dissertation]. University of Essex.

[2] P. Forck, T. Hoffmann, and A. Peters, “Detectors for slowly extracted heavy ions at the gsi facility,” in Proc. DIPAC’97, ser. European Workshop on Diagnostics and Instrumentation for Particle Accelerators. Frascati, Italy: INFN-LNF, 1997, pp. 165–167. [Online]. Available: http://www.lnf.infn.it/sis/preprint/getfilepdf.php?filename=LNF-97-048(IR).pdf

[3] P. J. Niedermayer, R. Singh, and R. Geißler, “Software-defined radio based feedback system for beam spill control in particle accelerators,” in Proc. GRCon23, ser. GNU Radio Conference. Tempe, AZ, USA: GNU Radio Journal, Sep. 2023. [Online]. Available: https://pubs.gnuradio.org/index.php/grcon/article/view/133

[4] P. Niedermayer and R. Singh, “Excitation signal optimization for minimizing fluctuations in knock out slow extraction,” Sci. Rep., vol. 14, no. 1, p. 10310, May 2024. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC11070430/

[5] R. R. Wilson, “Radiological use of fast protons.” Radiology 47(5), 487–491, 1946. [Online]. Available: https://doi.org/10.1148/47.5.487.

[6] U. Amaldi and G. Kraf, “Radiotherapy with beams of carbon ions.” Rep. Prog. Phys. 68(8), 2005, pp. 1861–1882. [Online]. Available: https://doi.org/10.1088/0034-4885/68/8/R04.

[7] T. Kamada et al. “Carbon ion radiotherapy in Japan: An assessment of 20 years of clinical experience”. Lancet Oncol, 2015. [Online]. Available: https://doi.org/10.1016/S1470-2045(14)70412-7.

[8] Malouf, T. D. et al. “Carbon ion therapy: A modern review of an emerging technology.” Front. Oncol, 2020. [Online]. Available: https://doi.org/10.3389/fonc.2020.00082.

[9] I. Goodfellow, Y. Bengio, and A. Courville, “Deep Learning.” MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[10] S. Engel, R. Singh, and P. Boutachkov, “Application of machine learning towards particle counting and identification,” in International Beam Instrumentation Conference (IBIC). JACOW Publishing, Geneva, Switzerland, 2022, pp. 508–511. [Online]. Available: https://accelconf.web.cern.ch/ibic2022/doi/JACoW-IBIC2022-WEP42.html