***Abstract—*This follow-up study presents a comprehensive approach to refining echo detection precision in ultrasonic sensing. Leveraging Convolutional Neural Networks (CNN), Random Forests, and XGBoost, we introduce a multi-faceted machine learning framework aimed at enhancing the detection of the first echo signal. Employing the Red Pitaya board and Ultrasonic Sensor SRF02, we focus on dataset #3 to validate our methodologies. This paper details the software implementations and evaluates the performance of each model, marking a significant advancement in echo detection technologies.**

Reliability test and improvement of a sensor system for object detection

Course Information Technology

Modules Autonomous Intelligent Systems and Machine Learning

By Dr. Peter Nauth and Dr. Andreas Pech

|  |  |
| --- | --- |
| Richa Padhi  Mat No: 1438260 | Theertha Bharathan  Mat No: 1445457 |
| richa.padhi@stud.fra-uas.de | theertha.bharathan@stud.fra-uas.de |

Reliability test and improvement of a sensor system for object detection

Course Information Technology

Modules Autonomous Intelligent Systems and Machine Learning

By Dr. Peter Nauth and Dr. Andreas Pech

|  |  |
| --- | --- |
| Richa Padhi  Mat No: 1438260 | Theertha Bharathan  Mat No: 1445457 |
| richa.padhi@stud.fra-uas.de | theertha.bharathan@stud.fra-uas.de |

***Keywords—Echo Detection, Ultrasonic Sensing, Machine Learning, Convolutional Neural Networks, Random Forest, XGBoost, Precision Enhancement***

# INTRODUCTION

Building upon the bedrock of our initial investigations, this research extends into the realm of sophisticated machine learning methodologies to elevate the acuity of first echo detection within ultrasonic sensor systems. Our holistic strategy encompasses an ensemble of machine learning models, each selected for their distinct strengths and collective synergy. We employ Convolutional Neural Networks (CNN) to leverage their exceptional signal processing capabilities, pivotal for discerning patterns within intricate data structures. Random Forests are incorporated for their esteemed robustness in classification tasks, offering reliability and resilience in diverse conditions. Lastly, XGBoost is integrated for its streamlined efficiency and formidable prowess in predictive modeling, providing a swift yet powerful analytical engine.

Our aim is to surpass existing standards in echo detection fidelity, a critical component for an array of applications ranging from autonomous navigation to intricate safety mechanisms. Key to our approach is the enhancement of first echo precision by meticulously searching for the maximum peak within designated time windows—windows that are intelligently delineated by our machine learning models. This advanced technique ensures that our system not only identifies the presence of an echo but also pinpoints its most prominent feature with unparalleled accuracy.

To validate our methodology and reinforce our findings, we plan to revisit and refine our experiment, focusing on dataset #3. This dataset will serve as a new proving ground for our enhanced detection algorithm, where we anticipate observing substantial improvements in performance metrics. Through this iterative process of experimentation and refinement, our research endeavours to make significant contributions to the field, pushing the boundaries of what is possible with ultrasonic sensor technology and echo detection.

# METHODOLOGY

Our methodology builds on the dual-layered approach of machine learning-driven time window definition followed by targeted peak analysis. This section expands on the software implementations for CNN, Random Forest, and XGBoost models, providing insights into their training, evaluation, and application in echo detection.

## Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs), also known as ConvNets, are a category of deep neural networks that are particularly effective for image recognition and classification tasks. At their core, CNNs consist of neurons with learnable weights and biases, which undergo a series of transformations to produce a final output that can be used for classification or regression tasks. Unlike standard fully connected networks that treat input data as a flat vector, CNNs preserve the spatial structure of the data, treating inputs as 2D images. This approach reduces the number of parameters, making the network less prone to overfitting and more efficient in learning spatial hierarchies in data. A CNN's structure typically includes several hidden layers, including convolutional, pooling, and fully connected layers. In the convolutional layers, filters or kernels process the input image to detect features such as edges, lines, and textures [1].

The hidden layers include a series of convolutional layers that convolve with a multiplication or dot product of the input data with a set of learnable filters, which helps in feature extraction. This is followed by non-linear activation functions like ReLU (Rectified Linear Unit) to introduce non-linear properties into the network. Pooling or subsampling layers follow, which reduce the dimensionality of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer, hence reducing the number of parameters and computations in the network, and helping in making the detection of features invariant to scale and orientation changes [1].

Finally, fully connected layers, similar to the traditional multi-layer perceptron, are used where each neuron is connected to every neuron in the previous layer to classify the images into various classes based on the high-level features extracted by the convolutional and pooling layers [2].

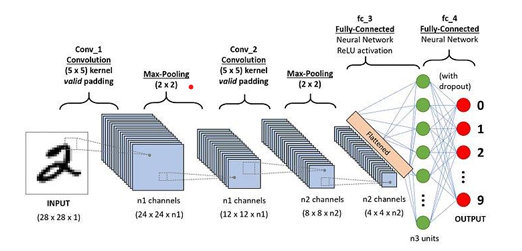
CNNs use a variation of multilayer perceptron’s designed to require minimal preprocessing. They are also known for their ability to scale to image data and their efficiency in image recognition tasks due to weight sharing and sparse interactions, which significantly reduce the number of parameters in the network. CNNs are powerful neural networks that leverage the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, they are widely used in image recognition and classification tasks, achieving remarkable performance [1]. 

Fig.1. Convolutional Neural Network Architecture

Our CNN model employs a sequence of convolutional, dropout, and pooling layers, culminating in dense layers for classification. The model is trained on the pre-processed dataset, focusing on windowed signal segments to identify the presence of the first echo [1].

## Random Forest

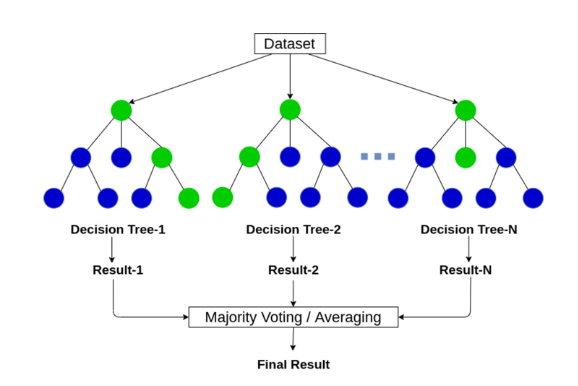
Random Forest is a sophisticated machine learning algorithm known for its versatility and ease of use in both classification and regression tasks. It operates by constructing an ensemble of decision trees, typically through a process called bagging, to improve the robustness and accuracy of predictions. In essence, it generates multiple decision trees on randomly selected data subsets, then integrates their predictions. This approach not only enhances prediction accuracy but also mitigates the risk of overfitting, a common issue in single decision tree models [3].

Fig.2. Random Forest Architecture

The algorithm introduces randomness in two key ways: first, by selecting a random subset of the training data for building each tree, and second, by choosing a random subset of features for splitting each node within the trees. This randomness ensures that the ensemble of trees is diverse, leading to a more generalizable model that performs well on unseen data [3].

A distinct advantage of Random Forest is its ability to handle a large number of features and identify the most significant ones, making it highly applicable in fields where feature importance is crucial. Despite its many benefits, it's important to consider that increasing the number of trees enhances the model's accuracy up to a point but also adds to computational complexity, which might not be ideal for time-sensitive predictions [3].

Random Forest is a powerful and reliable tool in the machine learning toolkit, providing high accuracy, handling overfitting effectively, and offering insights into feature significance, albeit with considerations for computational efficiency. This ensemble method contributes to the robustness of our echo detection system, particularly in classifying signal segments based on the presence of echo-related features [3].

## XGBoost

XGBoost, standing for eXtreme Gradient Boosting, is a highly efficient and advanced implementation of gradient boosting, a machine learning algorithm that builds upon the concept of boosting weak learners (decision trees) to form a strong predictive model. Developed by Tianqi Chen in 2014, XGBoost has gained prominence for its performance and speed in machine learning competitions and practical applications [4].

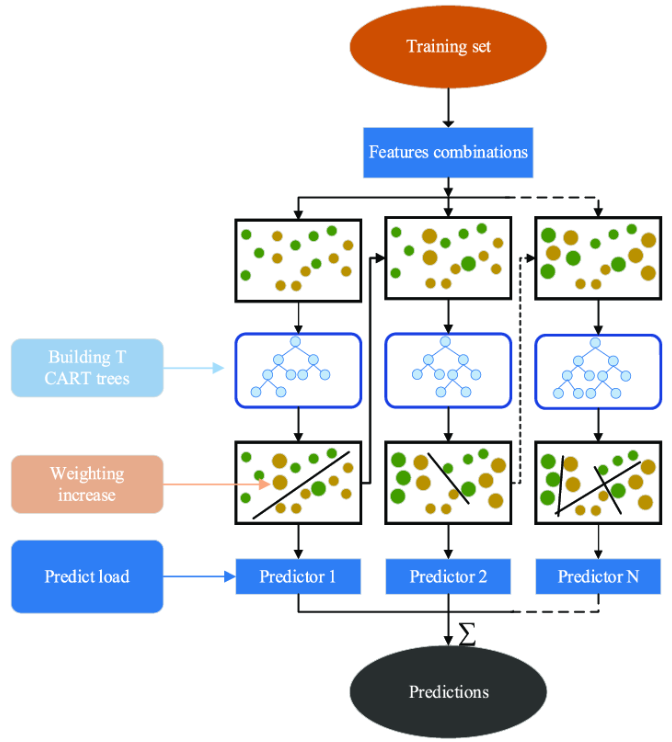
The core of XGBoost lies in its ability to perform parallel tree boosting, which significantly speeds up the learning process. Unlike traditional gradient boosting methods, XGBoost incorporates techniques such as regularization and auto-pruning to enhance model performance and prevent overfitting. Regularization helps control the model's complexity, reducing the risk of overfitting, while auto-pruning stops the growth of trees when they no longer contribute to the overall prediction accuracy, ensuring a compact and efficient model [4].

Fig.3. XGBoost Architecture

XGBoost is designed with flexibility and portability in mind, supporting integration with various programming languages and platforms, including R, Python, and Julia. Its capability to handle missing values and support for sparse data structures make it adaptable to a wide range of datasets and scenarios [4].

The algorithm's popularity stems not only from its predictive power but also from its computational efficiency, attributed to parallel processing and cache optimization. By utilizing all available CPU cores or distributed systems, XGBoost ensures maximum computational utilization. Additionally, its cache optimization technique speeds up data retrieval, further enhancing the algorithm's speed. Incorporating XGBoost into a machine learning pipeline can significantly improve the accuracy and efficiency of predictive models, making it a favoured choice among data scientists and machine learning practitioners for a variety of data-driven tasks. The model is trained on the dataset to predict the occurrence of the first echo with high accuracy, benefiting from its ability to handle diverse data types and distributions .

# IMPLEMENTATION

*Software Implementation:*

The implementation is displayed with the flowchart provided This represents a machine learning pipeline for implementing a system that involves reading data, processing it, analysing peaks, grouping and labelling it, defining a model, and then training and saving that model.

A diagram of a process

Description automatically generated

Fig.4. Flowchart of the Model

*Common Pre-Processing steps:*

*Data Preprocessing:* The read\_and\_prepare\_data function reads signal data from a CSV file, focusing on columns with relevant data. This step prepares the raw ultrasonic signals for further analysis.

*Signal Windowing:* apply\_window mitigates spectral leakage by applying a Hann window to each signal. This process shapes the data to enhance the accuracy of the frequency analysis, essential for reliable echo detection.

*Noise Reduction and Peak Detection:* Within reduce\_noise\_and\_label, each signal is first transformed into the frequency domain using the Fast Fourier Transform (FFT). A Power Spectral Density (PSD) threshold filters out noise. An inverse FFT reconstructs the signal with reduced noise. The Hilbert Transform is then used on the filtered signal to find its envelope, aiding in peak detection with find\_peaks.

*Labelling the data:* This function is designed to process a list of detected peak positions from signal data and organize these findings into a structured format that indicates the presence of peaks within specified windows of time across multiple signals. It operates by first determining the number of possible windows in a given signal, which is calculated by dividing the total length of the signal by the width of each window. The function then creates a matrix, y\_label, to hold binary labels for each signal across all the windows. This matrix has as many rows as there are signals (determined by the length of peaks\_list) and as many columns as there are windows (n\_windows).

For each signal, the function examines the peak position. If a peak is detected (indicated by a peak position greater than or equal to zero), the function calculates which time window this peak falls into. This is done by dividing the peak position by the window width.

Once the appropriate window for a peak is identified, the function sets the corresponding element in the label matrix to 1. This indicates that within this particular window of the signal, a peak is present. If a window contains no peaks, it remains labelled as 0, indicating the absence of a peak.

Ultimately, the function returns this label matrix, providing a simple, binary, structured representation of where peaks occur within the set of signals, as delineated by the predefined time windows. This labelled data can be particularly useful for further machine learning tasks, such as training a model to recognize patterns associated with these peaks.

*Training the Model:*

## Convolutional Neural Networks (CNN)

This function trains a CNN model, which is particularly suitable for data with spatial structure, such as images or time series data. Here's a step-by-step breakdown:

Initialization: Set the training parameters, including the verbosity level, number of epochs, and batch size.

Model Definition:

* Sequential() creates a linear stack of layers.
* Conv1D layers are added for convolution operations on 1D data, with 64 filters and a kernel size of 3.
* Dropout is applied with a 0.5 rate to prevent overfitting.
* MaxPooling1D reduces the dimensionality, which helps the model to generalize better.
* Flatten converts the 2D feature maps to a 1D vector for the following dense layers.
* Dense layers are fully connected layers, with the last one using softmax activation to output probabilities for the different classes.

Model Compilation: The model is compiled using the categorical crossentropy loss function and the Adam optimizer.

Training: The model is trained on the given training data (xtrain, ytrain) for a set number of epochs and batch size.

Evaluation: The model's accuracy is evaluated on the test set.

Model Saving: The trained model is saved to the file system for later use.

Return: The trained model is returned from the function.

## Random Forest

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training time.

Parameter Grid Setup: A grid of hyperparameters is created to search for the best combination. It includes the number of trees in the forest (n\_estimators), the max number of features considered for splitting a node (`max\_features), the maximum depth of each tree (max\_depth), the minimum number of samples required to split a node (min\_samples\_split), and the minimum number of samples required at a leaf node (min\_samples\_leaf), and whether bootstrap samples are used when building trees (bootstrap`).

Randomized Search: RandomizedSearchCV is used for hyperparameter tuning. It explores a given parameter space randomly rather than exhaustively, which can be more efficient. The best parameters are printed out.

Model Training with Best Parameters: After finding the best parameters, a new Random Forest model is instantiated with these parameters and trained on the full training set.

Model Saving: The best model is saved to the file system as a pickle file.

Prediction and Evaluation: The model predicts the classes for the training and test datasets, and the F1 scores for both predictions are calculated and printed.

Return: The trained Random Forest model with the best parameters is returned.

## XGBoost

XGBoost is a powerful gradient boosting machine learning library that is often used for structured or tabular data.

Model Initialization: An XGBoost classifier is initialized with specified hyperparameters, such as the objective function (binary:logistic for binary classification), number of estimators, maximum depth of trees, learning rate, and subsample ratio.

Model Training: The XGBoost model is fitted on the training data.

Model Saving: The trained model is saved to the file system as a pickle file.

Evaluation: Predictions are made on the training and test data, and the F1 score for each is calculated and printed. The F1 score is a harmonic mean of precision and recall and is a better measure than accuracy for imbalanced datasets.

Return: The trained XGBoost model is returned.

# RESULT AND ANALYSIS:

Based on training the above 3 models for ADC data for hard objects at 1m. We could see that following results:

## Convolutional Neural Networks (CNN)

The CNN model shows high precision, recall, and F1-score for the majority classes (128, 129, 130), indicating that it is predicting these classes very well. The overall accuracy is 0.905 (90.5% correct predictions). The weighted average of F1-score and recall are both also high at around 0.91, suggesting good performance across all classes while considering the number of instances (support) in each class.

A screenshot of a computer

Description automatically generated

Fig.5. Classification report of CNN Model

A screenshot of a computer

Description automatically generatedFig.6. Classification report of Random Forest Model

## Random Forest

The Random Forest model, after hyperparameter tuning, achieved the best performance with 100 trees (n\_estimators), a minimum of 5 samples required to split an internal node (min\_samples\_split), only 1 sample required at a leaf node (min\_samples\_leaf), max\_features set to use the square root of the number of features, a maximum depth of the trees (max\_depth) at 30, and without bootstrapping (bootstrap set to False).This model also performs well on the major classes, with slightly lower performance on class 130 compared to the CNN model. The overall accuracy of this model is higher at 0.925 (92.5% correct predictions). The weighted F1-score and recall are also high and consistent with the overall accuracy at around 0.92.

## XGBoost

The XGBoost model has precision, recall, and F1-scores similar to the Random Forest model for the major classes, with a very slight decrease in recall for class 130. The accuracy of the XGBoost model is 0.865 (86.5% correct predictions), which is lower than the CNN and Random Forest models. The weighted average F1-score and recall are correspondingly lower at around 0.88. A screenshot of a computer

Description automatically generated

Fig.7. Classification report of XGBoost Model

# V. CONCLUSION & FUTURE SCOPE

In conclusion, the comparative analysis of CNN, Random Forest, and XGBoost models on the 1m dataset has provided valuable insights into the strengths and weaknesses of each approach. The CNN model emerged as the top performer, likely due to its ability to harness spatial dependencies within the data—a characteristic that is particularly relevant for signal processing tasks. The Random Forest model also demonstrated commendable accuracy, suggesting that ensemble methods are robust and effective for this class of problems. XGBoost, while slightly trailing, still showed a respectable level of accuracy and remains a competitive option, especially considering its speed and scalability.

Future Scope:

Moving forward, to enhance the robustness and applicability of the models, the following future directions could be considered:

Dataset Expansion: Introducing a broader range of scenarios, including signals reflected from hard objects at greater distances such as 50m, and incorporating data from soft objects with different postures (sitting and standing), would enrich the dataset. This diversity could help improve the model's generalization capabilities and accuracy.

Model Optimization: Further tuning of the model hyperparameters, perhaps through more sophisticated methods like Bayesian Optimization, could lead to better performance. Additionally, exploring more complex CNN architectures or advanced ensemble techniques could uncover improvements.

Feature Engineering: Developing more intricate features that capture the essence of signal reflections from various surfaces and distances might enhance the models' ability to distinguish between different classes more effectively.

Class Imbalance Mitigation: Addressing the class imbalance with techniques such as synthetic minority oversampling (SMOTE) or adaptive sampling could improve the model performance, particularly for underrepresented classes.

Real-time Processing: Adapting the models for real-time signal processing could be beneficial for applications that require immediate decisions, such as autonomous driving or active surveillance systems.

Model Ensemble: Combining the predictions from multiple models in an ensemble method could yield better performance than any single model, leveraging the strengths of each.

Interpretability and Explainability: Investing in interpretability frameworks to understand the model's decision-making process could build trust in applications where decisions need to be justified.

Transfer Learning: Applying transfer learning, particularly for CNNs, where a model trained on a large dataset could be fine-tuned on the specific dataset used in this study to potentially improve results.

Hardware Optimization: Considering the computational efficiency and optimizing the models to run on specific hardware can make them more suitable for deployment in real-world scenarios.

Extended Validation: Testing the models on a separate, externally sourced validation set could provide a more robust measure of their true predictive performance.

By following these avenues, the future work can not only refine the predictive performance of the models but also expand their practicality and reliability in operational environments. This continuous improvement cycle is essential for developing AI systems that are both high-performing and trustworthy.

# REFERENCES

|  |  |
| --- | --- |
| [1] | K. J. Piczak, “ENVIRONMENTAL SOUND CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS,” *IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP),* p. 6, 2015. |
| [2] | Z. C. L. R. W. John R. Hershey, “Deep clustering: Discriminative embeddings for segmentation and separation,” 2015. |
| [3] | L. Breiman, “"Random forests," in Machine Learning,” vol. 45, pp. 5-32 . |
| [4] | T. C. a. C. Guestri, “XGBoost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016. |