IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Optimizing Neural Network training with Gradient Boosting Techniques

Author:
[1]Adigicherla Venkatesh,
M.Tech. Scholar,
Department of Computer Science,
School of Engineering,
Malla Reddy University,
Hyderabad, Telangana, India

Under the esteemed guidance of:
[2]Dr. Meeravali Shaik,
Associate Professor, DEAN – SOE (Computer Science)
Department of Computer Science,
School of Engineering,
Malla Reddy University,
Hyderabad, Telangana, India

Abstract: This study introduces two novel training frameworks—Gradient Boosted Recurrent Neural Network (GB-RNN) and Gradient Boosted Deep Neural Network (GB-DNN)—that synergize the principles of ensemble learning with deep learning models. By employing a stage-wise refinement strategy inspired by boosting, these models incrementally construct layered architectures that address limitations often seen in conventional neural networks, such as excessive model complexity, unstable gradients, and susceptibility to overfitting. Each layer in the network is trained to improve upon the residual shortcomings of previous iterations, enabling more efficient learning. Experimental validation is conducted using the CIFAR-10 image dataset, and performance is assessed using comprehensive classification metrics, demonstrating the effectiveness of the proposed approach.

In this context, this project proposes two specialized neural architectures: Gradient Boosted Recurrent Neural Network (GB-RNN) and Gradient Boosted Deep Neural Network (GB-DNN). These models are built upon the philosophy of modular learning, where additional components are introduced only when they yield measurable improvements. The GB-RNN framework is designed for data with inherent sequential or spatial ordering, such as images or signals, while GB-DNN is tailored for structured datasets where deep fully connected layers suffice.

Both models leverage a layered refinement mechanism where each new block addresses the residual errors left by previous layers. Early trained components are frozen to preserve learned patterns, and fine-tuning is selectively applied to ensure stability and efficiency. This approach not only enhances the learning process but also reduces redundancy, accelerates convergence, and improves generalization across data types.

Key Words **–** Gradient Boosting, Deep Learning, GB-RNN, GB-DNN, Image Classification, CIFAR-10, Ensemble Learning, Pseudo-Residual Learning, Layer Freezing, Dense Layers, Overfitting Reduction, Sequential Modeling, Fine-Tuning, Residual Correction, Performance Evaluation.

1. INTRODUCTION

1.1. The Evolution of AI and the Rise of Boosted Neural Architectures

The rapid evolution of artificial intelligence has been strongly influenced by the advancement of deep learning frameworks, which have demonstrated exceptional capabilities in modeling intricate patterns within data. These models have enabled a wide range of applications, from image understanding to speech recognition and sequential decision-making. However, the effectiveness of deep neural networks often hinges on architectural depth and large training datasets—factors that can introduce challenges such as gradient instability, excessive training time, and overfitting. To address these limitations, researchers have explored hybrid strategies that combine the strengths of deep learning with ensemble methods. One such method, gradient

boosting, has shown notable success in traditional machine learning for its iterative learning mechanism that refines model performance by focusing on prior errors. Integrating this principle into neural network design opens a new direction—one where networks are not trained in a single pass, but rather in successive stages, each aimed at improving upon the shortcomings of the last.

1.2. Integrating Gradient Boosting with Deep Learning Models

Gradient Boosting (GB) is an ensemble technique that enhances model accuracy by sequentially correcting errors made by weaker models. It typically involves decision trees that learn from the residuals of previous iterations, forming a robust overall predictor. Inspired by this, similar ideas have been applied in deep learning where multiple neural components are trained progressively to improve performance. This has led to the creation of Gradient Boosted Neural Networks (GBNNs), which merge the strengths of gradient boosting and deep architecture.

1.3 Proposed GB-RNN and GB-DNN Frameworks

The proposed study introduces two hybrid neural models that integrate gradient boosting principles with deep learning structures:

GB-RNN: Gradient Boosted Recurrent Neural Network

This model employs a boosting-like mechanism in RNNs. It adds new layers incrementally, each trained on the residual errors from previous iterations. Earlier layers are frozen to preserve learned knowledge and reduce computational complexity. Selective fine-tuning of recurrent units further improves feature extraction. Key features of GB-RNN include:

1. **Iterative Layer Addition**: At each boosting iteration, a new dense layer is added to a copy of the existing network. The newly added layer is trained on the residual errors from the previous iteration, ensuring that the network learns progressively from its mistakes.

2. **Layer Freezing**: Previously trained layers are frozen to prevent overfitting and to simplify the training process. This strategy reduces computational overhead and ensures that the network focuses on learning from residuals.

3. **Fine-Tuning RNN Layers**: In addition to adding new dense layers, previously trained recurrent layers are fine-tuned during each boosting iteration. This step enhances the network's ability to extract relevant features from input data.

GB-DNN: Gradient Boosted Deep Neural Network

Designed for structured or tabular data, this architecture applies a similar strategy using dense layers. It adds new fully connected layers iteratively, emphasizing the importance of modular learning and gradual refinement. Its key features include:

1. **Dense Layer Iteration**: Similar to GB-RNN, GB-DNN adds one dense layer at each boosting iteration and trains it on residual errors.

2. **Feature Extraction**: The framework explicitly integrates feature extraction into the training process, ensuring that the network captures meaningful patterns in the data.

3. **Weight Freezing**: Weights of previously trained layers are frozen to prevent overfitting and to stabilize the training process.

Both frameworks employ layer freezing and residual-based training to ensure efficient learning and better generalization without unnecessary computational overhead.

1.4 Addressing Model Complexity and Training Challenges:

Although effective, gradient-boosted neural models introduce challenges such as overfitting during fine-tuning, high computational demands, and instability in gradient propagation. To overcome these, the proposed frameworks incorporate:

1. **Overfitting in Fine-Tuning**: Fine-tuning recurrent layers at each boosting iteration may lead to overfitting, especially when dealing with small datasets.

2. **High Computational Costs**: The iterative addition of layers and fine-tuning increase the computational requirements, making GB-RNN less suitable for resource-constrained environments.

3. **Gradient Vanishing in Deep Networks**: As the network depth increases, gradient vanishing can still pose challenges, even with batch normalization.

4. **Complexity in Hyperparameter Tuning**: The need to balance learning rates, regularization, and other hyperparameters for each boosting iteration adds complexity to the training process.

1.5 Our Strategy to Address the Gaps:

The proposed GB-RNN and GB-DNN frameworks incorporate several enhancements to overcome the above limitations:

1. **Residual-Based Learning**: New layers directly address the errors made by earlier iterations, accelerating convergence and improving learning focus.

2. **Robust Regularization**: Use of layer freezing, dropout, and gradient clipping improves model generalization and training stability.

3. **Modular Flexibility**: Both models are designed to be extensible and adaptable to new datasets or task domains with minimal reconfiguration.

4. **Reduced Complexity in Tuning**: Particularly in GB-DNN, simpler architecture and residual-focused training reduce the hyperparameter tuning burden.

The integration of gradient boosting with neural networks is gaining momentum, offering new avenues for improving training efficiency and model generalization. While current approaches have demonstrated success, they often face limitations in flexibility, scalability, or performance consistency. Our proposed GB-RNN and GB-DNN frameworks aim to bridge these gaps by delivering a modular, robust, and adaptable approach. Future directions may include integration with advanced architectures (e.g., transformers), automated hyperparameter optimization, and applications in emerging areas like language modeling and real-time analytics. By integrating these strategies, our framework serves as a bridge between boosting theory and practical deep learning, enabling scalable and interpretable models across multiple domains.

2. LITERATURE SURVEY

The proposed framework introduces two gradient boosting-based neural architectures: GB-RNN and GB-DNN, each tailored for different data modalities. These models are designed to incrementally improve learning performance through a layer-wise refinement strategy. By employing a sequential learning process where each new layer is trained to capture patterns missed by its predecessors, the system leverages both boosting and deep learning to enhance generalization and convergence.

2.1. Integration of Neural Networks with Ensemble Principles

ProgressiveNet:

Instead of static architecture design, this method incrementally builds neural subnetworks over time, guided by an objective that balances model complexity and error minimization. At each stage, a new component is added only if it significantly contributes to performance, ensuring both efficiency and accuracy.

Confidence-Routed Networks:

These architectures rely on dynamic inference paths. If an initial model is confident in its prediction, it halts processing; otherwise, it passes the input to more complex models in a hierarchical stack. This mechanism conserves computational resources and provides adaptive complexity depending on input difficulty.

DomainBoost Adaptor:

Aimed at transfer learning tasks, this method aligns source and target distributions by using a staged adaptation mechanism. Each iteration introduces a lightweight learner trained on residual errors from prior stages, gradually narrowing the domain gap in scenarios involving distribution shifts.

2.2. Iterative Weak Supervision in Deep Learning

IteraLearn:

Designed for environments with partially labelled data, this framework generates soft labels using an initial base model. Over successive stages, additional learners are added to refine the predictions based on inconsistencies between predicted and observed patterns.

FairBoostNet:

Addressing fairness and performance simultaneously, this method optimizes two objectives in parallel. New learners are added to correct not just classification errors but also disparities in subgroup treatment, making it suitable for ethically sensitive applications.

PairwiseBoosted Learner:

Rather than working on class labels directly, this method trains relationships between data instances learning to distinguish similarly from dissimilar pairs. The boosting mechanism emphasizes incorrectly ranked pairs in subsequent iterations, improving performance in verification and ranking tasks.

2.3. Boosting-Inspired Architectures in Deep Learning

StageNet-CNN:

Inspired by additive modeling, this approach modifies the softmax layer to accommodate multiple weak classifiers whose outputs are combined iteratively. Each classifier attempts to correct the decision boundaries of the previous ones, leading to a more refined multi-class classification system.

ExpertCascade Framework:

This model combines the concept of specialized subnetworks with selective routing. At each level, distinct expert modules handle inputs that were poorly classified in prior stages. Trainable selector routes inputs through the appropriate expert path, making it effective for complex, high-variance data.

2.4. Our Strategy to Address the Gaps

Residual-Based Learning:

New layers directly address the errors made by earlier iterations, accelerating convergence and improving learning focus.

Robust Regularization:

Use of layer freezing, dropout, and gradient clipping improves model generalization and training stability.

Modular Flexibility:

Both models are designed to be extensible and adaptable to new datasets or task domains with minimal reconfiguration.

Reduced Complexity in Tuning:

Particularly in GB-DNN, simpler architecture and residual-focused training reduce the hyperparameter tuning burden.

The integration of gradient boosting with neural networks is gaining momentum, offering new avenues for improving training efficiency and model generalization. While current approaches have demonstrated success, they often face limitations in flexibility, scalability, or performance consistency. Our proposed GB-RNN and GB-DNN frameworks aim to bridge these gaps by delivering a modular, robust, and adaptable approach. Future directions may include integration with advanced architectures (e.g., transformers), automated hyperparameter optimization, and applications in emerging areas like language modeling and real-time analytics. By integrating these strategies, our framework serves as a bridge between boosting theory and practical deep learning, enabling scalable and interpretable models across multiple domains.

3. SYSTEM ANALYSIS

This chapter discusses the existing system and its disadvantages and how we overcome the challenges in proposed system.

3.1. EXISTING SYSTEM

Traditional deep learning systems such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Deep Neural Networks (DNNs) have been widely used for image classification, speech recognition, and time-series analysis. These models rely heavily on architectural depth and large datasets to learn intricate patterns in data. They employ end-to-end learning strategies where the entire model is trained in one pass, attempting to capture all data representations in a single training phase. Ensemble techniques like Gradient Boosting (GB), commonly used with decision trees (e.g., XGBoost, LightGBM), have also demonstrated superior performance on structured data. However, in traditional systems, these two powerful techniques deep learning and gradient boosting typically function independently, limiting their ability to leverage each other's strengths.

Disadvantages of Existing System:

- Gradient Instability in Deep Networks: Deep architectures like CNNs or RNNs suffer from vanishing or exploding gradients, which hampers training performance and stability, especially in very deep networks.
- Overfitting on Small Datasets: When trained on limited data, deep models often overfit due to their high capacity and large number of parameters.
- Long Training Times: Deep neural networks require extensive computational resources and time for convergence, especially when trained on large-scale datasets.
- Lack of Modular Learning: Existing models learn in a single-pass mode. There is no provision for incremental refinement of learning based on prior errors or residuals.
- Hyperparameter Sensitivity: These systems are highly sensitive to learning rates, regularization techniques, and other tuning parameters, making optimization a challenging task.
- Limited Reusability of Learned Features: Once a model is trained, adjusting or refining parts of the network without retraining the entire model is not straightforward.

3.2. PROPOSED SYSTEM

The proposed system introduces two novel architectures:

- **Gradient Boosted Recurrent Neural Network (GB-RNN) and**
- **Gradient Boosted Deep Neural Network (GB-DNN)**

Key Features of the Proposed System:

- Iterative training using residual-based learning.
- Layer freezing to retain previously learned information.
- Fine-tuning of only critical components to reduce computation.
- Adaptability for both image-based (GB-RNN) and tabular (GB-DNN) data types.

Advantages of Proposed System

- **Modular and Incremental Learning:** The boosting-based training approach allows the model to learn incrementally, focusing on residual errors at each stage, leading to improved learning efficiency.
- **Reduced Overfitting:** With selective fine-tuning and freezing of early layers, the model avoids redundant updates and overfitting, especially in small or noisy datasets.
- **Better Generalization:** By isolating training to residual errors and preserving earlier learned features, the model generalizes well across unseen data.
- **Improved Convergence:** Since each boosting iteration targets only the remaining error, convergence is faster and more stable compared to training a monolithic deep model.

- **Adaptability to Data Types:** GB-RNN handles sequential and image data efficiently, while GB-DNN is optimized for tabular/structured data making the system versatile across domains.
- **Scalable Architecture:** New layers can be added dynamically only when necessary, allowing the model to scale based on performance needs without redesigning the entire architecture.
- **Performance Gains:** Experimental results on CIFAR-10 demonstrate higher accuracy, better class-wise precision/recall, and improved learning stability compared to conventional CNN and RNN baselines.
- **Ease of Maintenance:** Component-based training reduces retraining effort. Previously learned blocks do not require modification unless necessary.

4. SYSTEM DESIGN AND IMPLEMENTATION

4.1. System Requirements

Hardware Requirements:

- System : i5 Processor.
- Hard Disk : 500 GB.
- Monitor : 15'' LED
- Input Devices : Keyboard, Mouse
- Ram : 8 GB

Software Requirements:

- Operating system : Windows 10 / 11
- Coding Language : Python 3.7
- Python Modules : tensorflow, keras, nuppy, matplotlib, sklearn
- Web Framework : Tkinter
- Frontend : HTML, CSS, JavaScript

4.2. Design Strategy and Model Construction

Both models share a common training paradigm but are structured differently to suit their target domains: GB-RNN for sequential or image-encoded sequences, and GB-DNN for structured datasets:

1. **Gradient Boosted Learning Cycle**:
   - GB iteratively minimizes the residual errors from previous models.
   - The additive nature of GB allows for building complex models layer by layer.
2. **Layer Expansion**:
   - A new dense layer is added at each iteration.
   - The newly added layer is trained on residuals from the previous iteration.
3. **Knowledge Preservation**:
   - All previously trained layers are frozen to prevent overfitting and reduce computational complexity.
   - Freezing ensures that the network focuses on correcting residual errors without revisiting already learned parameters.
4. **Architectural Flexibility**:
   - GB-RNN incorporates fine-tuning of convolutional layers, making it suitable for tasks involving spatial data like images.
   - GB-DNN focuses on dense layers, making it ideal for tabular and structured data.

4.3 GB-RNN Architecture for Visual-Sequential Data

GB-RNN is built to process input data that carries implicit order, such as image rows, time frames, or character sequences. For image datasets like CIFAR-10, the input matrix is interpreted as a sequence, e.g., treating each row of an image as a time step. This enables recurrent layers such as GRUs to model cross-row dependencies.

Architectural Components:

- **Sequence Encoding Layer:** The input is reshaped into a pseudo-sequential format to simulate temporal dependencies across spatial axes.
- **Recurrent Learning Core:** A GRU layer captures intermediate representations across the sequence.
- **Incremental Dense Blocks:** With each boosting iteration, a new dense block is added and trained solely on the remaining prediction errors.
- **Freezing and Focusing:** Once a layer completes training, its parameters are frozen, and the next layer is introduced.

Training Characteristics:

- The boosting iterations are driven by performance deltas, where new layers are only introduced if a measurable gain is observed.
- Early stopping mechanisms are integrated into the training pipeline to prevent overfitting.

4.4 GB-DNN Architecture for Structured Data

Unlike GB-RNN, the GB-DNN model caters to structured input where sequential relationships are not essential. The architecture relies entirely on dense layers and omits recurrent components. Its structure is optimized to learn hierarchical abstractions of tabular or flattened image data.

Architectural Blueprint:

1. **Initial Projection Layer**

   Transforms raw input features into a compact latent space.

2. **Layer-Boosting Cascade**

   New dense layers are introduced at each iteration, trained specifically on prediction residuals.

3. **Non-Trainable Backbone**

   Older layers remain unaltered during each phase, reducing computational load and avoiding catastrophic forgetting.

Training Logic:

- Each boosting step adds a layer that attempts to correct mistakes made by the ensemble so far.
- The final prediction is an aggregation of all layer outputs, each representing an incremental step in learning complexity.

4. **Dataset Adaptation: CIFAR-10**

CIFAR-10 is a multi-class image dataset comprising small 32×32 RGB images categorized into 10 distinct classes. In our framework:

- For GB-RNN, each image is treated as a sequence of 32 steps, each with 96 features (flattening RGB channels across rows).
- For GB-DNN, images are flattened into 3072-dimensional vectors before being processed through the dense block cascade.

5. **Integrating RNN (GRU) and Gradient Boosting (GB-RNN)**

Sequential Features with RNN: One way to integrate RNNs and Gradient Boosting could be to first use CNNs (for feature extraction) followed by a GRU-based sequence model to capture temporal dependencies or sequential patterns in the extracted features.

- Use a CNN to extract spatial features from CIFAR-10 images.
- Feed those features into an RNN (GRU or LSTM) to model sequential relationships or fine-grained spatial dependencies across different patches of the image or across multiple frames if the data is sequential.

Use Gradient Boosting on CNN Features: Another option could be to apply Gradient Boosting (XGBoost, LightGBM) after you extract CNN features. Essentially, you use a CNN for feature extraction, then apply Gradient Boosting to those features instead of a fully connected layer, which may provide a different perspective or performance boost for classification.

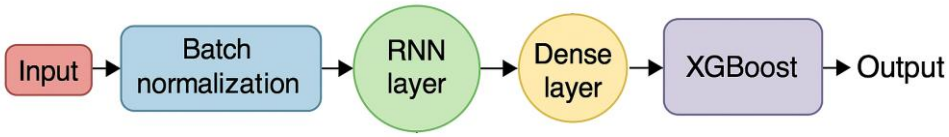


Figure: Integrating RNN and Gradient Boosting

Ensemble Approach: Combine an RNN with Gradient Boosting as part of an ensemble learning approach, where you might use the RNN's output as one of the inputs to a Gradient Boosting classifier.

6. Why Use GB-RNN for CIFAR-10?

The combination of GRU and Gradient Boosting could work well for specific challenges or experiments with non-traditional architecture. Here are some potential advantages of combining both methods:

Sequential Modeling: GRU or LSTM layers can capture long-term dependencies if there are sequential or spatial relationships in the feature map that are not well captured by CNNs alone.

Decision Trees in Gradient Boosting: Decision trees in Gradient Boosting can be highly interpretable and robust, especially when there are patterns in the extracted features that benefit from tree-based learning.

Improved Performance: Combining RNNs with Gradient Boosting might lead to better performance in some cases if the RNN can learn temporal/spatial patterns that Gradient Boosting models can then classify more effectively.

Train the combined architecture on the CIFAR-10 dataset, using cross-validation and hyperparameter tuning for both the RNN and Gradient Boosting components.

- Fully Connected (Dense) Layers:
- A dense layer with 128 units and ReLU activation adds non-linearity.
- The final dense layer with 10 units uses XGBoost activation for multi-class classification.

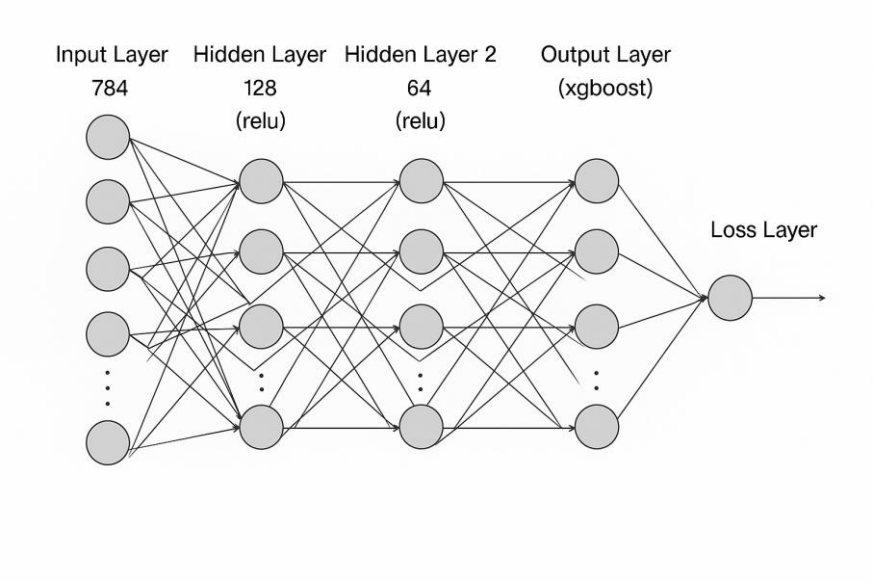


Figure: RNN training model

- The model is compiled with the Adam optimizer, sparse categorical cross-entropy (used for multi-class classification with integer labels), and accuracy as a metric.
- You train the model using model.fit() for 50 epochs, with the training data x_train_seq and labels y_train, and validate on x_test_seq and y_test.
- **Epochs: 50** epochs might be a bit much, especially if you're training on large datasets. You might want to experiment with early stopping to avoid overfitting.
- **Input Shape**: Ensure that your data (x_train_seq, x_test_seq) matches the input shape (32, 96)—32 time steps and 96 features per step.

- **Output Classes:** Ensure that y_train and y_test contain integer labels from 0 to 9 since you're using sparse_categorical_crossentropy.

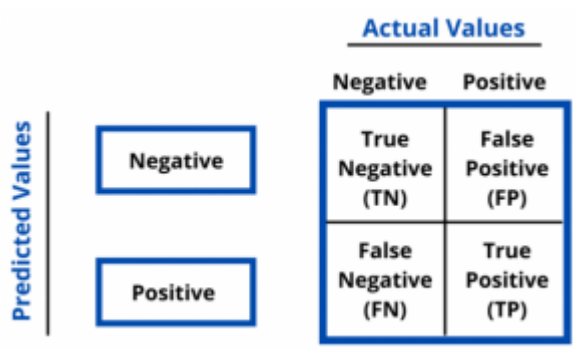
Evaluation and Test

- Test the model on the CIFAR-10 test set.
- Evaluate metrics like accuracy, precision, and recall for classification.

4.5 Performance Evaluation Metrics for GB-RNN and GB-DNN

To assess the effectiveness of the proposed GB-RNN and GB-DNN architectures, a range of evaluation metrics was applied, offering insights into classification behavior and model reliability. These metrics go beyond overall accuracy, focusing on how well individual classes are predicted and how errors are distributed across categories.

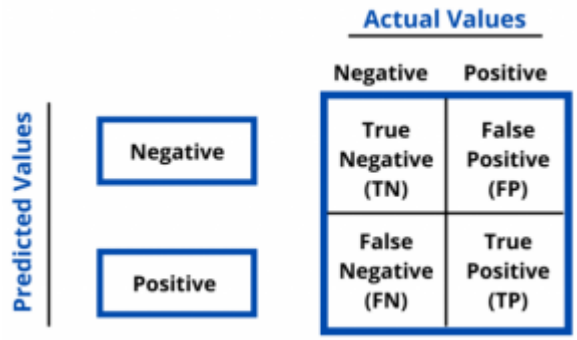
Confusion Matrix: The confusion matrix presents a class-wise summary of prediction outcomes, showing how many instances of each class were correctly or incorrectly labeled. Each cell of the matrix quantifies the number of samples predicted for a given class versus the actual class, enabling identification of class-specific misclassifications and patterns of confusion across similar categories.



confusion matrix:

```
[[665  29  59  24  21  15   9  26 107  45]
 [ 37 703  17  18   1  10   9  21  51 133]
 [ 80  17 448  99  83  93  65  77  16  22]
 [ 22  21  89 381  32 246  75  81  20  33]
 [ 40   5 175  61 444  65  59 119  23   9]
 [ 22   7  68 210  44 486  48  90  10  15]
 [ 11  11 106 108  61  74 576  22  11  20]
 [ 39   2  48  51  53  87  12 671  13  24]
 [119  41  12  25  13  10   5   8 708  59]
 [ 39 159  15  25   5  14  14  33  57 639]]
```

Figure: GB-RNN Confusion Matrix



confusion matrix:

```
[[798  39  22  15   5   3   6  10  69  33]
 [ 27 873   3   2   3   3   4   2  24  59]
 [120  24 492  68 108  69  51  36  13  19]
 [ 49  28  70 448  81 183  44  37  26  34]
 [ 66  20  96  62 566  22  39 102  15  12]
 [ 34  17  56 169  56 556  18  65  11  18]
 [ 18  24  66  87  58  12 689  13  10  23]
 [ 34  12  24  40  62  55   2 736   2  33]
 [137  60   9   7   8   5   2   4 744  24]
 [ 38 183   6  11   5   4   4   9  31 709]]
```

Figure: GB-DNN Confusion Matrix

Accuracy: Accuracy captures the proportion of correct predictions over the total number of inputs. While commonly used, it may not reflect true performance in imbalanced datasets. However, since the CIFAR-10 dataset used in this study contains uniformly distributed classes, accuracy serves as a reliable primary indicator:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Classification report is commonly used to evaluate the performance of a deep learning model for classification tasks. It typically includes the following metrics for each class (or label) in your dataset:

Precision: Precision evaluates the proportion of predictions labeled as a particular class that were actually correct. It answers the question: Out of all the samples predicted as class X, how many were truly class X? This metric is especially useful when minimizing false positives is crucial, such as in applications requiring high confidence predictions:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$$

Recall: Recall assesses the model's ability to detect all instances of a given class. It answers: Out of all actual samples belonging to class Y, how many did the model correctly identify? Higher recall indicates better sensitivity and is important when missing instances of a class is more costly than over-predicting it.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

F1-measure: The F1-score provides a balance between precision and recall. It is particularly useful when neither metric can be optimized independently without affecting the other. By combining both into a harmonic mean, the F1-score reflects a more balanced view of a model's predictive strength, especially in multi-class tasks like CIFAR-10:

$$\text{F1-Score} = 2 \text{ x } \frac{\text{Precision x Recall}}{\text{Precision+Recall}}$$

Support: Support refers to the actual occurrences of each class in the test set. Although not a performance metric, it provides context for interpreting precision, recall, and F1-score values—highlighting which classes dominate the dataset and which are underrepresented.

Classification Report:

	precision	recall	f1-score	support
0	0.62	0.67	0.64	1000
1	0.71	0.70	0.70	1000
2	0.43	0.45	0.44	1000
3	0.38	0.38	0.38	1000
4	0.59	0.44	0.51	1000
5	0.44	0.49	0.46	1000
6	0.66	0.58	0.62	1000
7	0.58	0.67	0.62	1000
8	0.70	0.71	0.70	1000
9	0.64	0.64	0.64	1000
accuracy			0.57	10000
macro avg	0.57	0.57	0.57	10000
weighted avg	0.57	0.57	0.57	10000

Figure: GB-RNN Classification Report

Classification Report:

	precision	recall	f1-score	support
0	0.60	0.80	0.69	1000
1	0.68	0.87	0.77	1000
2	0.58	0.49	0.53	1000
3	0.49	0.45	0.47	1000
4	0.59	0.57	0.58	1000
5	0.61	0.56	0.58	1000
6	0.80	0.69	0.74	1000
7	0.73	0.74	0.73	1000
8	0.79	0.74	0.77	1000

Figure: GB-DNN Classification Report

4.6 Training and Validation Monitoring of GB-RNN and GB-DNN

Throughout model training, both GB-RNN and GB-DNN architectures were evaluated at each epoch to track improvements and identify overfitting trends. Monitoring was carried out using the following elements:

- **Training Accuracy and Loss:** Observes how well the model fits the training data over time.
- **Validation Accuracy and Loss:** Assesses generalization performance on unseen data.

- **Performance Curves:** Epoch-wise plots of accuracy and loss reveal learning patterns and convergence behavior. A steady increase in accuracy and decrease in loss is an indicator of effective training.
- **Early Stopping Criteria:** Integrated mechanisms were used to halt training when validation performance ceased improving, preventing unnecessary computations and mitigating overfitting.

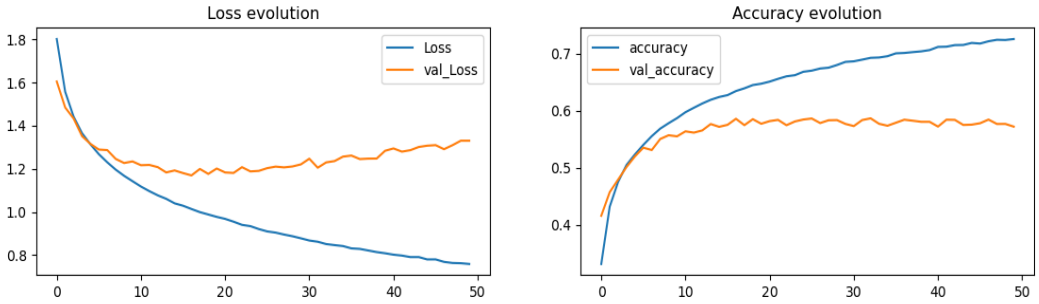


Figure GB-RNN Loss Evolution and Accuracy Evolution

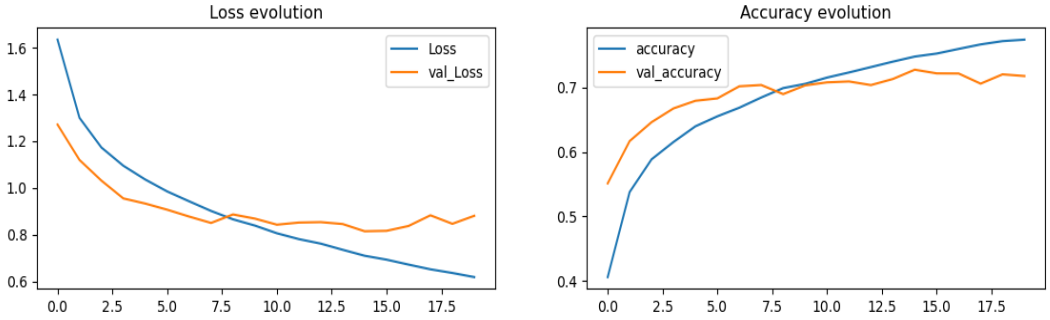


Figure: GB-DNN Loss Evolution and Accuracy Evolution

4.7 Calculate ROC curve for each GB-RNN and GB-DNN

To calculate and plot the ROC (Receiver Operating Characteristic) curve for each class in a multi-class classification problem, you can use the roc_curve function from Scikit-learn. The ROC curve helps to visualize the performance of a classification model at different thresholds, and it is typically used for binary classification. However, for multi-class problems, you can calculate the ROC curve for each class by considering each class as a "positive" class and the rest of the classes as "negative" (One-vs-Rest approach). Here is how you can compute and plot the ROC curve for each class using Scikit-learn:

Steps:

- **Train the Model***:* You need a trained classifier like GB-RNN.
- *Obtain Class Probabilities:* Use predict_proba() to get the predicted probabilities for each class.
- **Calculate the ROC Curve:** For multi-class classification, you need to use the One-vs-Rest method. This means calculating the ROC curve for each class by treating that class as the positive class and all other classes as negative.
- **Plot the ROC Curve:** Plot the ROC curve for each class, along with the AUC (Area under the Curve), which is a common metric for ROC curve evaluation.

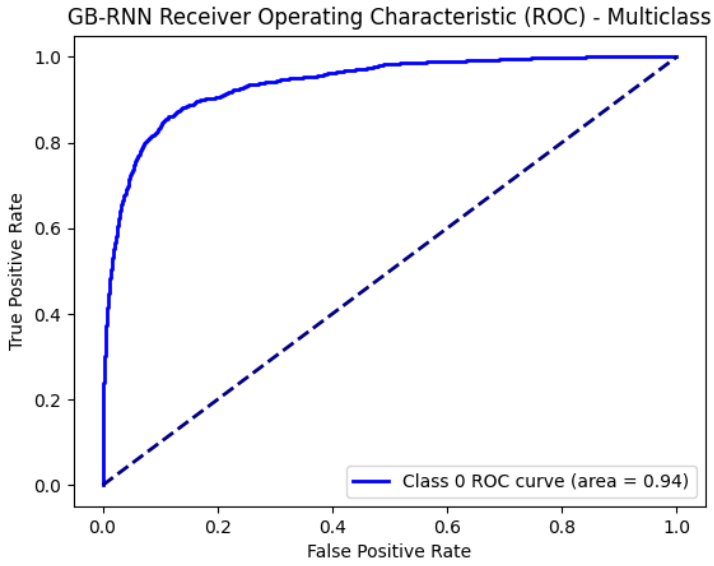


Figure: GB-RNN Receiver Operating Characteristic (ROC) - Multiclass

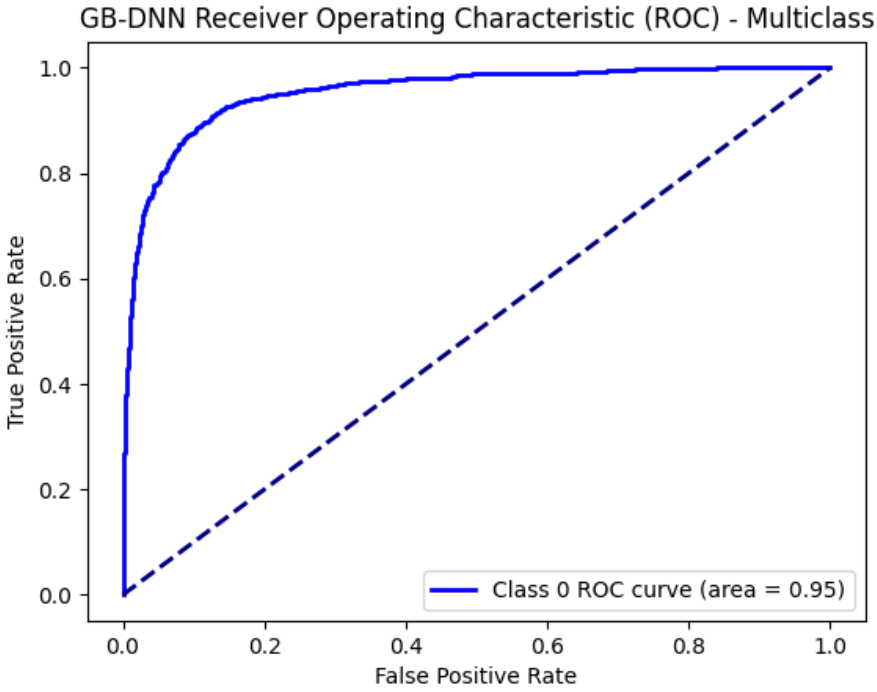


Figure: GB-DNN Receiver Operating Characteristic (ROC) – Multiclass

4.8 Detailed Explanation of Training Image Classification Using GB-RNN and GB-DNN

4.8.1. Dataset Preprocessing

The image classification workflow initiates with an essential stage—dataset preprocessing which significantly influences the model's accuracy and its ability to generalize across unseen data. The dataset is systematically split into three distinct parts: training, validation, and testing. This separation enables the model to be trained efficiently, fine-tuned using validation metrics, and objectively assessed using an untouched test set.

One of the key steps in preprocessing involves scaling pixel values to the [0, 1] range. This transformation converts raw image data into a normalized format that is more suitable for neural network models, promoting faster convergence and stable learning behaviour. To prevent the model from overfitting and to simulate real-world variability, data augmentation methods are employed. Techniques such as random flipping, rotation, translation, and zooming are introduced to artificially expand the dataset. These variations serve as a form of regularization by exposing the model to a broader spectrum of input patterns during training.

4.8.2. Model Architecture

The model's design incorporates a sequence of convolutional layers that progressively learn and extract high-level visual patterns from input images. These layers are arranged in depth to capture increasingly abstract representations. To streamline computations and retain vital spatial information, pooling operations, most commonly max pooling, are applied following the convolutional stages. This not only reduces dimensionality but also enhances the model's efficiency.

To enable the network to capture non-linear and intricate relationships within the data, **Rectified Linear Unit (ReLU)** activations are utilized throughout the hidden layers. To further enhance generalization and mitigate the risk of overfitting, dropout techniques are incorporated, where neurons are randomly disabled during training sessions. This encourages the model to develop redundant pathways, fostering robustness.

For the final prediction layer, the architecture integrates an **XGBoost**-based classifier, which excels in multi-class classification scenarios. It generates a set of probability scores, one for each class, ensuring that all values collectively sum to one. This configuration aligns well with the structure of the CIFAR-10 dataset, which contains ten mutually exclusive image categories.

4.8.3. Training Process

During training, the model is compiled with **categorical cross-entropy loss**, a suitable loss function for multi-class classification tasks. Optimizers such as **Adam** or **RMSprop** are used to minimize the loss function, leveraging techniques like adaptive learning rates to enhance convergence.

Hyperparameters, including learning rate and batch size, are carefully tuned to balance the speed and stability of training. Model checkpoints are periodically saved to ensure progress is not lost and to allow for restoring the best-performing model during training.

Validation on a separate dataset during training provides insights into the model's generalization capabilities. Monitoring metrics such as validation accuracy and loss helps identify overfitting or underfitting trends, guiding further adjustments to the architecture or hyperparameters.

Input Design

The input design serves as the interface between the user and the GB-RNN and GB-DNN systems. It involves developing specifications and procedures for data preparation to ensure that input data is in a usable form for processing. This can be achieved through direct user input or automated data extraction from existing sources. The focus of input design is to minimize input errors, reduce complexity, and maintain security while ensuring user-friendliness. Key aspects of the input design include:

- Data Requirements: Identifying the data that needs to be provided as input, such as 2D image data or tabular datasets for training and evaluation.
- The CIFAR-10 dataset contains 60,000 32x32 color images in 10 classes, with 6,000 images per class. Classes include airplanes, cars, birds, cats, etc. It is commonly used for computer vision tasks.

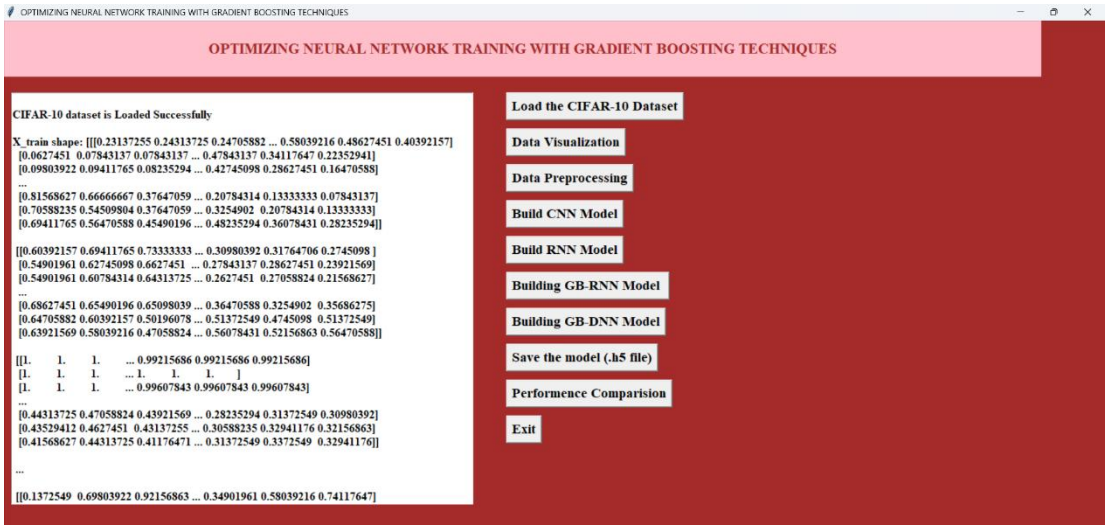


Figure: Load the CIFAR-10 Dataset

- Use tensorflow.keras.datasets.cifar10 or a similar library to load the dataset.
- Data Arrangement: Determining how input data should be arranged or coded, such as normalizing image pixel values to a range of [0, 1] and augmenting datasets for diversity.
- Guidance for Users: Providing clear instructions or dialogs to guide users in preparing and submitting input data.
- Visualization is crucial to understand the dataset distribution and verify its integrity.

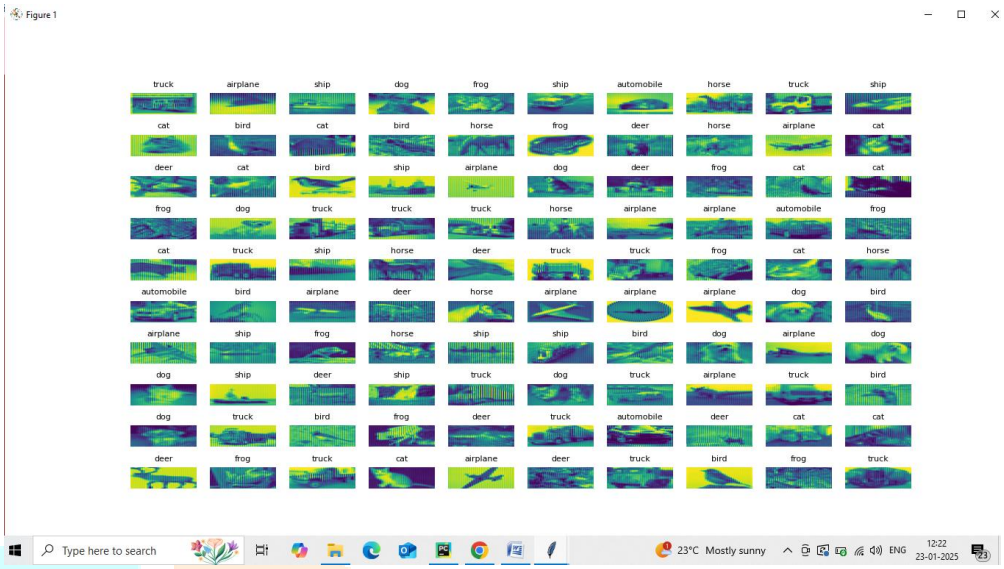


Figure: List of CIFAR-10 Images

Implementing methods for input validation to ensure data quality and steps to address errors when they occur.

Use **Matplotlib** or **Seaborn** to display example images from each class. Show the number of images per class to identify imbalances

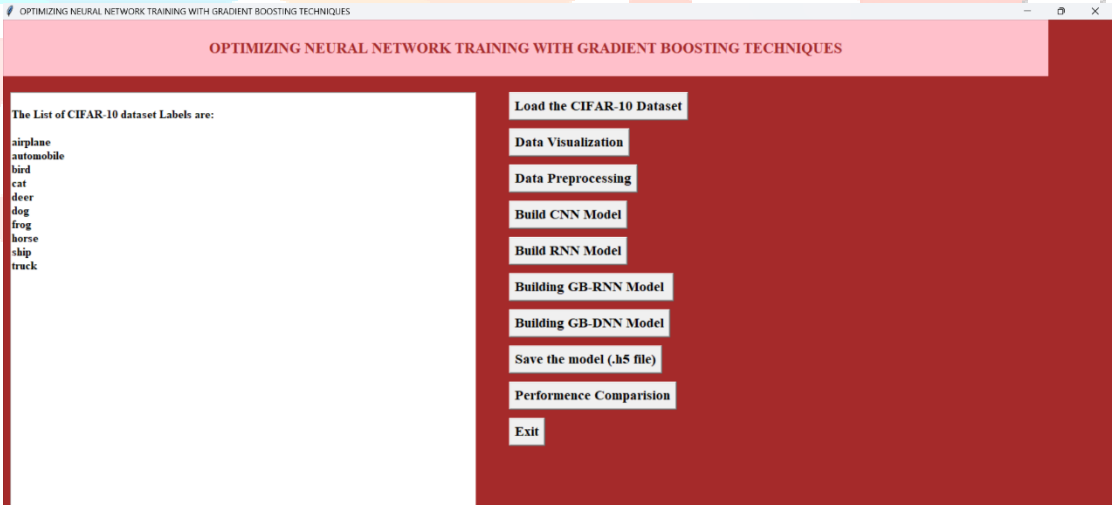


Figure: List of CIFAR-10 Dataset Labels

Preprocessing is essential to prepare data for input into a model.

Normalization: Scale pixel values to the range [0, 1] for faster convergence.

One-Hot Encoding: Convert class labels into a binary matrix

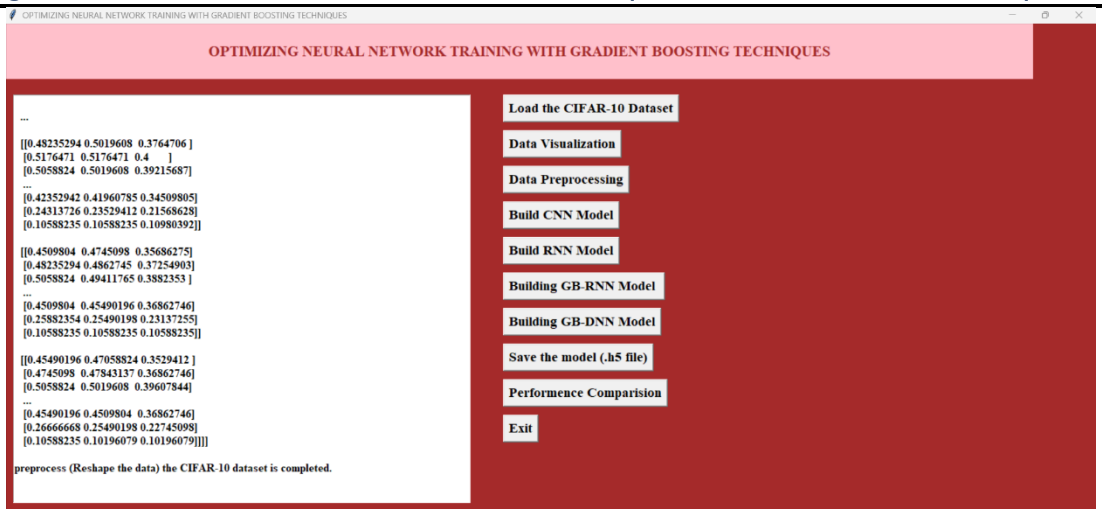


Figure: Data Preprocessing

Rescale pixel values from 0-255 to 0-1

Convert class labels into one-hot vectors for multi-class classification

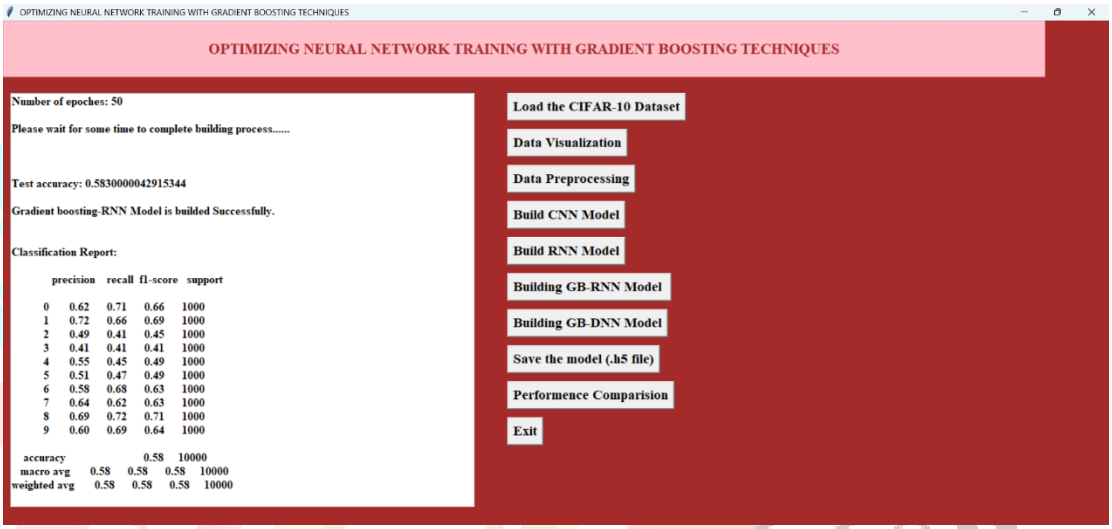


Figure: Building GB-RNN Model and calculate performance matrices

Implementation of building GB-RNN Model and calculate performance matrices like classification report (precision, recall, f1-score), Confusion matrix, Accuracy score.

Combine the strengths of Gradient Boosting (accuracy and robustness) with RNNs (sequential data processing).

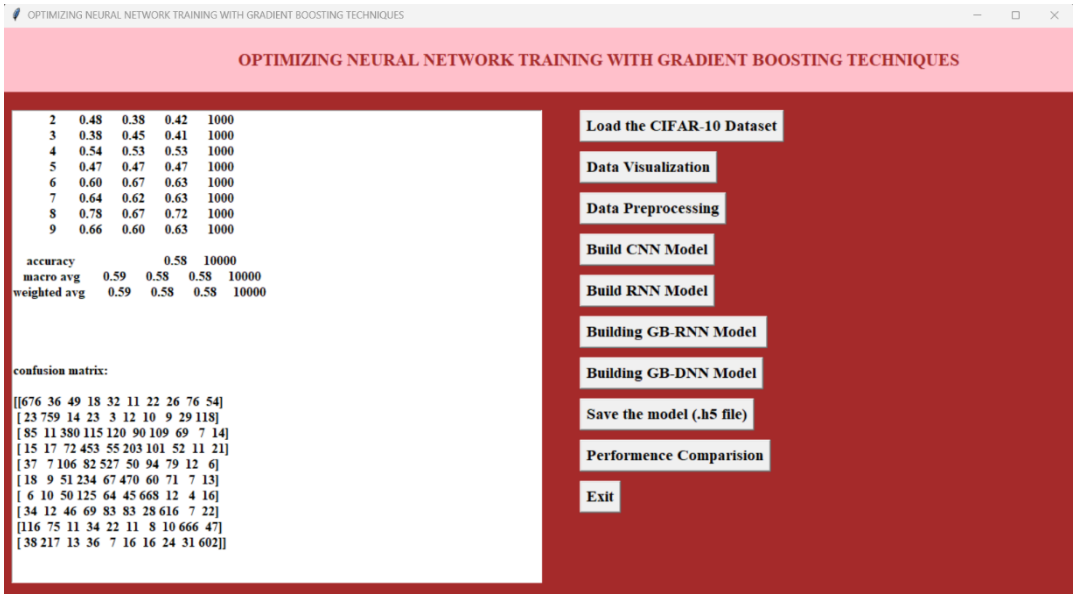


Figure: Building GB-RNN Model and calculate confusion matrix

Input layer processes image features (e.g., after reshaping).

RNN layers like LSTM/GRU extract temporal or sequential patterns.

Fully connected layers predict class probabilities.

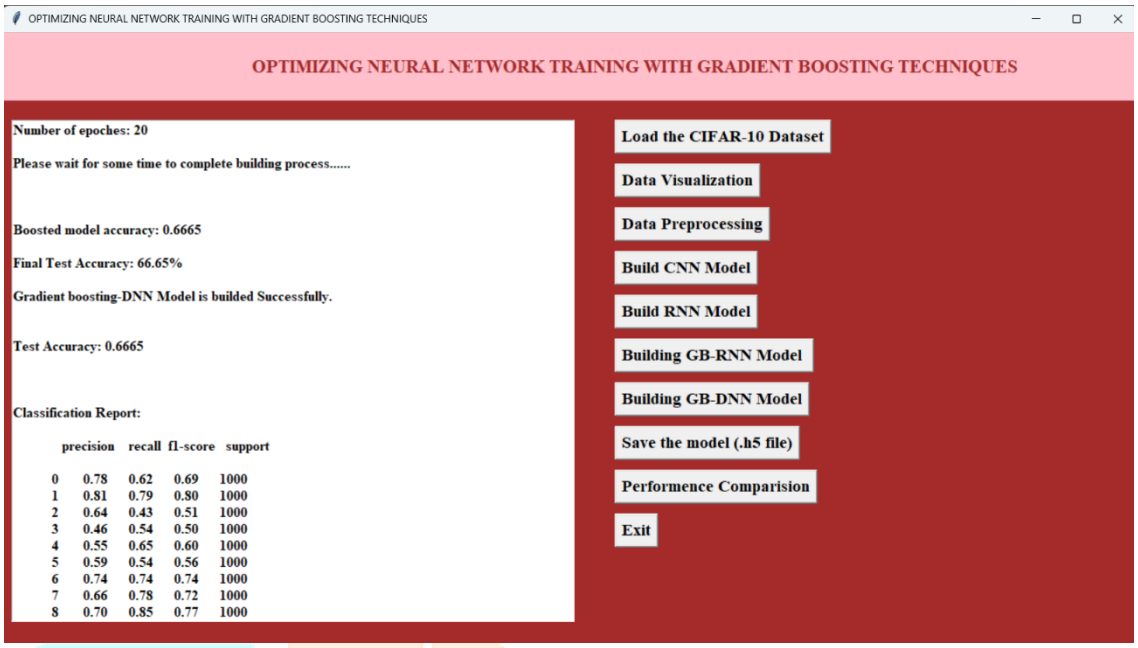


Figure: Building GB-DNN Model and calculate performance matrices

Use a Deep Neural Network (DNN) architecture with Gradient Boosting for enhanced feature extraction.

Dense layers extract high-dimensional features.

Dropout layers reduce overfitting.

Final output layer with a XGBoost activation.

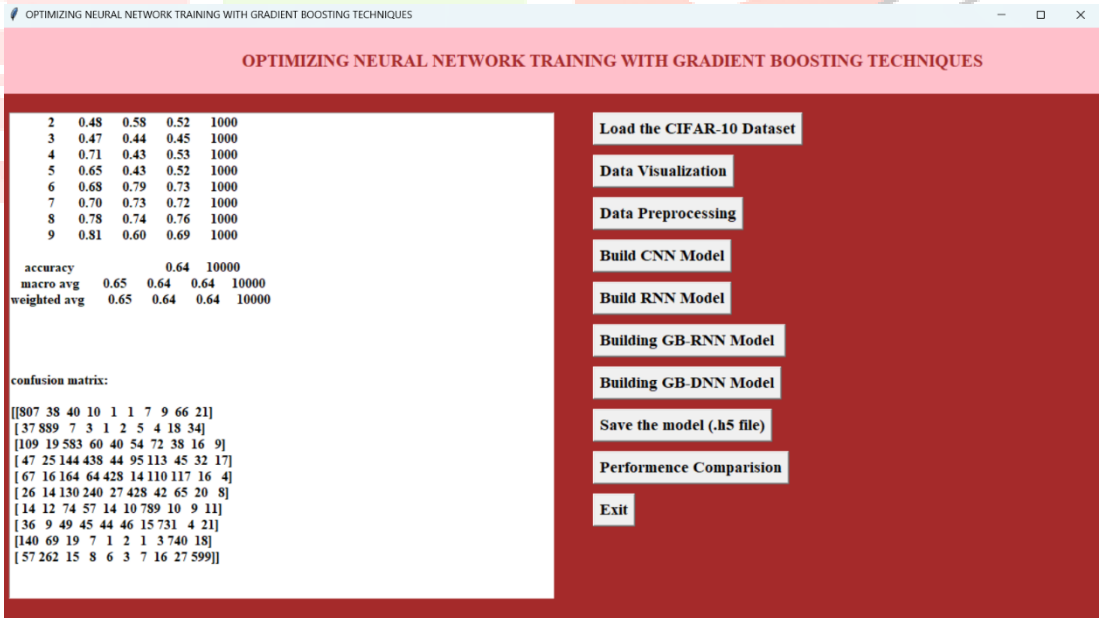


Figure: Building GB-DNN Model and calculate confusion matrix

Implementation of building GB-DNN Model and calculating performance matrices like classification report (precision, recall, f1-score), Confusion matrix, Accuracy score.

Combine the strengths of Gradient Boosting (accuracy and robustness) with RNNs (sequential data processing).

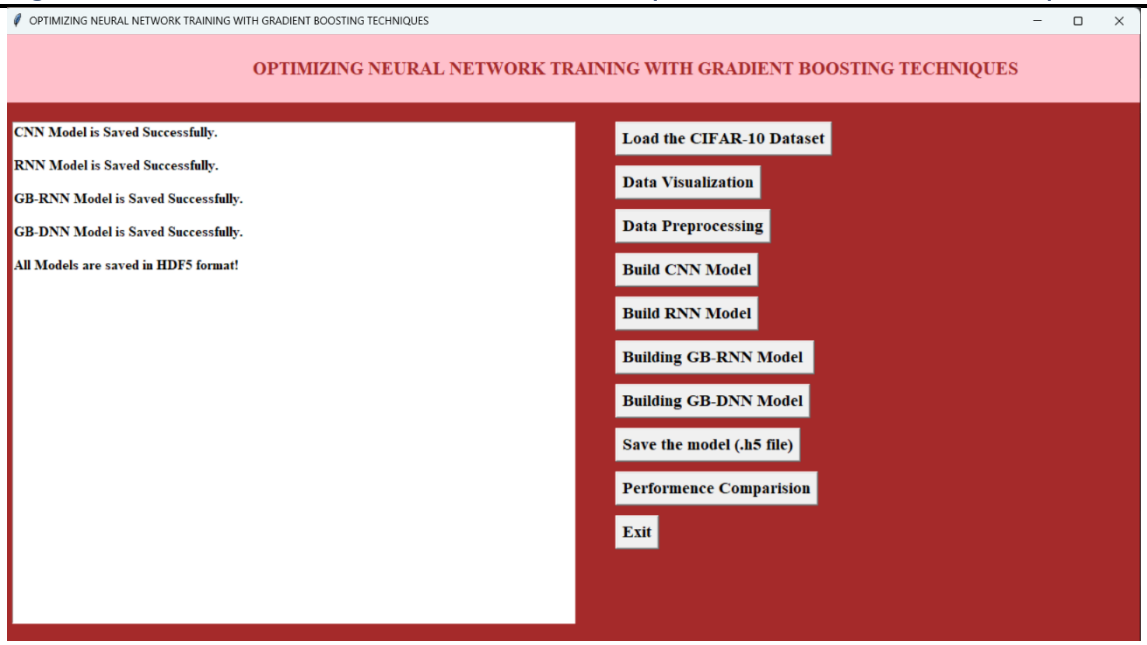


Figure: Save the Model

- Save the trained model for reuse without retraining:
- Later, load the saved model using: tensorflow.keras.models

Output Design

The output design focuses on delivering quality results from the GB-RNN and GB-DNN systems in a manner that meets the users' needs. Outputs communicate the results of the models' processing to the end users, aiding in decision-making and providing actionable insights. The design ensures that outputs are clear, accurate, and useful for immediate interpretation or further analysis.

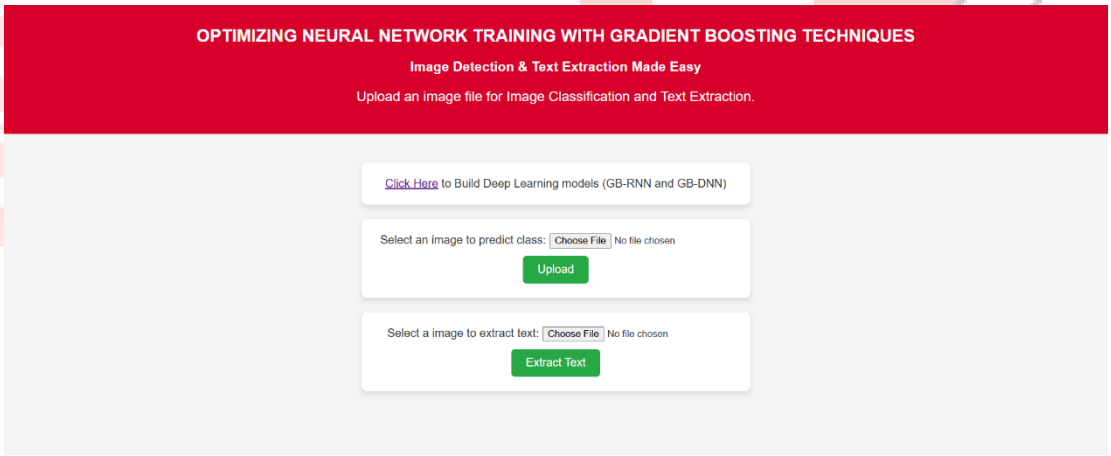


Figure: Image Detection Web UI

Organized Development: Outputs are designed systematically to ensure usability and effectiveness. Outputs such as model accuracy, loss metrics, and classification results are clearly presented.

Methods of Presentation: Information is displayed in visual formats, such as tables, charts, or confusion matrices, to highlight classification accuracy and error analysis.

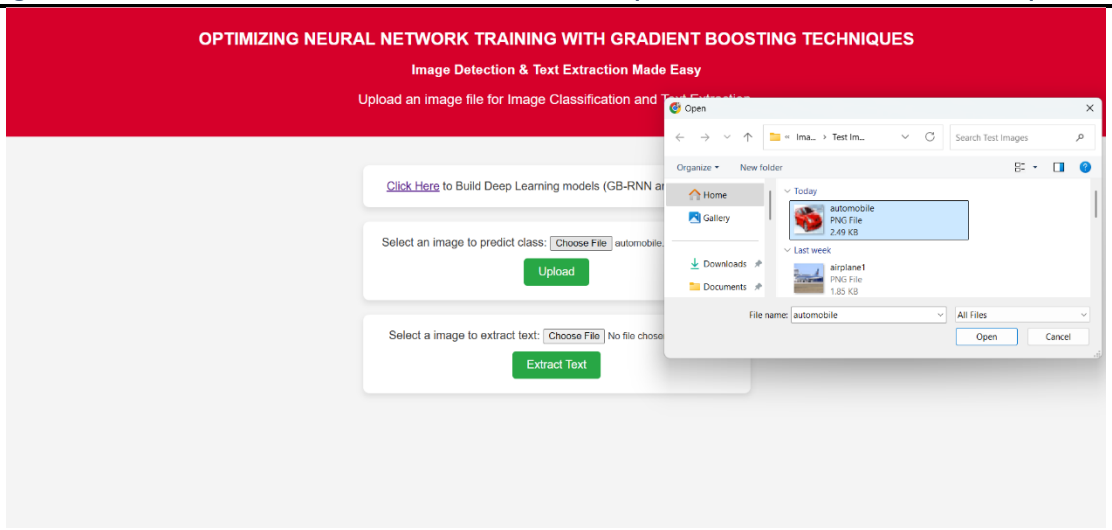


Figure: Choose Image to Detect

Output Formats: The outputs include reports or dashboards summarizing key metrics such as accuracy, precision, recall, F1-score, and detailed performance breakdowns across classes.

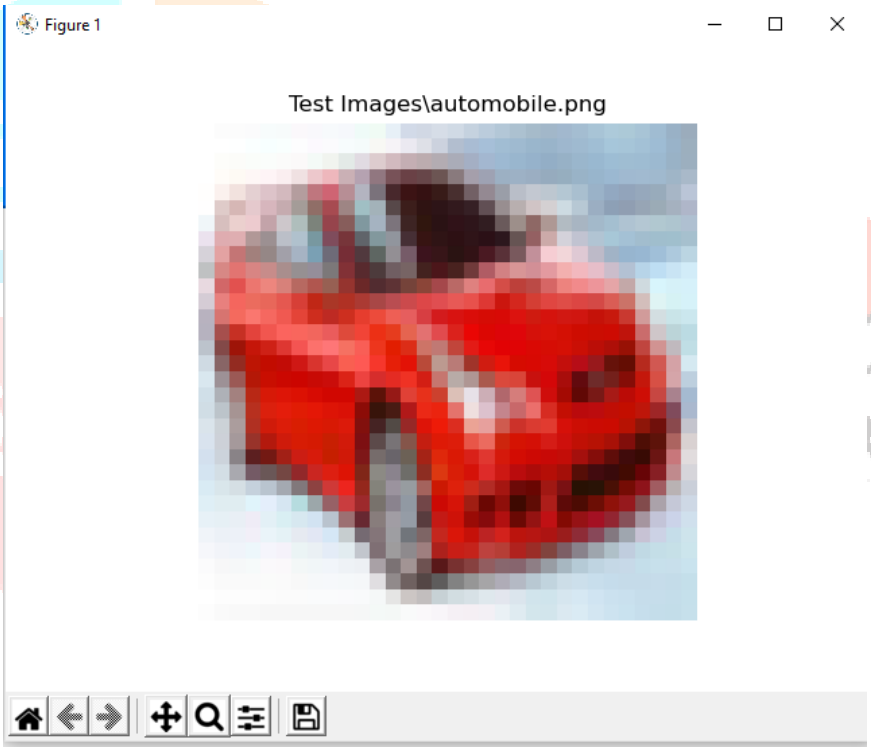


Figure: Test Image

Use the trained model to predict classes for new or test images.

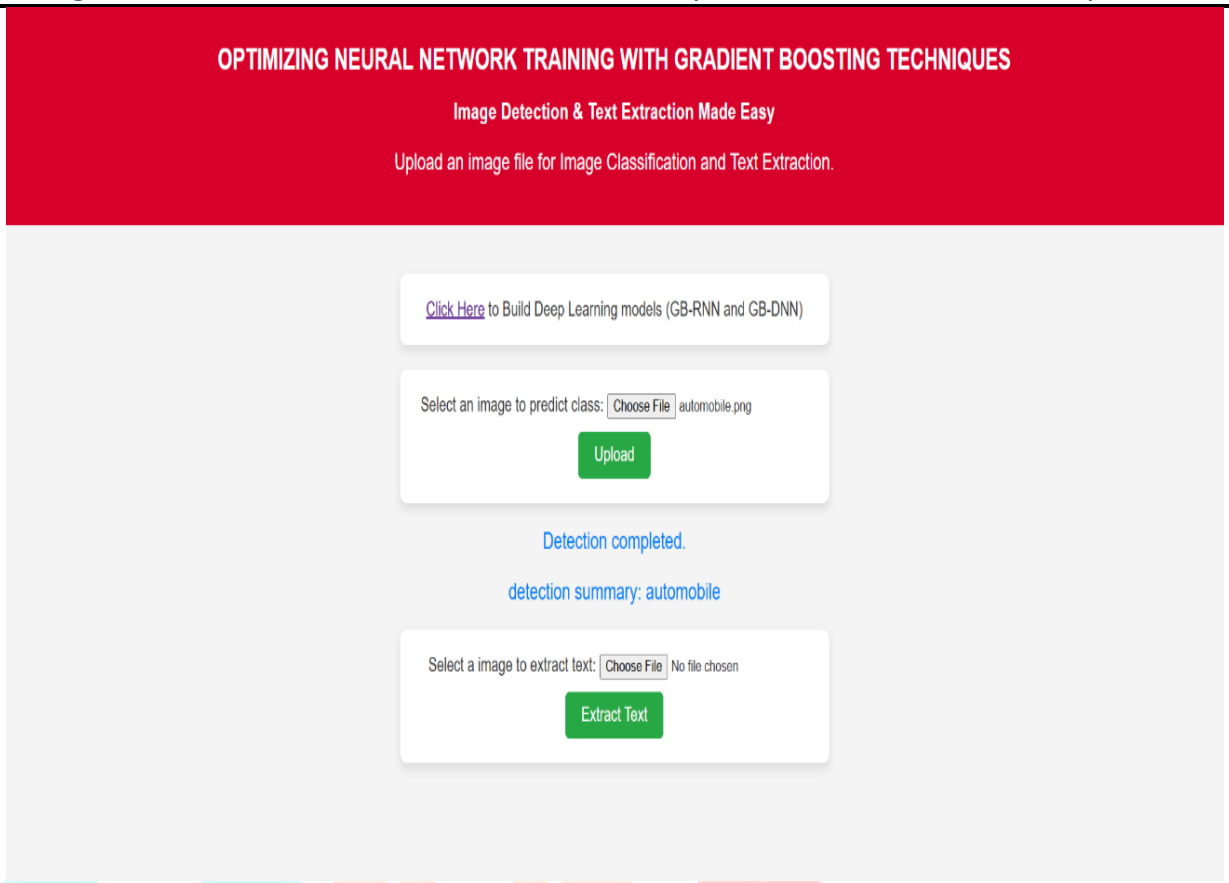


Figure: Output Detection

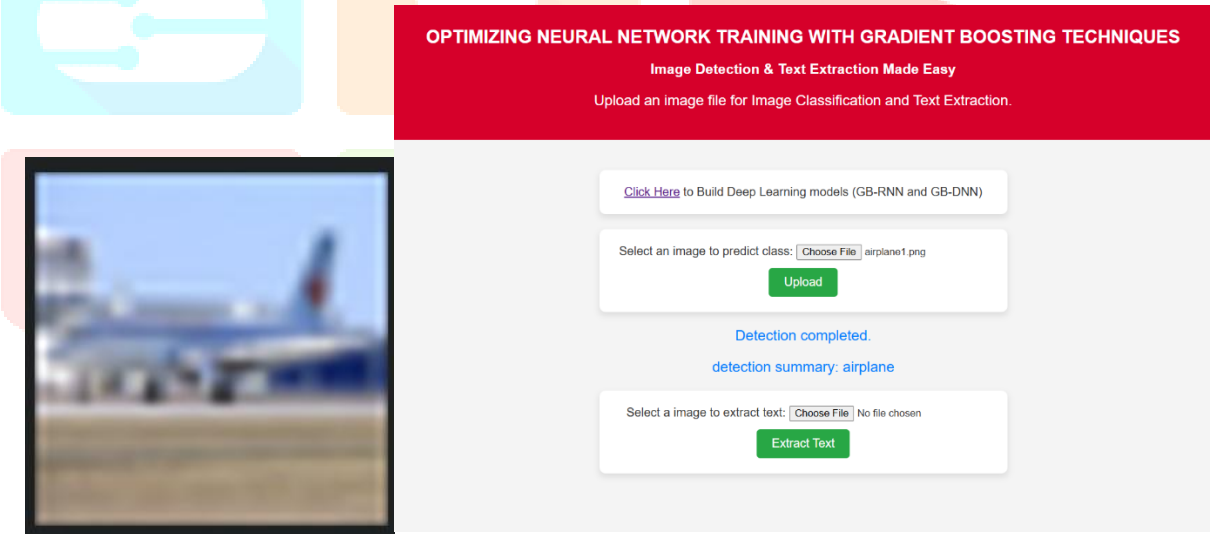


Figure: Test Image 2 and Output Detection

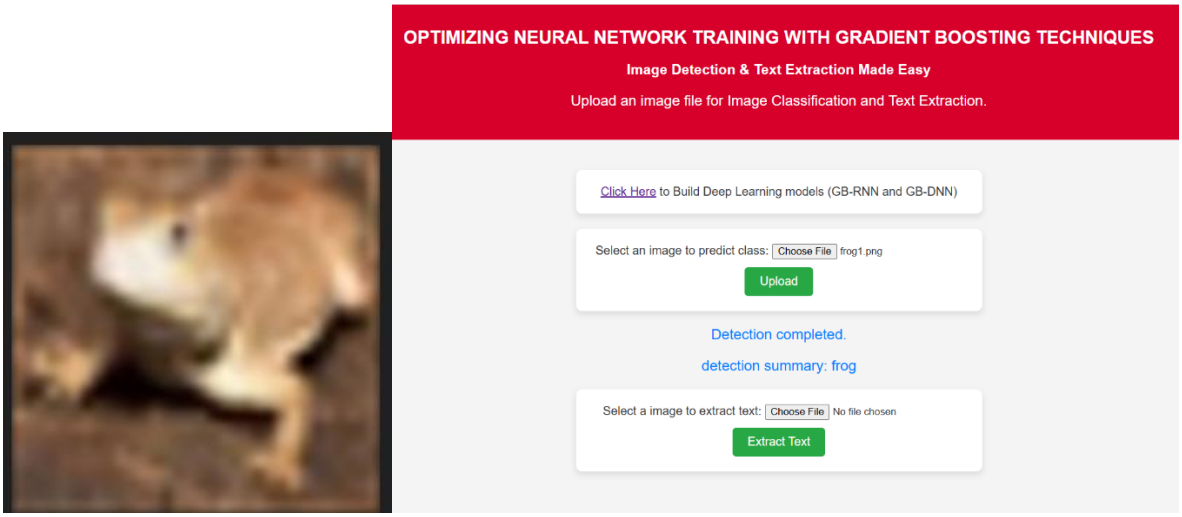


Figure: Test Image 3 and Output Detection

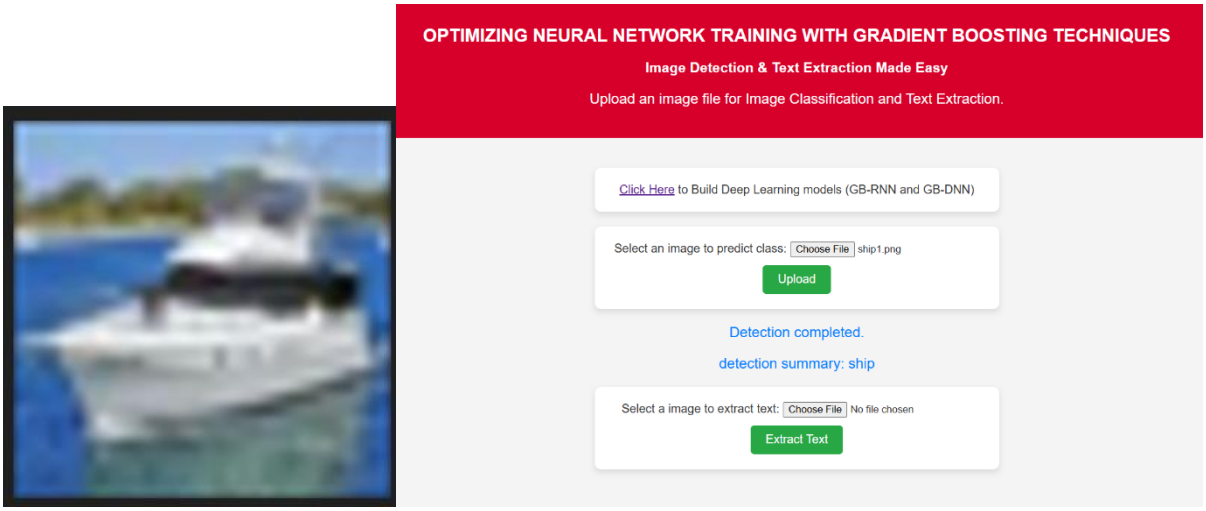


Figure: Test Image 4 and Output Detection

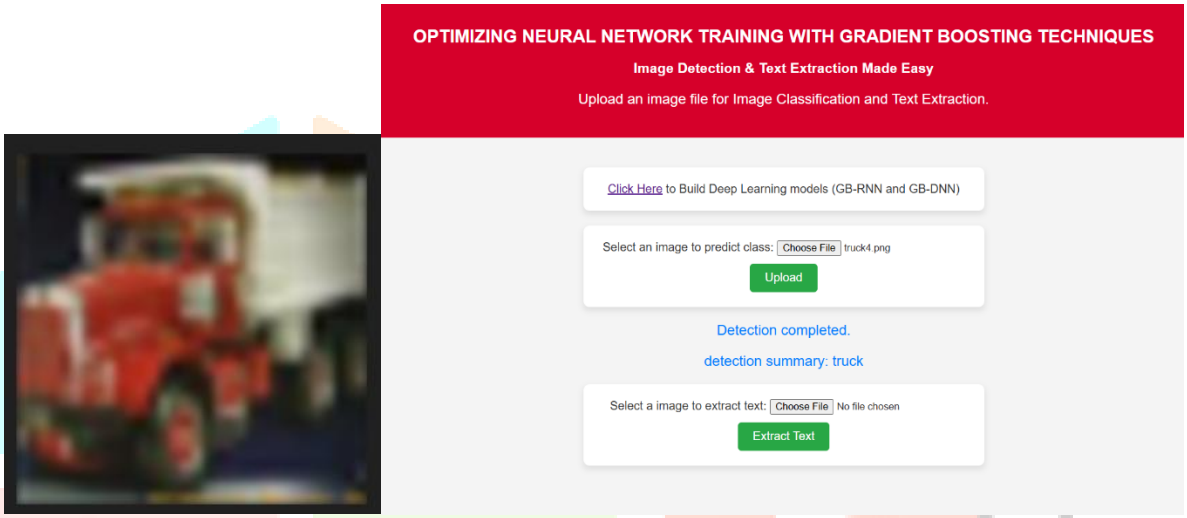


Figure: Test Image 5 and Output Detection

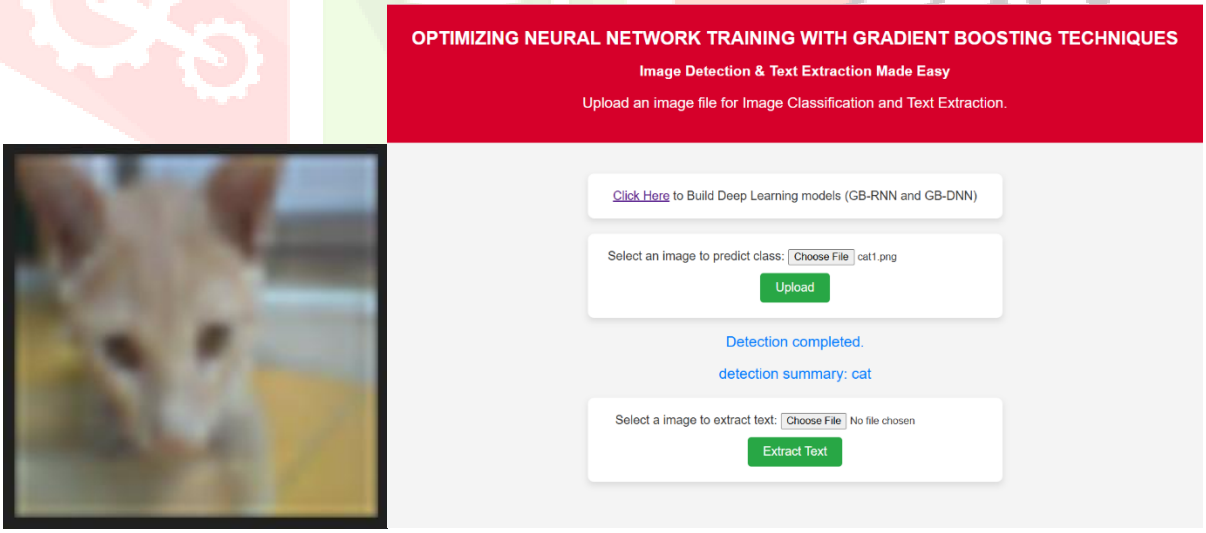


Figure: Test Image 6 and Output Detection

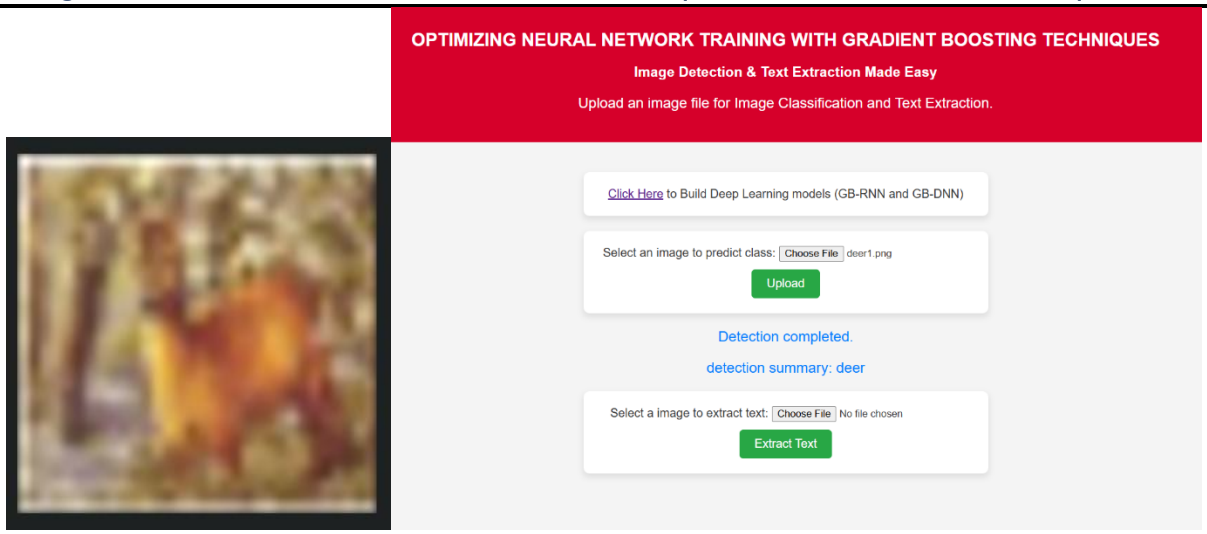


Figure: Test Image 7 and Output Detection

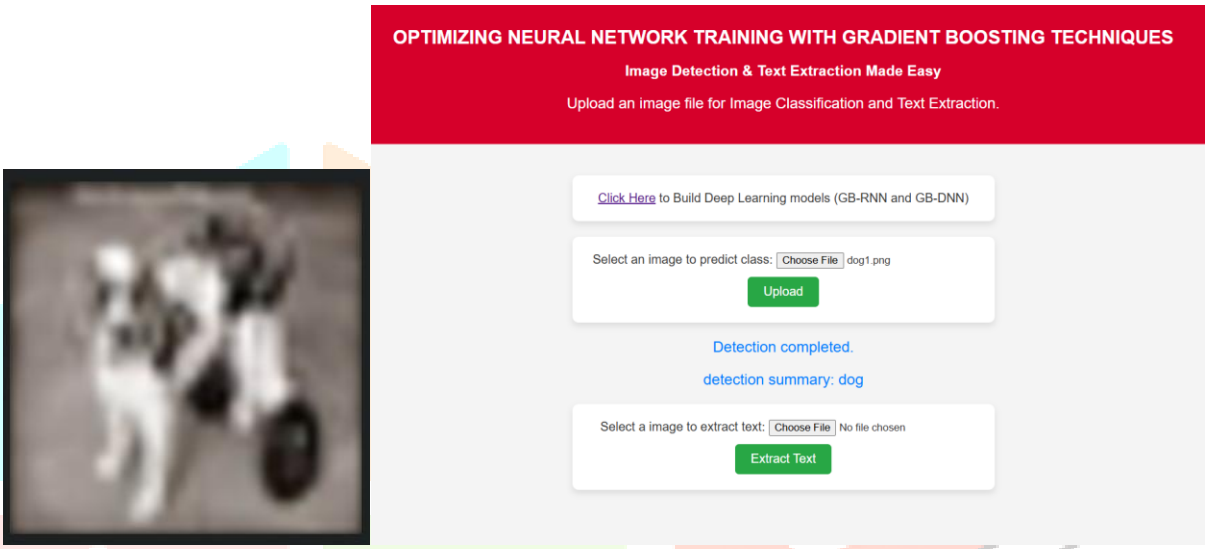


Figure: Test Image 8 and Output Detection

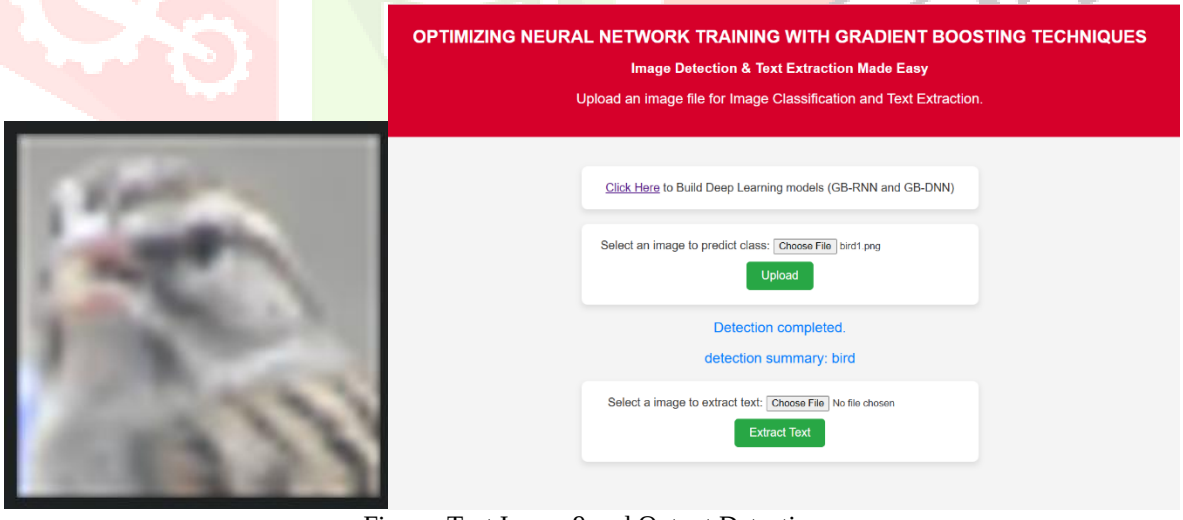


Figure: Test Image 9 and Output Detection

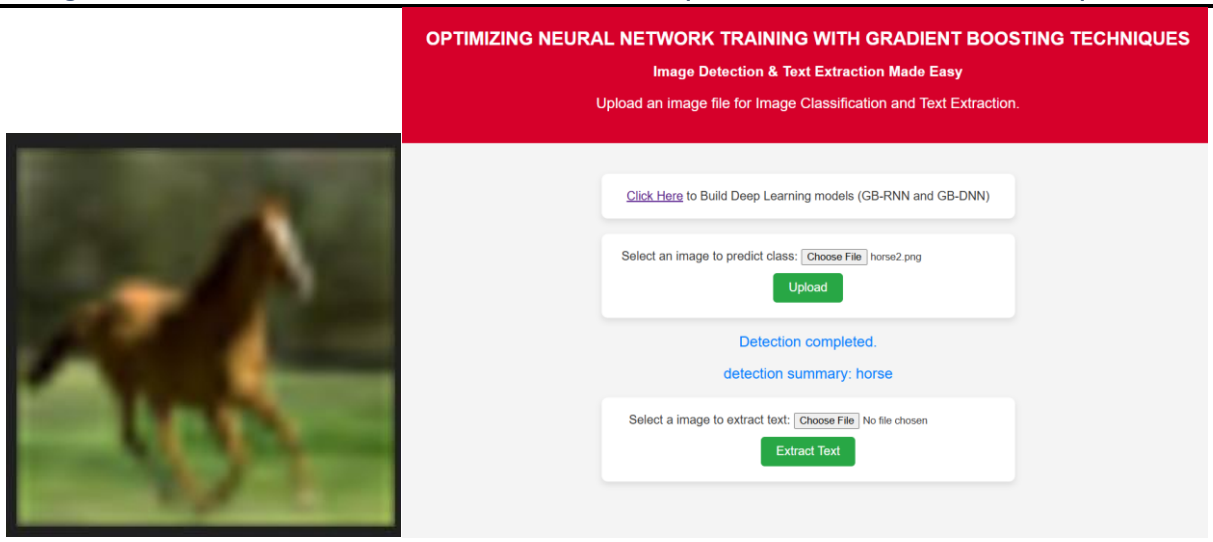


Figure: Test Image 10 and Output Detection

Visualize the predictions with corresponding images. Ensure adequate hardware for training deep models.

4.8.4. Comparative Analysis of GB-RNN vs. GB-DNN vs. RNN vs. CNN

Despite promising results, challenges emerged during training and evaluation. Overfitting, even with dropout regularization, remained a concern, indicating the potential need for additional techniques like weight regularization or early stopping. The small size of the CIFAR-10 dataset presented constraints, limiting the model's ability to generalize fully to unseen data.

Class-specific performance analysis revealed that certain classes achieved higher accuracy than others. This discrepancy highlights the difficulty of distinguishing visually similar categories, suggesting the need for further refinement in feature extraction or augmentation strategies tailored to specific classes.

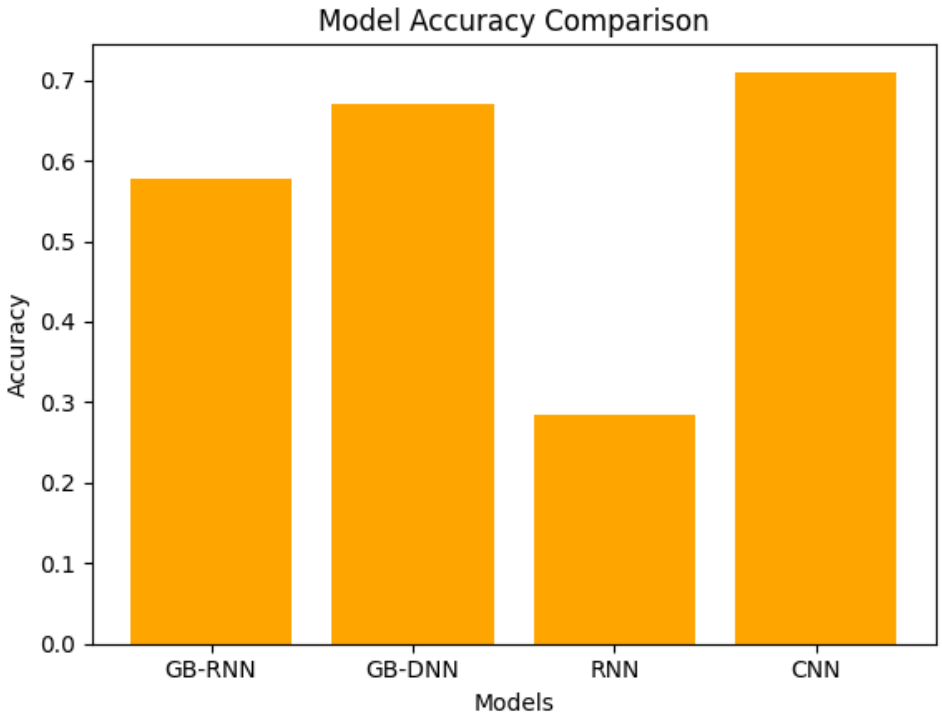


Figure: Performance Comparison Results

4.8.5. Future Directions

To enhance performance and address existing challenges, future work could explore:

The GB-RNN and GB-DNN models presented in this chapter introduce a novel gradient boosting-inspired training mechanism within neural network design. By learning in stages and focusing on mistakes, the models achieve improved convergence, greater interpretability, and better generalization compared to monolithic deep networks. This modular approach to network growth not only enhances performance but also enables easy scalability and adaptability across domains.

5. CONCLUSION AND FUTURE SCOPE

5.1. Conclusion

This project introduces an innovative strategy for training deep neural networks through two custom architectures—Gradient Boosted Recurrent Neural Network (GB-RNN) and Gradient Boosted Deep Neural Network (GB-DNN). By integrating gradient boosting principles into neural network training, these models are designed to iteratively refine learning by focusing on the residual errors left behind in earlier stages. A key feature of both models is the use of dense layer freezing, which reduces training complexity and improves learning stability by retaining learned representations across iterations.

The practical impact of the proposed architecture was evaluated across diverse datasets, including 2D image-based tasks and structured tabular data. These datasets encompassed domains such as radar signal analysis, handwritten digit classification, agricultural prediction, and fashion item recognition. Results demonstrated the following:

1. **GB-RNN Highlights:**

Delivered consistently higher accuracy than conventional RNN and CNN models across all image-based tasks. Showed strong feature learning capabilities by fine-tuning convolutional layers during gradient boosting cycles.

2. **GB-DNN Highlights:**

Outperformed traditional DNN architectures on structured data benchmarks, showing improvements in both accuracy and F1-scores. Validated the advantage of freezing dense layers in reducing overfitting and speeding up convergence.

Moreover, convergence analysis indicated that only a few boosting iterations were needed to reach optimal performance. Even after traditional models reached their saturation point, GB-RNN and GB-DNN were able to extract additional improvements, emphasizing their adaptability and robustness for a wide variety of machine learning tasks.

5.2. Future Scope

The success of the GB-RNN and GB-DNN frameworks provides a strong foundation for further exploration in multiple directions:

1. **Scaling to High-Dimensional Data**: Extending these models to process large-scale and high-resolution image datasets, as well as massive, structured datasets in real-world applications.

2. **Architectural Expansion**: Explore integration with advanced neural models, such as Vision Transformers (ViTs), hybrid CNN-RNN combinations, or self-attention-based systems.

3. **Exploration of Additional Domains**: Apply the proposed frameworks to new domains including natural language processing (NLP), real-time diagnostics in healthcare, financial forecasting, and autonomous navigation systems.

4. **Optimization Enhancements**: Investigate dynamic training techniques such as adaptive learning schedules, smarter regularization, and layer-wise freezing/unfreezing mechanisms to improve both efficiency and accuracy.

5. **Deployment in Real-Time Systems**: Evaluate the suitability of GB-RNN and GB-DNN for edge devices or latency-sensitive applications, focusing on lightweight deployment and fast inference.

6. **Theoretical Exploration**: Analyze the theoretical underpinnings of convergence behavior and error correction dynamics within the GB-based training loop to better understand model performance.

By addressing these future directions, the proposed GB-RNN and GB-DNN models have the potential to evolve into versatile, high-performing solutions across a wide range of artificial intelligence challenges, making them suitable for both academic research and industrial deployment.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.

[2] D. Han, Q. Liu, and W. Fan, "A new image classification method using CNNtransfer learning and web data augmentation," Expert Syst. Appl., vol. 95, pp. 43–56, 2018.

[3] S. Ren, K. R. H. Girshick, and J. Sun, "Faster R-CNN: Towards real time object detection with region proposal networks," in Proc. Adv. Neural Inf. Process. Syst., 2015, vol. 28, pp. 91–99.

[4] T. Kim, S. C. Suh, H. Kim, J. Kim, and J. Kim, "An encoding technique for CNN-based network anomaly detection," in Proc. IEEE Int. Conf. Big Data, 2018, pp. 2960–2965.

[5] K.Kamnitsasetal., "Efficient multi-scale 3D CNN with fully connected CRFforaccurate brain lesion segmentation," Med. Image Anal., vol. 36, pp. 61–78, 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proc. 3rd Int. Conf. Learn. Representations, May 7-9, 2015.

[8] D. M. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in Proc. 13th Eur. Conf. Comput. Vis., 2014, pp. 818–833.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proc. 32nd Int. Conf. Mach. Learn., 2015, pp. 448–456.

[10] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proc. 13th Int. Conf. Artif. Intell. Statist., 2010, pp. 249–256.

[11] Jerome H. Friedman, "Greedy function approximation: A gradient boosting machine," Ann. Statist., vol. 29, no. 5, pp. 1189–1232, 2001.

[12] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in Proc. Adv. Neural Inf. Process. Syst., 1999, vol. 12, pp. 512–518.

[13] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in Proc. Adv. Neural Inf. Process. Syst., 2017, vol. 30, pp. 3146–3154

[14] T. Chen et al., "Xgboost: Extreme gradient boosting," R Package Version 0.4-2, vol. 1, no. 4, pp. 1–4, 2015.

[15] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," Inf. Fusion, vol. 81, pp. 84–90, 2022.

[16] C. Bentéjac, A. Csörgö, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," Artif. Intell. Rev., vol. 54, pp. 1937–1967, 2021.

[17] S. Emami and G. Martýnez-Muñoz, "Sequential training of neural networks with gradient boosting," IEEE Access, vol. 11, pp. 42738–42750, 2023.

[18] F. Huang, J. Ash, J. Langford, and R. Schapire, "Learning deep ResNet blocks sequentially using boosting theory," in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 2058–2067.

[19] A.Nitanda andT.Suzuki, "Functional gradient boosting based on residual network perception," in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 3819–3828.

[20] Y. Bengio, N. Roux, P. Vincent, O. Delalleau, and P. Marcotte, "Convex neural networks," in Proc. Adv. Neural Inf. Process. Syst., 2005, vol. 18, pp. 123–130.

[21] S. Emami and G. Martínez-Muñoz, "Multioutput regression neural network training via gradient boosting," in Proc. 30th Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn., 2022, pp. 145–150.

[22] S. Ö Arik and T. Pfister, "TabNet: Attentive interpretable tabular learning," in Proc. AAAI Conf. Artif. Intell., 2021, pp. 6679–6687.

[23] A. Abutbul, G. Elidan, L. Katzir, and R. El-Yaniv, "DNF-Net: A neural architecture for tabular data," 2020, arXiv:2006.06465.

[24] Y. Yang, I. G. Morillo, and T. M. Hospedales, "Deep neural decision trees," 2018, arXiv:1806.06988.