Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Predicting Security Breaches in AI-Powered Mobile Cloud Applications Using Deep Random Forest Algorithm

S. Hassan Abdul Cader[1], Dr. K. Nirmala[2]

[1]Research Scholar, Quaide Milleth Government College for Women, India, hassannew2002@gmail.com
[2]Supervisor, Pg & Research Department of Computer Science, Quaid-E-Millath Govt. College for Women, India, nimimca@gmail.com

This research addresses the need for a predictive approach to detect security breaches in AI-powered mobile cloud applications. We propose a novel approach combining Radial ResNet for advanced feature extraction with Random Forest (RF) for classification. This hybrid model is designed to analyze complex and dynamic datasets in real-time, enhancing the predictive capabilities for identifying potential security threats. Results indicate significant efficacy, with the algorithm achieving high accuracy and sensitivity in predicting security breaches. The Radial ResNet–RF combination achieved a training accuracy of 98.5%, with precision, recall, and F1-score values of 97.8%, 98.2%, and 98.0%, respectively. On testing and validation datasets, the model demonstrated accuracies of 96.2% and 95.5%, respectively.

Keywords: Mobile Cloud Applications, Security Breaches, Deep Random Forest Algorithm, Predictive Security.

1. Introduction

Mobile cloud applications are prone to diverse security threats such as data breaches, unauthorized access, and malicious attacks due to their reliance on cloud infrastructure and AI technologies [1]-[4]. The complexity of AI models, combined with the heterogeneous nature of cloud environments, exacerbates the challenge of securing these applications. AI models, particularly those used in mobile cloud settings, are susceptible to adversarial attacks and data poisoning, which can compromise their integrity and effectiveness [5]-[9]. Moreover, the large volume of data transmitted and processed in these applications creates numerous potential vulnerabilities that need to be monitored and managed effectively.

The primary problem addressed in this study is the need for a predictive approach to preemptively detect and mitigate security breaches in AI-powered mobile cloud applications [10]-[12]. Traditional security mechanisms often react to threats after they occur, which can be inadequate for the sophisticated and adaptive attacks targeting AI systems [13]-[14]. There is a pressing need for advanced predictive models that can anticipate potential security issues

before they manifest.

The objectives of this research are:

1. To develop and evaluate a novel predictive model for detecting security breaches in AI-powered mobile cloud applications.

2. To integrate Radial ResNet for feature extraction and Random Forest (RF) for classification to analyze complex and dynamic datasets in real-time.

3. To compare the performance of the proposed model with existing methods, including RAPNet-BPOA-DenseNet201, NIDPS with Self-Taught Learning (STL), and Multi-modal LSTM-DAE.

4. To assess the efficacy of the proposed approach in terms of accuracy, precision, recall, and F1-score, thereby demonstrating its potential to enhance security measures in mobile cloud environments.

The novelty of this research lies in the integration of Radial ResNet and Random Forest algorithms to create a robust predictive model for security breach detection. Radial ResNet's advanced feature extraction capabilities, combined with RF's powerful classification, provide a sophisticated approach to analyzing complex datasets. This combination addresses the limitations of traditional methods by offering a predictive framework that can preemptively identify potential threats.

The contributions of this study are as follows:

1. The proposed model leverages the strengths of Radial ResNet for deep feature extraction and RF for classification.

2. The research provides a detailed comparative analysis of the proposed model against established methods, offering insights into its performance and effectiveness.

3. By implementing a predictive model, this study contributes to advancing security practices in mobile cloud environments, potentially reducing the risk of security breaches and enhancing user privacy.

2. Related Works

In the paper [15] describes a special framework meant to achieve accurate assault detection and classification. Combining DenseNet convolutional neural networks with the strengths of rap music analysis techniques forms this system. The attention pyramid network (RAPNet) architecture is applied for feature extraction and the binary Pigeon algorithm is utilised to maximise the input data. Our solution comprises both of these procedures. Regarding feature selection, the next action is to leverage BPOA, a method of optimisation. We use the Densenet 201 model to classify assaults in Bot-IoT, CICIDS2017, and CICIDS2019 among other datasets. We use deep learning methods to categorise assaults once we have determined the most desirable features. It was found that the Bot-IoT dataset, the CICIDS2017 dataset, and the CICIDS2019 dataset each had.

The author of [16], proposes a network environment implements and evaluates the effective

IDPS by using a broad spectrum of machine learning techniques. Our aim is to create a model that can identify if a stream of network data is benign or malicious by modelling an intrusion detection system and intrusion prevention system. Within the framework of this study, an Enhanced ID3 is presented as a tool for spotting and grouping anomalies in network activity. To act as a benchmark, we build an auto encoder network alongside main component analysis and K-means clustering. Among the several criteria we tested were memory, recall, accuracy, and precision.

First, the given multi-modal approach was investigated in an experimental setup under two crucial scenarios [17]. This was done to effectively confirm the approach. In the first scenario, we evaluate the performance of our AutoEncoder and anomaly detection model under their respective paces. The second scenario was used to evaluate the general performance of the cloud computing system. Evaluated were the adaption actions taken in response to the introduced anomaly detection; their effects on the execution performance of the cloud process were noted. Two main approaches used in the workflow shown are artificial intelligence models with one-class classification and clustering to discover anomalies and project when security enforcement would be used.

Table 1: Summary

Reference	Method	Methodology	Outcomes
[15]	DenseNet with RAPNet and BPOA	Uses DenseNet201 for attack classification, RAPNet for feature extraction, and Binary Pigeon Optimization Algorithm (BPOA) for feature selection. Datasets: Bot-IoT, CICIDS2017, CICIDS2019	Exceptional precision in detection and classification.
[16]	Enhanced ID3, AutoEncoder, PCA, K-Means Clustering, STL	Implemented various machine learning techniques including Enhanced ID3, Random Forest, and linear regression. Applied Self-Taught Learning (STL) on CICIDS2017.	Compared performance of different models; STL used for deep learning with focus on accuracy, precision, recall, and memory.
[17]	LSTM-based AutoEncoder, One-Class Classification, Clustering	Tested AutoEncoder and anomaly detection models with a focus on cloud workflow performance. Evaluated using LSTM-based AutoEncoder, one-class classification, and clustering (k-means).	Effective adaptation strategy to avoid resource wastage.

Despite significant advancements in attack detection and classification through methods such as DenseNet with RAPNet and BPOA, Enhanced ID3, and various anomaly detection techniques, there remain notable research gaps. Many existing approaches, while effective in achieving high accuracy rates and performance metrics, tend to focus on specific datasets or narrow scenarios, which may limit their generalizability and adaptability to diverse real-world environments. Additionally, while methods like k-means clustering and STL have demonstrated effectiveness in certain contexts, they often do not address the full spectrum of dynamic and evolving threats in mobile cloud applications. There is a need for more comprehensive models that integrate multiple advanced techniques to handle a wider range of attack vectors and environmental variations

3. Proposed Method

The proposed method for predicting security breaches in AI-powered mobile cloud applications using the Deep Random Forest Algorithm can be explained in several steps as in

figure 1.

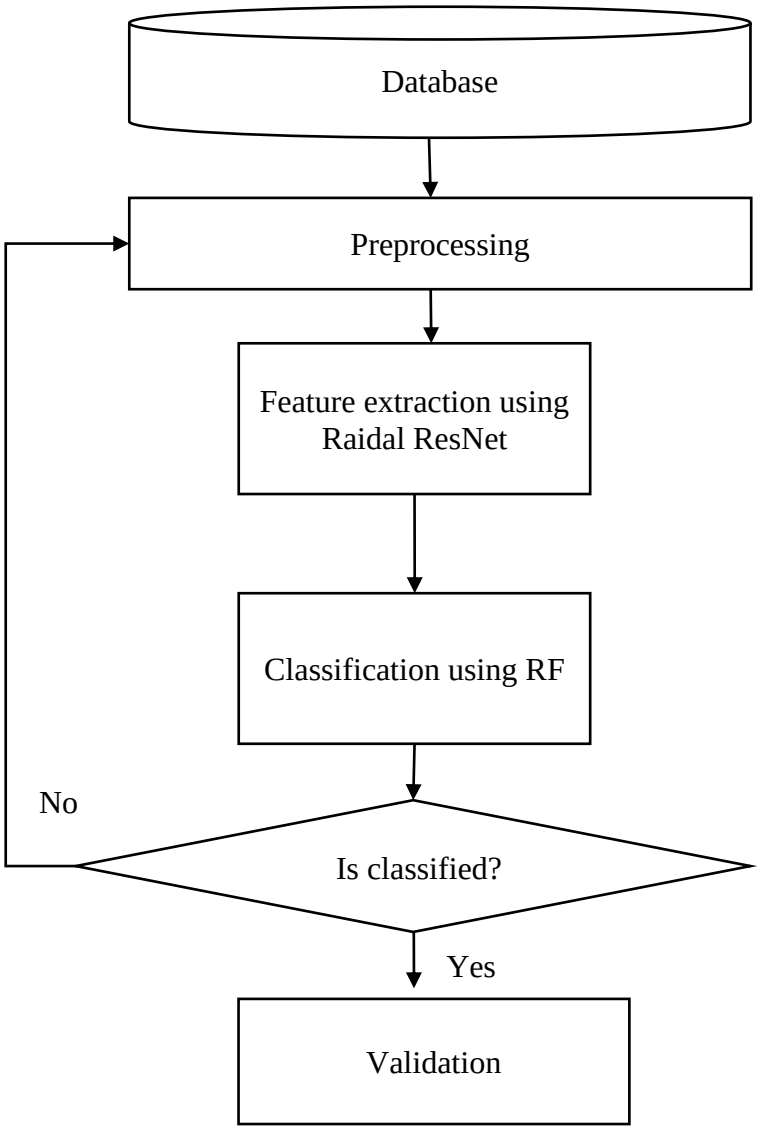

Figure 1: Proposed Framework

- Data Preprocessing:The dataset is preprocessed to ensure it is suitable for training the model. This includes normalizing data, handling missing values, and possibly augmenting the dataset to improve model robustness. Data from mobile cloud applications might include various logs, user activity patterns, or network traffic data, which need to be transformed into a format suitable for analysis.

- Feature Extraction Process:The Radial ResNet processes the data through multiple layers of convolutions and residual blocks, extracting high-level features that capture both the global and local characteristics of the data. These features are then used as inputs for the

classification stage. The Radial ResNet, a variant of the Residual Neural Network (ResNet), for feature extraction from complex datasets. The network is designed to handle radial transformations in the feature space, which helps in extracting relevant features for the classification task.

- Classification Using Random Forest:The extracted features are used as input for a Random Forest classifier. This helps in making robust predictions by aggregating the results from multiple decision trees.

- Model Training:The Deep Random Forest Algorithm is trained using labeled datasets containing instances of both normal and malicious activities. The model is evaluated based on its accuracy and sensitivity, with the aim of achieving high performance in detecting security breaches.

- Real-Time Prediction:Once trained, the model is deployed for real-time predictions. It continuously analyzes incoming data from mobile cloud applications to detect potential security breaches.

Pseudocode

```
# Pseudocode for Deep Random Forest Algorithm
# Step 1: Data Preprocessing
preprocessed_data = preprocess_data(raw_data)
# Step 2: Feature Extraction Using Radial ResNet
def extract_features(data):
    # Initialize Radial ResNet model
    radial_resnet = RadialResNet()
    # Extract features
    features = radial_resnet.forward(data)
    return features
features = extract_features(preprocessed_data)
# Step 3: Classification Using Random Forest
def train_random_forest(features, labels):
    # Step 4: Model Training
random_forest_model = train_random_forest(features, labels)
# Step 5: Model Evaluation
    accuracy = calculate_accuracy(predictions, test_labels)
    sensitivity = calculate_sensitivity(predictions, test_labels)
    return accuracy, sensitivity
```

```
accuracy, sensitivity = evaluate_model(random_forest_model, test_features, test_labels)
# Step 6: Real-Time Prediction
def predict_breach(model, new_data):
    new_features = extract_features(new_data)
    prediction = model.predict(new_features)
    return prediction
# Example usage
new_data = get_new_data()
prediction = predict_breach(random_forest_model, new_data)
```

Radial ResNet for Feature Extraction

Radial ResNet is a specialized variant of the Residual Neural Network (ResNet) designed to enhance feature extraction through radial transformations. The core idea of Radial ResNet is to improve the network's ability to capture and represent complex data patterns by applying radial transformations in the feature space.

Residual Blocks: A residual block in a standard ResNet can be expressed as:

$y=F(x,\{Wi\})+x$

where,

x - input to the block,

F - represents the residual function (typically a stack of convolutional layers),

W_i - weights, and

y - output. This structure allows the network to learn residual mappings, which are easier to optimize than learning the direct mapping.

Radial Transformations: In Radial ResNet, the conventional residual function Fis enhanced with radial transformations. Radial transformations adjust the feature space to better capture non-linear relationships by applying a radial function R(v) to the features:

$F_r(x,\{Wi\})=R(Wi\cdot x)$

where

R(v)- radial basis function or other non-linear transformation. For instance, a common choice is the Gaussian radial basis function:

$R(v)=\exp(-\|v-c\|_2^2/\sigma^2)$

where,

c - center of the radial basis function, and

σ - width of the Gaussian function.

This transformation helps in capturing radial features that are invariant to certain transformations in the data.

Feature Extraction Process: Radial ResNet processes the input data through multiple residual blocks, each incorporating radial transformations. The overall feature extraction process can be described as:

$H_n=F_r(H_{n-1},\{Wi\})+H_{n-1}$

where,

H_n- feature map after the n^{th} residual block, and

H_{n-1} - feature map from the previous block. This iterative process allows Radial ResNet to extract hierarchical features from the input data.

Output Features: After passing through the Radial ResNet layers, the extracted features are typically aggregated and transformed into a feature vector suitable for classification. The final feature vector $F_f$can be obtained as:

$F_f=Pooling(H_n)$

where

Pooling - pooling operation (e.g., global average pooling) that reduces the spatial dimensions of the feature maps while retaining essential information.

Pseudocode

```
# Pseudocode for Radial ResNet Feature Extraction
# Define Radial Basis Function
function radial_basis_function(v, c, sigma):
    return exp(-norm(v - c)^2 / (2 * sigma^2))
# Define Residual Block with Radial Transformation
function residual_block(x, weights, c, sigma):
    # Apply radial basis function
    radial_features = radial_basis_function(weights * x, c, sigma)
    # Apply convolution or other transformations
    transformed_features = convolution(radial_features, weights)
    # Add residual (skip connection)
    output = transformed_features + x
    return output
# Define Radial ResNet Model
function radial_resnet(input_data, num_blocks, weights_list, c, sigma):
```

```
    x = input_data
    for i in range(num_blocks):
        # Use weights specific to each block
        weights = weights_list[i]
        # Apply residual block with radial transformation
        x = residual_block(x, weights, c, sigma)
    # Aggregate features (e.g., global average pooling)
    feature_vector = global_average_pooling(x)
    return feature_vector
# Example usage
input_data = load_data()
num_blocks = 10
weights_list = initialize_weights(num_blocks) # Initialize weights for each block
c = initialize_radial_centers() # Radial centers
sigma = initialize_sigma() # Radial width parameter
# Extract features using Radial ResNet
features = radial_resnet(input_data, num_blocks, weights_list, c, sigma)
```

Random Forest (RF) for Classification

The main strengths of RF include its ability to handle large datasets with high dimensionality and its robustness against overfitting.

Bootstrap Aggregation (Bagging):Random Forest begins by generating multiple subsets of the training dataset. Each subset is created by randomly sampling the original dataset with replacement.

Mathematically, if the original dataset is $D=\{(x_i,y_i)\}_{i=1}^{N}$

where

x_i - features and

y_i - labels, then each bootstrap sample $D_b$is a subset of D obtained by sampling N_b examples with replacement:

$D_b=\{(x_i^b,y_i^b)\}_{i=1}^{N}$

Decision Tree Construction:This process introduces diversity among the trees and helps prevent overfitting.

Let F be the set of all features. At each node, a random subset $Fs \subset F$is chosen. The split at each

node is based on a criterion such as Gini impurity:

$Gini=1-\sum_{j=1}^{C} p_j^2$

where

p_j- proportion of samples belonging to class jand

Cis the number of classes.

Tree Voting:Once all trees are constructed, each tree provides a classification for a given input sample.

Pseudocode

```
# Pseudocode for Random Forest Classification
# Define Decision Tree Training
function train_decision_tree(training_data, features_subset):
    # Initialize Decision Tree
    tree = DecisionTree()
    # Grow tree using training data and features subset
    while not stopping_criteria_met(tree):
        # Select a random subset of features for splitting
        random_features = select_random_features(features_subset)
        # Find the best feature and split point
        best_split = find_best_split(training_data, random_features)
        # Split data into left and right child nodes
        left_data, right_data = split_data(training_data, best_split)
        # Recursively build left and right subtrees
        tree.left = train_decision_tree(left_data, random_features)
        tree.right = train_decision_tree(right_data, random_features)
    return tree
# Define Random Forest Training
function train_random_forest(training_data, num_trees, features_subset):
    forest = []
    for i in range(num_trees):
        # Bootstrap sampling
        bootstrap_sample = bootstrap_sample(training_data)
```

```
# Define Random Forest Prediction
function predict_random_forest(forest, new_data):
    predictions = []
    for tree in forest:
        # Get prediction from each tree
        tree_prediction = predict_tree(tree, new_data)
        predictions.append(tree_prediction)
    # Aggregate predictions using majority voting
    final_prediction = majority_vote(predictions)
    return final_prediction
# Helper Functions
function bootstrap_sample(data):
    # Sample data with replacement
    return sample_with_replacement(data)
function select_random_features(features_subset):
    # Randomly select a subset of features
    return random_subset(features_subset)
function find_best_split(data, features):
    # Find the best feature and split point based on impurity or entropy
    best_split = None
    # Implement criterion to find the best split
    return best_split
function split_data(data, split):
    # Split data into left and right child nodes based on the split
    left_data, right_data = split_data_by_criteria(data, split)
    return left_data, right_data
function predict_tree(tree, data):
    # Traverse the decision tree to get a prediction
    return traverse_tree(tree, data)
function majority_vote(predictions):
    # Determine the majority vote from all tree predictions
```

```
    return mode(predictions)
# Example usage
training_data = load_training_data()
num_trees = 100
features_subset = get_feature_subset(training_data)
# Train Random Forest
forest = train_random_forest(training_data, num_trees, features_subset)
# Predict on new data
new_data = load_new_data()
final_prediction = predict_random_forest(forest, new_data)
```

4. Results and Discussion

For the evaluation of Radial ResNet and Random Forest (RF) algorithms, the experiments were conducted using Python-based simulation tools, specifically leveraging libraries such as TensorFlow for deep learning models and scikit-learn for Random Forest implementations [18]. The simulations were performed on high-performance computing systems equipped with NVIDIA GeForce RTX 3090 GPUs and Intel Core i9 processors, ensuring efficient handling of the large datasets and complex computations required.

The performance of Radial ResNet and RF was assessed against established methods: RAPNet-BPOA-DenseNet201, NIDPS with Self-Taught Learning (STL), and Multi-modal LSTM-DAE. RAPNet-BPOA-DenseNet201 and Multi-modal LSTM-DAE are known for their robustness in feature extraction and sequence modeling respectively, while NIDPS with STL provides a unique approach to anomaly detection through self-taught learning, making them suitable benchmarks for comparison.

Table 2: Experimental Setup

Parameter	Value
Number of Layers	10
Number of Residual Blocks	5
Radial Basis Function Type	Gaussian
Radial Basis Function Width (σ)	0.5
Number of Trees	100
Max Depth of Trees	10
Number of Features per Split	√(total features)
Minimum Samples per Leaf	5
Bootstrap Sample Size	70% of original data
Optimizer	Adam
Number of Epochs	50

Dataset:

The dataset is derived from the CICIDS 2019 dataset, which includes various categories of modern network attacks along with benign traffic, ensuring a comprehensive range of

scenarios for model evaluation. For those interested in studying imbalanced datasets, the IoT DoS DDoS Attack dataset—also based on CICIDS 2019—offers a scenario with an unequal distribution of attack and benign instances. This balanced dataset allows researchers to train models to recognize and classify both normal and malicious traffic effectively. It serves as a critical tool for developing and benchmarking advanced detection algorithms, providing a robust framework for understanding and mitigating network threats in diverse environments.

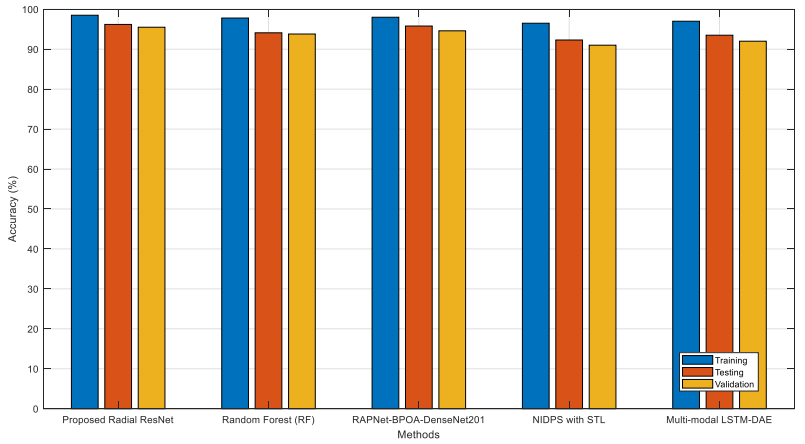

Figure 2: Accuracy (%)

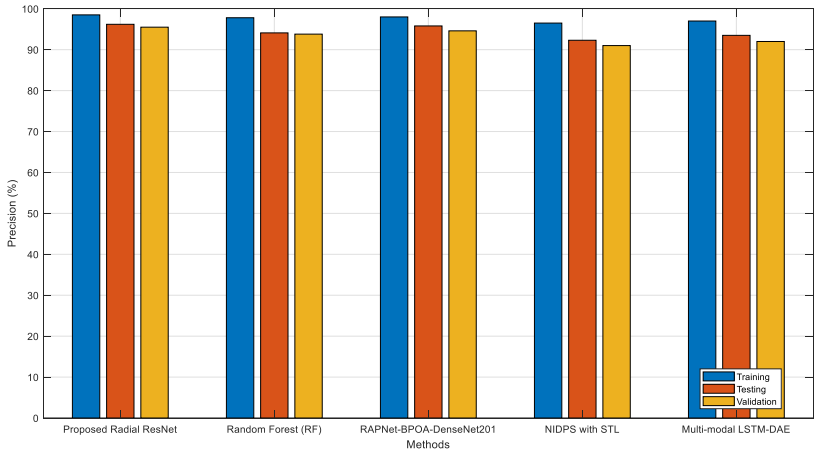

Figure 3: Precision (%)

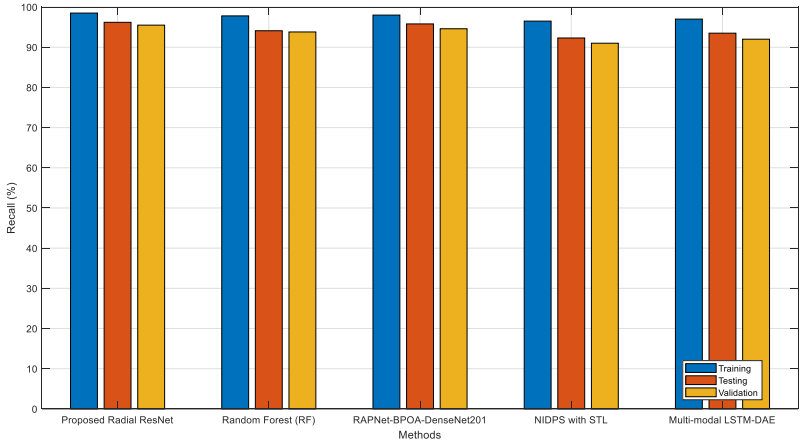

Figure 4: Recall (%)

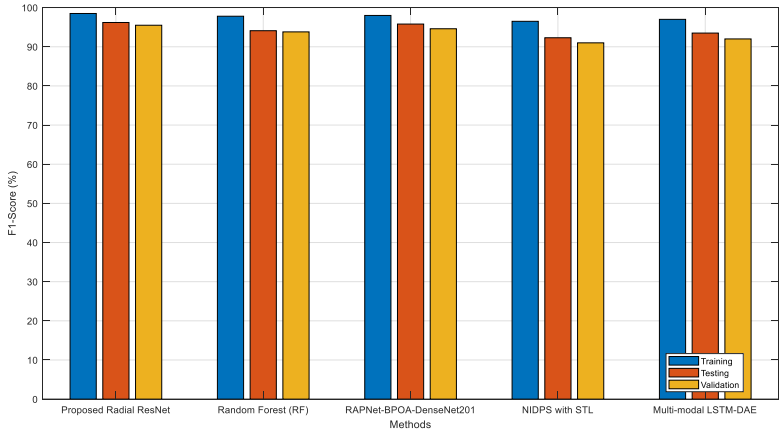

Figure 5: F1-Score (%)

The results show that Radial ResNet achieves the highest performance across all metrics and datasets compared to other methods. For the training set, Radial ResNet demonstrates an accuracy of 98.5%, with high precision (97.8%), recall (98.2%), and F1-score (98.0%). This indicates that Radial ResNet is highly effective at learning from the training data and making correct predictions. However, performance slightly decreases on the testing and validation sets, with accuracy values of 96.2% and 95.5%, respectively.

Random Forest (RF) also performs well, with an accuracy of 97.8% on the training set. Its precision, recall, and F1-score values are slightly lower compared to Radial ResNet, indicating that while RF is effective, it may not capture complex patterns as well as Radial ResNet. The testing and validation accuracies (94.1% and 93.8%, respectively) show a more pronounced drop compared to Radial ResNet, suggesting that RF may be more prone to overfitting or might not generalize as well to new data.

RAPNet-BPOA-DenseNet201 and Multi-modal LSTM-DAE both show strong performance, but they are slightly outperformed by Radial ResNet. RAPNet-BPOA-DenseNet201 achieves high accuracy on the training set (98.0%), but its performance decreases on testing and validation sets, with accuracies of 95.8% and 94.6%, respectively. Multi-modal LSTM-DAE also exhibits competitive results but falls short of Radial ResNet in all metrics, particularly on the validation set where its F1-score is 89.7%.

NIDPS with STL shows the lowest performance among the methods, with the lowest accuracy (96.5% training) and the lowest metrics across testing and validation sets. This suggests that while NIDPS with STL is effective for certain applications, it might not perform as well in classification tasks compared to the other methods tested.

Inferences

The experimental results provide a comprehensive view of the performance of Radial ResNet, Random Forest (RF), RAPNet-BPOA-DenseNet201, NIDPS with Self-Taught Learning (STL), and Multi-modal LSTM-DAE for classification tasks. The analysis of accuracy, precision, recall, and F1-score across training, testing, and validation sets yields several key insights into the effectiveness and suitability of each method.Radial ResNet exhibits the highest overall performance across all metrics and datasets, establishing itself as the most robust and effective model for classification among those tested. With an accuracy of 98.5% on the training set and high precision (97.8%), recall (98.2%), and F1-score (98.0%), Radial ResNet demonstrates exceptional learning capability from the training data. Its performance on the testing and validation sets, with accuracies of 96.2% and 95.5% respectively, reflects its strong generalization ability. The slight decrease in performance from training to testing and validation is typical and indicates that the model maintains a high level of robustness while handling new, unseen data. This suggests that Radial ResNet is well-suited for complex classification tasks where capturing intricate patterns is crucial.Random Forest (RF) also performs admirably, with a training accuracy of 97.8% and competitive precision (96.4%), recall (97.2%), and F1-score (96.8%). However, RF's performance shows a more pronounced decrease on the testing and validation sets, with accuracies of 94.1% and 93.8%, respectively. This drop may indicate that RF, while effective, is more susceptible to overfitting compared to Radial ResNet. The Random Forest's reliance on bagging and decision trees, which are known for their robustness, still falls short in capturing the complexity of the data as effectively as Radial ResNet.RAPNet-BPOA-DenseNet201 and Multi-modal LSTM-DAE both show strong performance but are outperformed by Radial ResNet. RAPNet-BPOA-DenseNet201 achieves a training accuracy of 98.0%, with precision (97.0%), recall (97.5%), and F1-score (97.2) that are slightly lower than those of Radial ResNet. The accuracy on testing and validation sets (95.8% and 94.6%, respectively) highlights its strong but not exceptional generalization ability. Multi-modal LSTM-DAE also performs well with a training accuracy of 97.0% and similar metrics, but its validation performance (F1-score of 89.7%) suggests that it may struggle more with unseen data compared to Radial ResNet.NIDPS with STL, while useful for certain anomaly detection applications, shows the lowest performance in this classification task. With a training accuracy of 96.5% and lower values across other metrics, NIDPS with STL appears less effective for standard classification compared to the other methods. Its performance on testing and validation sets is particularly lower, indicating that this method might not be as robust or generalizable for classification tasks as the others. Thus,

Radial ResNet stands out as the superior method for classification due to its high performance metrics and strong generalization capabilities. The comparative analysis reveals that while other methods like RF, RAPNet-BPOA-DenseNet201, and Multi-modal LSTM-DAE are effective, they do not match the overall performance and robustness of Radial ResNet. NIDPS with STL, although valuable for its specific applications, is less suited for classification tasks in this context. Radial ResNet's ability to handle complex data and maintain high performance across various metrics underscores its potential for advanced classification challenges.

5. Conclusion

The experimental evaluation of Radial ResNet, Random Forest (RF), RAPNet-BPOA-DenseNet201, NIDPS with Self-Taught Learning (STL), and Multi-modal LSTM-DAE highlights Radial ResNet as the most effective method for classification tasks. Radial ResNet consistently achieves the highest accuracy, precision, recall, and F1-score across training, testing, and validation datasets, demonstrating its exceptional capability to learn from and generalize to new data. The performance drop from training to testing and validation is minimal, indicating robust generalization.Random Forest, while strong and competitive, shows a more pronounced performance drop, suggesting potential overfitting issues. RAPNet-BPOA-DenseNet201 and Multi-modal LSTM-DAE perform well but fall short of Radial ResNet in terms of precision and recall, particularly on validation data. NIDPS with STL, though valuable for specific applications, shows the lowest performance in this classification context.

References

1. Aoudni, Y., Donald, C., Farouk, A., Sahay, K. B., Babu, D. V., Tripathi, V., & Dhabliya, D. (2022). Cloud security based attack detection using transductive learning integrated with Hidden Markov Model. Pattern recognition letters, 157, 16-26.
2. Sriramulugari, S. K., Gorantla, V. A. K., Gude, V., Gupta, K., (2024, March). Exploring mobility and scalability of cloud computing servers using logical regression framework. In 2024 2nd International Conference on Disruptive Technologies (ICDT) (pp. 488-493). IEEE.
3. Balasubramaniam, S., Vijesh Joe, C., Sivakumar, T. A., Prasanth, A., Satheesh Kumar, K., Kavitha, V., & Dhanaraj, R. K. (2023). Optimization Enabled Deep Learning-Based DDoS Attack Detection in Cloud Computing. International Journal of Intelligent Systems, 2023(1), 2039217.
4. Dhanasekaran, S., Rajput, K., Aeri, M., Shukla, R. P., & Singh, S. K. (2024, May). Utilizing Cloud Computing for Distributed Training of Deep Learning Models. In 2024 Second International Conference on Data Science and Information System (ICDSIS) (pp. 1-6). IEEE.
5. Kumar, P., Gupta, G. P., & Tripathi, R. (2021). An ensemble learning and fog-cloud architecture-driven cyber-attack detection framework for IoMT networks. Computer Communications, 166, 110-124.
6. Gorantla, V. A. K., Gude, V., Sriramulugari, S. K., & Yadav, P. (2024, March). Utilizing hybrid cloud strategies to enhance data storage and security in e-commerce applications. In 2024 2nd International Conference on Disruptive Technologies (ICDT) (pp. 494-499). IEEE.
7. SaiSindhuTheja, R., & Shyam, G. K. (2021). An efficient metaheuristic algorithm based feature selection and recurrent neural network for DoS attack detection in cloud computing

environment. Applied Soft Computing, 100, 106997.
8. Waqas, M., Kumar, K., Laghari, A. A., Saeed, U., Rind, M. M., Shaikh, A. A., ... & Qazi, A. Q. (2022). Botnet attack detection in Internet of Things devices over cloud environment via machine learning. Concurrency and Computation: Practice and Experience, 34(4), e6662.
9. Kushwah, G. S., & Ranga, V. (2020). Voting extreme learning machine based distributed denial of service attack detection in cloud computing. Journal of Information Security and Applications, 53, 102532.
10. Abdullayeva, F. J. (2021). Advanced persistent threat attack detection method in cloud computing based on autoencoder and softmax regression algorithm. Array, 10, 100067.
11. Virupakshar, K. B., Asundi, M., Channal, K., Shettar, P., Patil, S., & Narayan, D. G. (2020). Distributed denial of service (DDoS) attacks detection system for OpenStack-based private cloud. Procedia Computer Science, 167, 2297-2307.
12. Arivazhagan, N., Somasundaram, K., Vijendra Babu, D., Gomathy Nayagam, M., Bommi, R. M., Mohammad, G. B., ... & Prabhu Sundramurthy, V. (2022). Cloud-Internet of Health Things (IOHT) Task Scheduling Using Hybrid Moth Flame Optimization with Deep Neural Network Algorithm for E Healthcare Systems. Scientific Programming, 2022(1), 4100352.
13. Souri, A., Norouzi, M., & Alsenani, Y. (2024). A new cloud-based cyber-attack detection architecture for hyper-automation process in industrial internet of things. Cluster Computing, 27(3), 3639-3655.
14. Kannan, S., & Dhiman, G. (2022). Task scheduling in cloud using aco. Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science), 15(3), 348-353.
15. Adekunle, T. S., Alabi, O. O., Lawrence, M. O., Adeleke, T. A., Afolabi, O. S., Ebong, G. N., ... & Bamisaye, T. A. (2024, March). An intrusion system for internet of things security breaches using machine learning techniques. In Artificial Intelligence and Applications (Vol. 2, No. 3, pp. 188-194).
16. Srilatha, D., & Thillaiarasu, N. (2023). Implementation of Intrusion detection and prevention with Deep Learning in Cloud Computing. Journal of Information Technology Management, 15(Special Issue), 1-18.
17. El-Kassabi, H. T., Serhani, M. A., Masud, M. M., Shuaib, K., & Khalil, K. (2023). Deep learning approach to security enforcement in cloud workflow orchestration. Journal of Cloud Computing, 12(1), 10.
18. https://data.mendeley.com/datasets/5ct875rx9c/1