COS711 Assignment 2

Jaimen Govender u20464348

3 October 2024

1 Introduction

This project focuses on classifying almonds into three varieties: MAMRA, REGULAR, and SANORA, using neural networks. I utilize two optimization algorithms, ADAM and RProp, while emphasizing data preprocessing and hyperparameter tuning. The main objective is to develop an effective neural network model for almond classification and to evaluate the performance of different training optimizers and hybrid learning approaches that combine ADAM and RProp models.

1.1 Background

Neural networks are computational models designed to distinguish complex patterns within a dataset, making them good for classification tasks. In this project, I utilize two optimization algorithms:

1. ADAM (Adaptive Moment Estimation) is an optimization algorithm that dynamically adjusts the learning rates for each parameter based on the first and second moments of the gradients. ADAM can adapt to the changing landscape of the loss function during training, allowing for faster convergence and better performance. Adam incorporates a form of momentum first moment of the gradients. [1]

2. RProp (Resilient Propagation) focuses on adjusting the weights of the model based solely on the sign of the gradient, rather than its magnitude. This approach allows RProp to make updates that are insensitive to the scale of the gradient, effectively mitigating the effects of noisy gradients and improving convergence speed. [2]

3. As well as a hybrid approach that integrates both ADAM and RProp to enhance the model's performance. [3]

2 Data Preparation

2.1 Handling Missing Values

Every row in the dataset had at least one missing value, making it important to address the missing values. I used **mean imputation** to handle missing values because it preserves the central tendency of the feature without distorting the distribution. Since the aim is to maintain data consistency across features, this method avoids introducing bias toward extreme values. Mean imputation is simple and effective, ensuring that no information is lost while keeping the feature space complete, and maintaining the integrity of the dataset.

2.2 Encoding Outcome (Target in Dataset)

Since the target variable Type (almond type) is categorical, I used **Label Encoding** to convert the categorical values into numerical format. **The almond types MAMRA, REGULAR, and SANORA) were encoded as 0, 1, and 2**. Label encoding was chosen because there are only three different categories and neural networks cannot process categorical values.

2.3 Exploratory Data Analysis

2.3.1 Mean Feature Values for Each Almond Type

Table 1 and 2 shows the mean values of the features for each type of almond (MAMRA, REGULAR, and SANORA).

Table 1: Mean Values of Features by Almond Type (Part 1)

Type	Length (mm)	Width (mm)	Thickness (mm)	Area	Perimeter
MAMRA	321.81	171.56	100.80	27854.95	806.45
REGULAR	265.60	161.20	112.33	24271.05	688.49
SANORA	283.25	180.12	115.37	27383.59	736.37

Table 2: Mean Values of Features by Almond Type (Part 2)

Type	Roundness	Solidity	Aspect Ratio	Eccentricity
MAMRA	0.40	0.94	1.92	0.85
REGULAR	0.51	0.97	1.68	0.80
SANORA	0.50	0.96	1.63	0.79

2.3.2 Correlation Analysis

The correlation matrix in Figure 9 shows the relationship between features and almond types. This analysis helps in understanding which features are most strongly associated with each almond type.

The correlation matrix reveals key relationships between almond features and classification outcomes. Length (major axis) shows a positive correlation with Outcome_0 (0.41) and a negative correlation with Outcome_1 (-0.32), while Thickness (depth) is strongly negatively correlated with Outcome_0 (-0.39) and positively with Outcome_1 (0.13). Notably, Roundness has a strong negative correlation with Outcome_0 (-0.47), and both Aspect Ratio (0.78) and Eccentricity (0.77) are positively correlated with Outcome_0, indicating their importance in distinguishing this class. Features such as Aspect Ratio and Eccentricity are critical for classification.

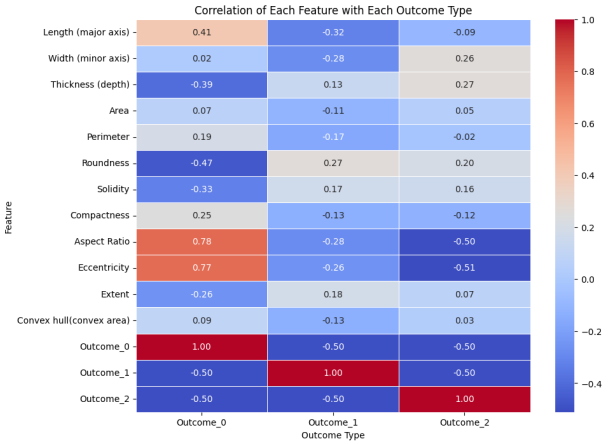

Figure 1: Correlation Matrix Between Features and Almond Types. Outcome_0=MAMRA, Outcome_1=REGULAR, Outcome_2=SANORA

2.3.3 Scatterplot Analysis

Figure 2 presents scatterplots of the almond features, color-coded by almond type, to visualize the relationships between the key variables for classification.

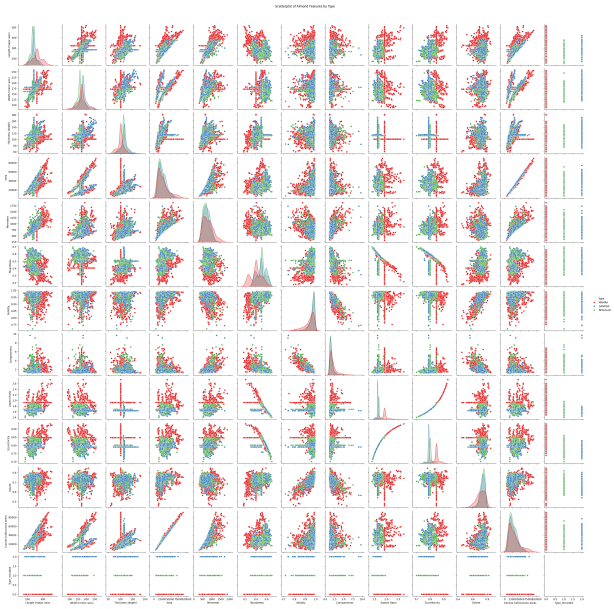

Figure 2: Scatterplot of Almond Features by Type

2.4 Feature Selection

The features were selected based on empirical analysis of the almond classification problem. After assessing correlations and reviewing key features related to almonds in Figure 1 and 2, certain features were discarded to focus on the most relevant aspects for classification:

- Selected Features: Aspect Ratio, Eccentricity, Roundness, Solidity, Length, Width, Thickness.
- Dropped Features: Area, Perimeter, Compactness, Extent, Convex Hull (Convex Area)

The discarded features did not provide substantial predictive power for almond classification. The retained features are used to distinguish different almond types.

2.5 Data Set Splitting

The dataset was split into training/validation (80% - 2242 records) and testing (20 % - 561 records) sets using a stratified train-test split, Stratification was necessary to maintain the proportion of almond types across the splits. This ensures that each almond type is adequately represented in both the training and testing sets. Additionally, 5-fold cross-validation was applied to assess the model's performance across different data partitions.

- Stratification ensures that each almond type (MAMRA, REGULAR, and SANORA) is proportionally represented in both the training/validation and testing sets. This is important when dealing with class imbalances, as it ensures that no class is over- or under-represented in either set.
- Cross-validation helps mitigate overfitting. Training and validating the model on different subsets of the data prevents the model from over-relying on a specific subset, ensuring that the model generalizes well to unseen data.
- K-fold cross-validation is good for hyperparameter optimization. Each fold offers an independent assessment of model performance, meaning that the model's parameters can be optimized based on how well they perform across multiple subsets of the data. Reducing the risk of overfitting.

2.6 Scaling Inputs/Output

1. ReLU (Rectified Linear Unit) activation function is used in the hidden layers. This helps to scale the inputs by outputting zero for any negative input, which allows the model to learn complex patterns.
2. For the output layer, the Softmax activation function is used, which transforms the raw output scores into probabilities for each almond type. The Softmax function ensures that the outputs are scaled between 0 and 1 and sum to 1, making it good for classification.

3 Model 1 (Adam) Architecture, Hyperparameters, and Training Algorithm

Model 1 uses the Adam optimizer for training the neural network. Adam helps improve convergence and stability by adjusting the learning rates for each parameter, making it effective for classifying almond types.

3.1 Neural Network Architecture

1. Input layer: 7 features (Aspect Ratio, Eccentricity, Roundness, Solidity, Length, Width, Thickness). From the feature selection above.
2. 5 hidden layers, each with 128 neurons and ReLU activation.
3. Output layer: 3 neurons (one for each almond type) with softmax activation.

3.2 Activation Functions

1. ReLU (Rectified Linear Unit) activation function is used in the hidden layers. This helps to scale

the inputs by outputting zero for any negative input, which allows the model to learn complex patterns.

2. For the output layer, the Softmax activation function is used, which transforms the raw output scores into probabilities for each almond type. The Softmax function ensures that the outputs are scaled between 0 and 1 and sum to 1, making it good for classification.

3.3 Weight Initialization

Weights are initialized using the Glorot (Xavier) initialization strategy, which balances variance across layers. This initialization helps mitigate issues like vanishing/exploding gradients

3.4 Error Function

The loss function used in this classification is Sparse Categorical Cross-Entropy. This function is ideal for multi-class classification where the target labels are provided as integers, such as the almond types in this dataset (0 for MAMRA, 1 for REGULAR, 2 for SANORA).

The Sparse Categorical Cross-Entropy loss is defined as:

$$L(y, \hat{y}) = -\log(\hat{y}_y)$$

where:

- y is the true class label (as an integer),
- $\hat{y}$ is the predicted probability for the true class, which is the output of the softmax activation function.

3.5 Hyperparamter Optimization

Hyperparameter tuning was performed using grid search with 5-fold cross-validation to find the optimal combination of batch size and learning rate.

3.5.1 Hyperparameters Considered

- Batch Size: I tested batch sizes of 16, 32, and 64. The larger the batch sizes the slower the training, while smaller batches can lead to faster training but noisier updates.

- Learning rates of 0.0001, 0.001, 0.01, and 0.1 were used in the grid search. The learning rate determines the size of the steps the optimizer takes during gradient descent. Smaller learning rates help in fine-tuning the model, while larger rates can speed up convergence but may overshoot.

3.5.2 Grid Search Setup

The grid search was configured to explore combinations of batch sizes and learning rates. A 5-fold cross-validation ensured that the model was evaluated on different subsets of the data to get reliable performance estimates.

3.5.3 Grid Search Results

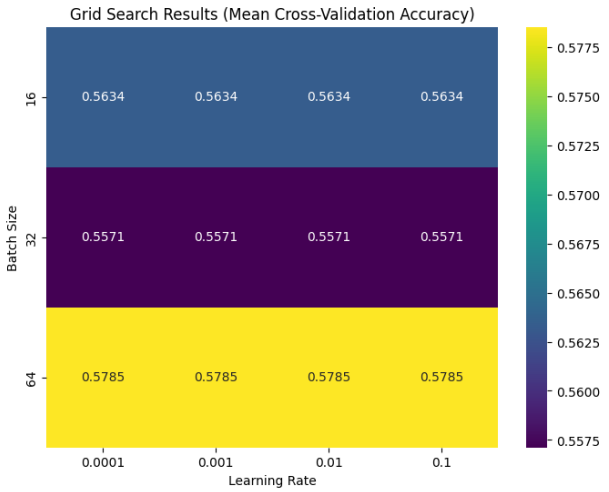

Figure 3: Grid Search Results (Mean Cross-Validation)

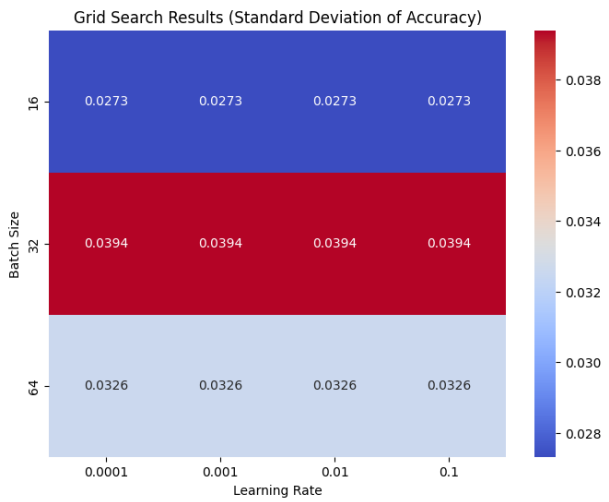

Figure 4: Grid Search Results (Standard Deviation Cross-Validation)

The grid search results show that the model's performance (mean cross-validation accuracy) remains consistent across different learning rates for each batch size. The best performance was achieved with a batch size of 64 and a learning rate of 0.0001, with a mean accuracy of 0.5785. The standard deviation of accuracy across folds shows slight variations, with batch size 32 having the highest variability, indicating less consistent performance compared to batch sizes 16 and 64.

3.6 Training Process

After finding the optimal hyperparameters, the model was trained for 250 epochs. Training with a moderate number of epochs ensures the model has enough time to learn without overfitting the training data.

3.7 Model Evaluation

The final neural network model's performance was evaluated using accuracy.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

4 Model 2 (Rprop) Architecture, Hyperparameters, and Training Algorithm

Model 2 employs the Rprop (Resilient Propagation) optimizer for training the neural network. Rprop focuses on adjusting the magnitude of weight updates based solely on the sign of the gradient, independent of its size.

4.1 Neural Network Architecture

Same as model 1.

4.2 Activation Functions

same as model 1.

4.3 Weight Initialization

The weights are initialized using PyTorch's default initialization method for fully connected layers. Specifically, PyTorch uses a uniform distribution, which is derived from the input size of the layer (Xavier initialization). This method helps ensure that the gradients to not explode, allowing the model to converge faster during training.

4.4 Loss Function

Cross Entropy Loss is selected due to its suitability for classification tasks where the goal is to predict one class from multiple options. Since the task involves distinguishing between different almond types, Cross Entropy provides a measure of how well the model's predicted probability distribution aligns with the true class labels.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c})$$

Where:

- N is the number of samples,
- C is the number of classes,

- $y_{i,c}$ is a binary indicator (0 or 1) if class label c is the correct classification for sample i,
- $p_{i,c}$ is the predicted probability that sample i belongs to class c.

4.5 Hyperparamter Optimization

Hyperparameter tuning was performed using grid search with 5-fold cross-validation to find the optimal combination of batch size and learning rate.

4.5.1 Hyperparameters Considered

- Batch Size: I tested batch sizes of 16, 32, and 64. The larger the batch sizes the slower the training, while smaller batches can lead to faster training but noisier updates.
- Learning rates of 0.001, 0.01, and 0.02 were used in the grid search. The learning rate determines the size of the steps the optimizer takes during gradient descent. Smaller learning rates help in fine-tuning the model, while larger rates can speed up convergence but may overshoot.

4.5.2 Grid Search Setup

Same as model 1.

4.5.3 Grid Search Results

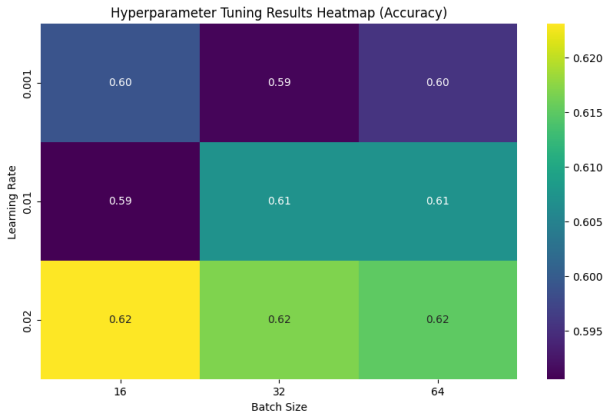

Figure 5: Grid Search Results (Mean Cross-Validation Accuracy)

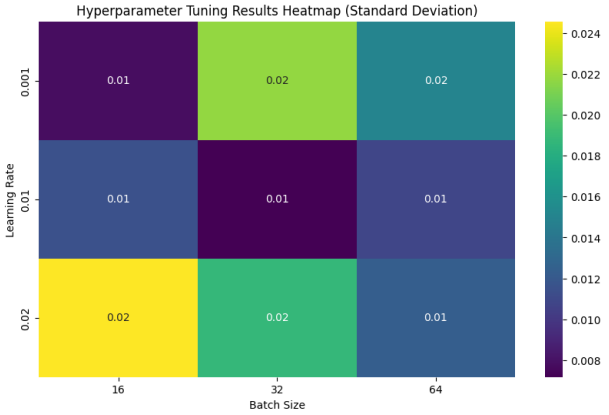

Figure 6: Grid Search Results (Standard Deviation Cross-Validation Accuracy)

The results show the performance of different hyperparameter combinations for the neural network mode. The accuracy matrix indicates that the best-performing model was achieved with a learning rate of 0.02 and a batch size of 16, yielding an average accuracy of approximately 0.623. The standard deviation matrix shows relatively low variability across different runs, especially for this combination, suggesting that the model's performance is stable and reliable with these hyperparameters.

4.6 Training Process

After finding the optimal hyperparameters, the model was trained for 250 epochs. Training with a moderate number of epochs ensures the model has enough time to learn without overfitting the training data.

4.7 Model Evaluation

The final neural network model's performance was evaluated using accuracy.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

5 Model 3: Hybrid Learning (Adam + Rprop)

5.1 Neural Network Architecture

Same as models 1 and 2.

5.2 Activation Functions

Same as models 1 and 2.

5.3 Weight Initialization

The Hybrid approach merges the contributions of two separate models trained with different optimizers (Adam and Rprop) into a unified model. Once both Adam and Rprop have been trained, the weights from both models are combined into a single model using a weighted average.

In this section, we assess the similarity between the weights of the models trained using Adam and Rprop before combining them. We use cosine similarity to do this, which tells us how similar two sets of weights are. Cosine similarity measures the angle between two weight vectors. If the angle is small, the weights are more similar, and the cosine similarity will be close to 1. If the weights are very different, the cosine similarity will be closer to 0.

For the hybrid model, I compared the weights from Adam and Rprop. I calculate the cosine similarity between the weights from both models. This function reshapes and aligns the weight matrices from the two models and computes how similar each pair of corresponding weights are. The closer the similarity values are to 1, the more aligned the models' weights are, indicating that the two optimizers are converging towards similar solutions.

After assessing the weight similarities, the next step is to combine the weights from both models into a single unified model. I use the weighted average of the weights based on each model's accuracy. This is done by giving more weight to the better-performing model. For instance, if the Adam model has higher accuracy than the Rprop model, the Adam weights will have a larger influence on the final model.

Total Performance Calculation: The combined performance is calculated as the sum of the accuracies of both models: M1 is Adam and M2 is Rprop

$$\text{Total Performance} = \text{M1 Accuracy} + \text{M2 Accuracy}$$

Weight Contribution Calculation: The contribution of each model is determined by dividing its accuracy by the total performance. For instance, if Adam achieves 80% accuracy and Rprop achieves 70%, their respective contributions are:

$$\text{Adam Contribution} = \frac{80}{80+70} = 0.533 \quad (53.3\%)$$

$$\text{Rprop Contribution} = \frac{70}{80+70} = 0.467 \quad (46.7\%)$$

Weighted Averaging of Weights: For each weight in the neural network, the final weight is computed as a weighted average of the corresponding Adam and Rprop weights:

$$\text{Final Weight} = (\text{A} \times \text{B}) + (\text{C} \times \text{D})$$

- A = Adam Weight
- B = Adam Contribution
- C = Rprop Weight
- D = Rprop Contribution

This ensures that the final model incorporates the strengths of both optimizers, with a higher emphasis on the model with better performance.

5.4 Error Function

Same as model 2.

5.5 Hyperparamter Optimization

Hyperparameter tuning was performed using grid search with 5-fold cross-validation to find the optimal combination of batch size and learning rate.

5.5.1 Hyperparameters Considered

- Batch Size: I tested batch sizes of 16, 32, and 64. The larger the batch sizes the slower the training, while smaller batches can lead to faster training but noisier updates.

- Learning rates of 0.0001 is used in the grid search.

5.5.2 Grid Search Setup

The grid search was configured to explore combinations of batch sizes and learning rates. A 5-fold cross-validation ensured that the model was evaluated on different subsets of the data to get reliable performance estimates.

5.5.3 Grid Search Results

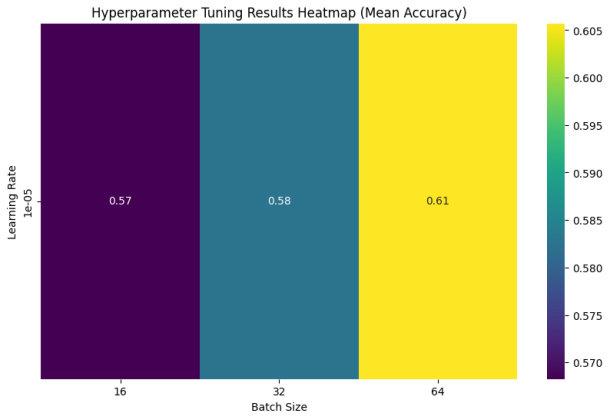

Figure 7: Grid Search Results (Mean Cross-Validation)

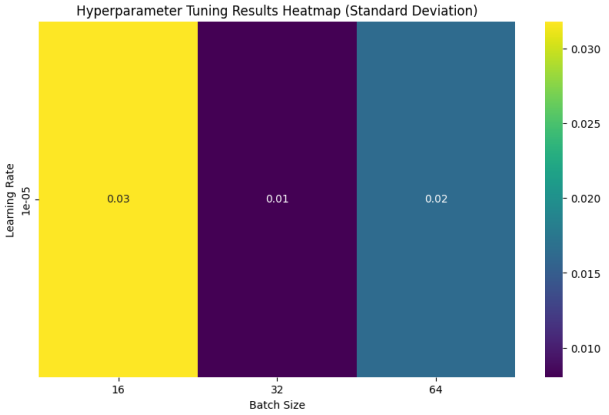

Figure 8: Grid Search Results (Standard Deviation Cross-Validation)

The results from the grid search indicate that a learning rate of 0.00001 yielded the highest average accuracy across different batch sizes. Specifically, the best performance was achieved with a batch size of 64, resulting in an accuracy of approximately 60.57%. The standard deviation values suggest that the model's performance was relatively stable, with the lowest variability observed at this batch size.

5.6 Training Process

After finding the optimal hyperparameters, the model was trained for 200 epochs. Training with a moderate number of epochs ensures the model has enough time to learn without overfitting the training data.

5.7 Model Evaluation

Same as models 1 and 2.

6 Experimental Setup and Results

6.1 Experimental Setup

To ensure the reproducibility of results, all experiments were run using a fixed range of random seeds. Seeds 40-49 were used for all models to maintain

consistency across the training and evaluation processes. Meaning that 10 individual runs were performed. This allows for accurate replication and comparison of results across different runs and ensures that any variations are observed.

You can access the Jupyter Notebook by clicking on the link below:

Jupyter Notebook

6.1.1 Hyperparameters

The table below presents the hyperparameters used for the Adam, RProp, and Hybrid models in the experiment:

Table 3: Hyperparameters for Adam, RProp, and Hybrid Models

Hyperparameter	Adam	RProp	Hybrid
Input Neurons	7	7	7
Number of Hidden Neurons	128	128	128
Number of Output Neurons	3	3	3
K-Folds	5	5	5
Batch Size	32	16	64
Learning Rate	0.0001	0.2	0.00001
Training Epochs	250	250	200

Table 4: Accuracy Results for Different Optimizers

Seed	**Adam (%)**	**Rprop (%)**	**Hybrid (%)**
40	87.52	64.17	77.98
41	90.55	59.82	83.78
42	90.55	60.27	81.88
43	85.03	61.83	81.49
44	90.91	59.15	81.05
45	85.03	62.05	79.88
46	83.60	59.82	79.60
47	91.44	60.49	79.38
48	89.66	58.71	78.21
49	82.53	62.72	85.90
Avg	**88.20%**	**60%**	**80.07%**

6.2 Results

The evaluated performance of three different optimizers—Adam, Rprop, and a Hybrid approach—using various random seeds (40-49). The following table summarizes the accuracy results for each optimizer, as well as the average accuracy across the different seeds.

Interpretation of Results

1. Adam Optimizer: consistently achieved the highest accuracy across all seeds, with an average accuracy of 88.20%.

2. Rprop Optimizer: The Rprop optimizer showed a significantly lower average accuracy of 60.00% compared to Adam. The Rprop method, which is designed for minimizing only the loss function without considering the magnitude of gradients, seems less suited for this dataset.

3. Hybrid optimizer: demonstrated a notable performance, with an average accuracy of 80.07%. Although it did not perform as Adam, it outperformed Rprop, indicating that the Hybrid method may leverage advantages from both optimizers.

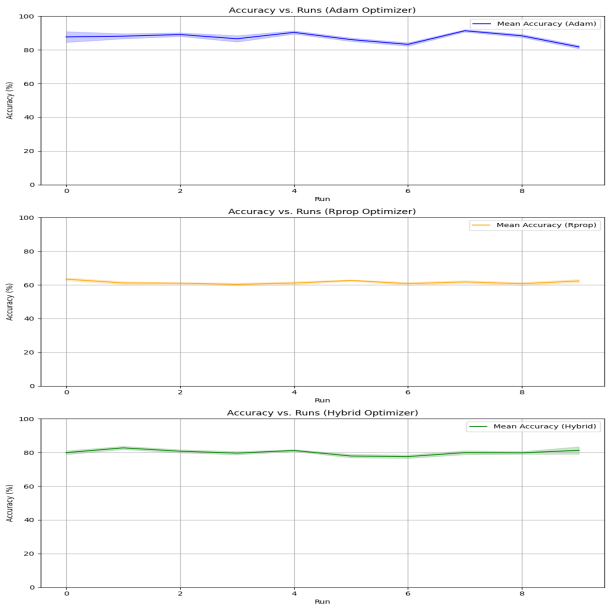

Figure 9: Comparative Analysis of Accuracy Across Different Optimizers in Neural Network Training)

This set of graphs illustrates the mean accuracy of a neural network model trained using three different optimization algorithms: Adam, Rprop, and a Hybrid approach, across ten independent runs (with random seeds ranging from 40 to 49). **Each graph presents the mean accuracy with a shaded area representing the standard deviation**.

1. Adam Optimizer: demonstrates a generally high mean accuracy that peaks above 90% in several runs. The shaded region indicates relatively low variability, suggesting consistent performance across trials.

2. Rprop: The accuracy achieved with the Rprop optimizer is significantly lower, with a mean accuracy ranging around 60-65%. The standard deviation is wider compared to Adam, indicating higher variability in performance across the runs.

3. Hybrid: The hybrid approach yields a mean accuracy between that of Adam and Rprop, demonstrating a notable improvement over Rprop but still not matching Adam's performance. The variability in performance is moderate, reflecting a balanced consistency in accuracy across runs.

7 Conclusion

The results indicate that the Adam optimizer outperformed both the Rprop and hybrid approaches in terms of accuracy in neural network training. Rprop eis the least effective optimizer, showing the lowest accuracy among the three methods. While the hybrid approach provides a middle ground in accuracy, it is still outclassed by Adam. The hybrid method offers a quicker training time since it does not start from scratch, allowing it to leverage previous knowledge. These findings suggest that for this neural network model, the choice of optimizer plays a crucial role in determining the overall performance and reliability of the training process. Thus, while Adam excels in accuracy, the hybrid approach may be preferred when time constraints are in place.

References

[1] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2014. https://arxiv.org/abs/1412.6980

[2] M. I. Riedmiller and H. Braun, *A direct adaptive method for faster backpropagation learning: the Rprop algorithm*, In *Proceedings of the IEEE International Conference on Neural Networks*, 1993. https://doi.org/10.1109/ICNN.1993.298623

[3] L. M. P. de Lima, R. J. M. Nascimento, and J. M. de S. Carvalho, *Hybrid Optimization Algorithm for Efficient Feature Selection in High-Dimensional Data*, In *2021 IEEE Latin America Transactions*, vol. 19, no. 3, pp. 429-436, 2021. https://doi.org/10.1109/TLA.2021.9443483