
ForestKNN - Combining Random Forests and KNN

xAI-Proj-M: Master Project Explainable Machine Learning

Ali Ostadi*

Otto-Friedrich University of Bamberg
96049 Bamberg, Germany
ali.ostadi@stud.uni-bamberg.de

Andreas Franz Schwab†

Otto-Friedrich University of Bamberg
96049 Bamberg, Germany

andreas-franz.schwab@stud.uni-bamberg.de

Abstract

In an era where AI systems like ChatGPT consume electricity equivalent to powering 180,000 U.S. households daily (Gordon, 2024), efficient machine learning methods are crucial. This project enhances k-Nearest Neighbour (kNN) methods for image classification using vector databases and latent space analysis. We explored the CIFAR-10 and CIFAR-100 (Krizhevsky, 2009), as well as the DermaMNIST and BreastMNIST datasets (Yang et al., 2021, 2023), visualizing data clusters with t-SNE plots and examining neighborhood structures. Additionally, we implemented a simple kNN in PyTorch and compared its performance to a trained Linear Layer. Our key contribution, ForestKNN, integrates the principles of Random Forests with kNN by using multiple kNN classifiers on random subsets of samples and features. This approach enhances speed while maintaining accuracy comparable to linear probing. Although ForestKNN's training process remains somewhat resource-intensive, it is less so than training a linear layer and yields faster results, making it a promising alternative for image classification tasks. For detailed implementation and results, visit our [Git Repository](#).

1 Introduction

Deep learning has revolutionized fields such as medical image analysis, leading to significant advancements. This project explores the core principles of deep learning and applies them in a practical setting, focusing on three main areas: understanding vector databases and latent spaces, comparing the performance of KNN and Linear Layer models, and developing an enhanced kNN algorithm to effectively handle modern, complex datasets.

At the center of our project is the hypothesis that specific adaptations and refinements of deep learning techniques can significantly improve model performance on both simple and complex classification tasks. We hypothesized that by refining our SimpleKNN, we can achieve efficient training with low computational cost, providing comparable or even better results than linear models.

This report is structured as follows: We begin by introducing the purpose and method for our RandomForest-inspired kNN. We then outline the theoretical foundations, including Data Resampling, Feature Projection, Ensembling, Bayesian Optimization, and Majority Voting. Following this, we describe the architectures of the models investigated, starting from SimpleKNN and Simple Linear Layer to the more advanced ForestKNN. An overview of the datasets used—CIFAR-10, CIFAR-100, DermaMNIST, and BreastMNIST—is provided to set the context for our investigation. We present the results obtained from each model, addressing the specific challenges encountered. Finally, the discussion section critically evaluates our findings and concludes with insights on the potential impact of our research on medical image analysis.

*Degree: M.Sc. WI, matriculation #: 2128150

†Degree: M.Sc. AI, matriculation #: 2017990

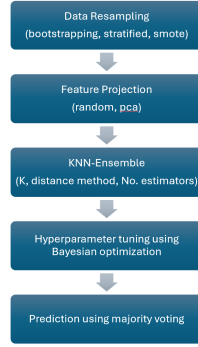


Figure 1: Outline of our purpose method

2 Methods

2.1 Theoretical Foundations

2.1.1 Purpose Method

In this project, we propose a novel machine learning method that enhances the traditional Random Forest (Breiman, 2001) algorithm by integrating K-Nearest Neighbors (KNN) as the base model instead of Decision Trees and we named it Forest-KNN. Our method leverages the strengths of KNN within a Random Forest framework, resulting in a more robust and accurate predictive model. Figure 1 illustrates the flow chart of our proposed model Forest-KNN. First, we resample the training data to create multiple subsamples. This step involves various resampling techniques, such as stratified sampling (Cohen, 2011), SMOTE (Synthetic Minority Over-sampling Technique) (Chawla et al., 2011), and Bootstrapping sampling. Next, we perform feature projection to reduce the dimension. We use two main techniques: Random Gaussian Projection (Nabil, 2017), and Principal Component Analysis (PCA) (Wold et al., 1978). After that, we train the Ensemble-KNN on resampled projected subsamples of training data and employ the Bayesian optimization algorithm (Jia Wu et al., 2019) to find the best configuration for the Ensemble-KNN and the optimal resampling and projection methods. Finally, after training the model and identifying its best configuration and hyperparameters, we use a majority vote to determine the final prediction. Our method combines advanced techniques in data sampling, dimensional reduction, and hyperparameter tuning to significantly improve the performance of Random Forests with kNN base models. This approach not only enhances accuracy but also ensures the model is efficient and robust across different types of data.

2.1.2 Data Resampling

We utilize various bootstrapping techniques to create multiple samples from the original dataset.

Stratified Sampling: Stratified sampling ensures that each subsample has the same proportion of classes as the original dataset. This is particularly useful for maintaining class distribution in imbalanced datasets (Cohen, 2011).

SMOTE (Synthetic Minority Over-sampling Technique): SMOTE generates synthetic samples for the minority class by interpolating between existing minority class samples. This helps balance the dataset by increasing the number of minority class samples (Chawla et al., 2011).

3- Bootstrapping: Bootstrapping involves creating multiple samples from the original dataset by sampling with replacement. Each bootstrap sample is used to train a model, and the predictions are aggregated to improve accuracy and robustness.

2.1.3 Feature Projection

Our model uses two primary dimensionality reduction techniques to handle high-dimensional data efficiently: Random Gaussian Projection and Principal Component Analysis (PCA). These methods

help to simplify the data while preserving its essential characteristics, improving both the performance and efficiency of our machine learning model.

Random Gaussian Projection: Random Gaussian Projection reduces the dimensionality of the data by projecting it onto a lower-dimensional subspace using a random matrix with Gaussian-distributed entries. This method is computationally efficient and preserves the distances between data points [Nabi \(2017\)](#).

- **Formula:** $X' = X \cdot R$

where X is the original data matrix with dimensions $N \times D$ (N samples and D features), R is a random matrix with dimensions $D \times d$ (D original features and d reduced dimensions), and X' is the transformed data matrix with dimensions $N \times d$. The entries of R are drawn from a Gaussian distribution with mean 0 and variance $\frac{1}{d}$.

Principal Component Analysis (PCA): PCA is a statistical technique that transforms the data into a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first principal component, the second greatest variance on the second principal component, and so on. PCA captures the most significant features of the data, making it easier to analyze and visualize [\(Wold et al., 1978\)](#).

- **Steps and Formulas:**

1. **Standardize the Data:** $X_{\text{standardized}} = \frac{X - \mu}{\sigma}$
where X is the original data matrix, μ is the mean, and σ is the standard deviation.
2. **Compute the Covariance Matrix:** $\Sigma = \frac{1}{N-1} X_{\text{standardized}}^T X_{\text{standardized}}$
where Σ is the covariance matrix.
3. **Compute the Eigenvalues and Eigenvectors:** Solve the eigenvalue equation:
$$\Sigma v = \lambda v$$

where λ are the eigenvalues and v are the eigenvectors.
4. **Select Principal Components:** Choose the top d eigenvectors corresponding to the largest eigenvalues to form a matrix V with dimensions $D \times d$.
5. **Transform the Data:** $X' = X_{\text{standardized}} \cdot V$
where X' is the transformed data matrix with dimensions $N \times d$.

2.1.4 KNN-Ensemble

The KNN-Ensemble structure illustrated in Figure 2 combines multiple KNN models to create a robust and accurate predictive model.

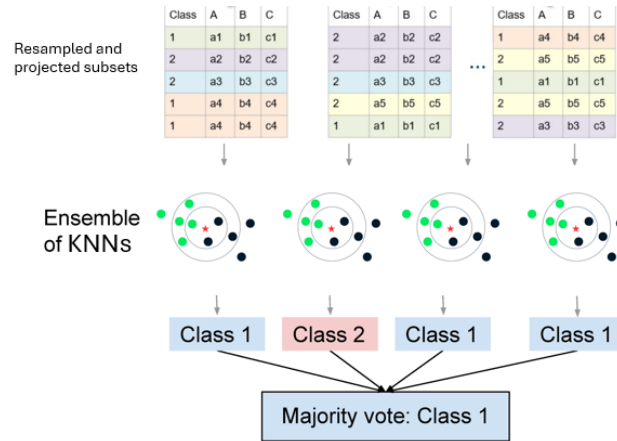


Figure 2: KNN-Ensemble structure

KNN-Ensemble contains multiple KNN models, each represented by a circle. These models are trained on different resampled projected subsets of the training data. Each KNN model predicts a

class label for the input. The final prediction is determined by majority voting. The class with the most votes among the individual KNN model predictions is selected as the final output. Three key hyperparameters are tuned in the kNN-Ensemble structure to optimize performance. (1) number of estimators which refers to the number of kNN models in the ensemble. (2) K (Number of Neighbors). This parameter determines how many nearest neighbors each kNN model considers when making a prediction. (3) Distance Method. Different distance metrics can be used to calculate the distance between data points. The choice of distance metric can significantly affect the performance of the kNN models. The three common distance methods used are:

1. Cosine Distance:

- Measures the cosine of the angle between two vectors. It is useful for comparing the orientation of the data points rather than their magnitude.

$$\text{Cosine Distance} = 1 - \cos(\theta) = 1 - \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

2. Manhattan Distance:

- Also known as L1 distance, it calculates the distance between two points by summing the absolute differences of their coordinates. It is useful for grid-like data structures.

$$\text{Manhattan Distance} = \sum_{i=1}^n |A_i - B_i|$$

3. Euclidean Distance:

- Also known as L2 distance, it measures the straight-line distance between two points in space. It is the most common distance metric.

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$$

2.1.5 Bayesian Optimization

Bayesian Optimization is a powerful method for optimizing the hyperparameters of machine learning models. It is particularly effective for tuning models with expensive evaluation functions, where traditional grid search or random search methods may be inefficient. The main goal of Bayesian Optimization is to find the optimal set of hyperparameters that minimize or maximize the objective function, often the validation error of the model (Jia Wu et al., 2019).

Key Concepts

1. Surrogate Model:

- Bayesian Optimization uses a probabilistic surrogate model to approximate the objective function. A common choice for the surrogate model is Gaussian Process (GP).
- Formula:** The GP model predicts the mean (μ) and variance (σ^2) of the objective function at a given point.

$$f(x) \sim GP(\mu(x), \sigma^2(x))$$

2. Acquisition Function:

- The acquisition function determines the next point to evaluate by balancing exploration (uncertainty) and exploitation (expected improvement).
- Common acquisition functions include Expected Improvement (EI) and Upper Confidence Bound (UCB).
- Formula for Expected Improvement (EI):**

$$EI(x) = (\mu(x) - f(x^+))\Phi(Z) + \sigma(x)\phi(Z)$$

Where $Z = \frac{\mu(x) - f(x^+)}{\sigma(x)}$, Φ is the cumulative distribution function, and ϕ is the probability density function of the standard normal distribution.

The Bayesian optimization algorithm is shown in Algorithm 1, where $D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$ represents the training dataset which consists of $t - 1$ observations of the function f . From the description of the algorithm, we can see that the whole algorithm is composed of two parts: Updating the posterior distribution (steps 3 and 4) and maximizing the acquisition function (step 2). As the observations accumulate, the posterior distribution is updated continuously; based on the new posterior, the point where the acquisition function is maximized is found and added to the training dataset. The whole process is repeated until the maximum number of iterations is reached or the difference between the current value and the optimal value obtained so far is less than a predefined threshold. It is noted that Bayesian optimization does not require the explicit expression of the function f (Snoek et al., 2012).

Algorithm 1 Bayesian optimization

```

1: for  $t = 1, 2, \dots$  do
2:   Find  $x_t$  by optimizing the acquisition function  $u$  over function  $f$ :

$$x_t = \arg \max_x u(x | D_{1:t-1}).$$

3:   Sample the objective function:  $y_t = f(x_t)$ .
4:   Augment the data  $D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$  and update the posterior of function  $f$ .
5: end for
```

2.1.6 Majority voting

After training and finding the best configuration of the Forest-KNN model, we use majority voting to determine the final prediction. Each KNN model in the ensemble makes its prediction, and the class with the most votes across these predictions is selected as the final output. This method ensures that the final decision leverages the strengths of all individual models, enhancing robustness and accuracy.

2.2 Model Architectures

2.2.1 Simple kNN

To evaluate the performance of the k-Nearest Neighbour (kNN) classifier for Step 1, we implemented a custom kNN model using PyTorch, enabling the use of GPU acceleration. The kNN algorithm classifies a data point by identifying the k closest points in the training set and assigning the majority class among these neighbors. The kNN classifier is initialized with the number of neighbors k and the device (CPU or GPU) on which computations are performed. The training process involves storing the training data and corresponding labels as PyTorch tensors on the specified device. During prediction, the classifier computes pairwise distances between the input data and the training data using the Euclidean distance metric. The nearest neighbors are identified by sorting these distances. The class labels of the nearest neighbors are then used to predict the class of each input sample by taking the mode (most common label) of these labels.

2.2.2 Simple Linear Layer

The Linear Probing method involves training a simple linear classifier using the feature representations (embeddings) from the datasets. This method aims to improve classification by using a single layer neural network directly on the embeddings. The Linear Layer is initialized with the input dimension, representing the feature size of the embeddings, and the number of classes in the dataset. The training process involves defining a loss function (Cross-Entropy Loss) and an optimizer (Adam). The model is trained by performing forward and backward passes to minimize the loss function over a series of epochs. The training data is fed into the linear layer, and the model parameters are updated to improve classification accuracy. Hyperparameter tuning is performed by training the model with various learning rates, batch sizes, and epochs. The performance on the validation set is monitored to identify the best set of hyperparameters.

2.2.3 ForestKNN

To assess the capabilities of the Forest k-Nearest Neighbors (Forest-KNN) model, we developed an ensemble of custom KNN classifiers with PyTorch, utilizing GPU acceleration for faster processing.

The Forest-KNN model aggregates the predictions from multiple KNN classifiers, each trained on distinct subsets of the dataset. This implementation involves storing the training data and labels as PyTorch tensors on the GPU. During inference, the model calculates the distances between each input data point and the stored training points using the chosen distance metric, such as Euclidean. Each KNN classifier identifies its k nearest neighbors from its specific training subset and predicts the input data’s class based on the majority class among these neighbors. The ensemble’s final prediction for each sample is determined by combining the individual predictions from all classifiers, usually through a majority voting mechanism. This ensemble method enhances prediction reliability and accuracy by leveraging diverse training subsets.

3 Experiments

3.1 Datasets

Our experimental framework focuses on the use of four central datasets: CIFAR-10, CIFAR-100, DermaMNIST, and BreastMNIST, each of which is briefly introduced here.

3.1.1 CIFAR-10

The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 different classes, with 6,000 images per class, ensuring a balanced dataset. These classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks (Krizhevsky, 2009). The dataset is split into a training set of 50,000 images and a test set of 10,000 images. Each image is labeled with one of the 10 classes, providing a robust benchmark for evaluating image recognition algorithms. The embeddings provided for this dataset have 768 dimensions. Some examples of the CIFAR-10 dataset can be seen in Figure 3.

3.1.2 CIFAR-100

Similar to CIFAR-10, the CIFAR-100 dataset contains 60,000 32×32 color images, balanced across 100 classes, with 600 images per class. The classes are grouped into 20 superclasses, with each image labeled at two levels of granularity: a coarse label (superclass) and a fine label (class) (Krizhevsky, 2009). The dataset is also divided into a training set of 50,000 images and a test set of 10,000 images. CIFAR-100 is more challenging than CIFAR-10 due to its larger number of classes and the finer granularity of the labels. The embeddings provided for this dataset have 768 dimensions. Some examples are also given in Figure 3.

3.1.3 DermaMNIST

The DermaMNIST subset of the MedMNIST collection focuses on dermatological image classification, specifically for skin lesion analysis. The dataset includes 10,015 “multi-source dermatoscopic images of common pigmented skin lesions” (Yang et al., 2021, 2023). The images are classified into seven categories: actinic keratoses, basal cell carcinoma, benign keratosis-like lesions, dermatofibroma, melanocytic nevi, melanoma, and vascular lesions (Tschandl, 2018). Originally, the images were high resolution ($3 \times 600 \times 450$), but for this project, they were resized to $3 \times 28 \times 28$ pixels. The dataset is split into a training set of 7,007 samples, a validation set of 1003 samples, and a test set of 2,005 samples. The embeddings provided for this project for this dataset have 768 dimensions. Some examples are shown in Figure 3.

3.1.4 BreastMNIST

The BreastMNIST subset is another component of the MedMNIST collection, designed for breast ultrasound image classification. It contains 780 images categorized into three classes: normal, benign, and malignant. For this project, the task was simplified to binary classification “by combining normal and benign as positive and classifying them against malignant as negative” (Yang et al., 2021, 2023). The original high-resolution images (500×500) were resized to 28×28 pixels. The dataset is divided into a training set of 546 samples, a validation set of 78 samples, and a test set of 156 samples. The given embeddings also were of size 768. Some examples of the BreastMNIST dataset can be seen in Figure 3.

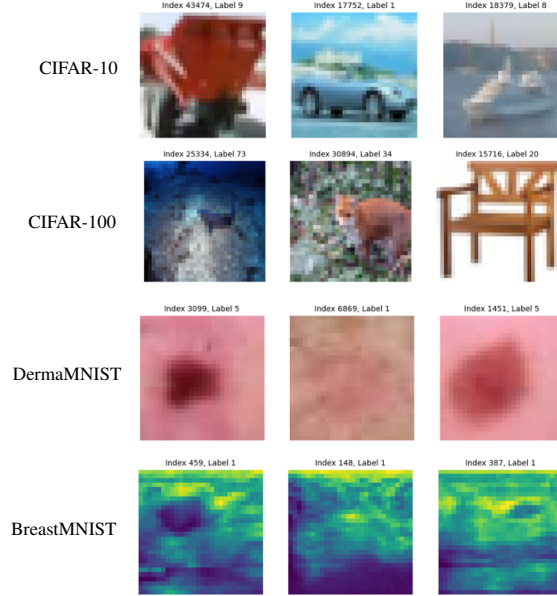


Figure 3: Example images of the datasets: CIFAR-10, CIFAR-100, DermaMNIST, and BreastMNIST.

3.2 Data Preparation

For loading the data, we used the given embeddings and labels from the CIFAR-10, CIFAR-100, DermaMNIST, and BreastMNIST datasets, stored in numpy files. The indices of the vector databases matched those of the original image databases, allowing us to map the embeddings to the original images for representation purposes. For datasets without predefined validation sets, the training data was split into training and validation sets (80/20 split) to facilitate model evaluation and hyperparameter tuning. The data was normalized to ensure consistent feature scaling, which is important for the performance of distance-based algorithms like kNN.

3.3 Step 1 - Analyze Data

The first step of this project involved analyzing and understanding the given embedding space of the data. We began by visualizing the latent spaces using t-distributed Stochastic Neighbor Embedding (t-SNE) plotting (van der Maaten and Hinton, 2008), which reduced the high-dimensional data (768 dimensions) to two dimensions. This reduction revealed patterns and class overlaps, allowing us to create a colored 2D plot that showed clusters for each label within the data.

Simply having an overview was not sufficient to gain deeper insights, so we examined randomly selected indices and analyzed their nearest neighbors. This approach enabled us to identify connections and similarities among neighboring points. However, the random selection often resulted in similar images for the same label, lacking diversity.

To enhance our evaluation, we specifically focused on points in the dataset that had several nearest neighbors with different labels. For example, as shown in Figure 4, we selected a point with index 151 from the CIFAR-10 dataset and identified its five nearest neighbors. This method allowed us to identify edge cases where points were positioned between several classes, helping us understand why an image was labeled as it was.

Additionally, we marked the selected index within the t-SNE plot, as well as its nearest and furthest points. Although the t-SNE plot might not visually represent the nearest or furthest points accurately due to dimensionality reduction, these points are indeed the true neighbors or furthest points in the original high-dimensional space. This approach, illustrated in Figure 4 for the CIFAR-10 dataset, has helped us to understand the internals of the embedding space a little better. Similar t-SNE plots and selected indices with neighbors for the other datasets can be found in the appendix: Figure 6, Figure 7, Figure 8.

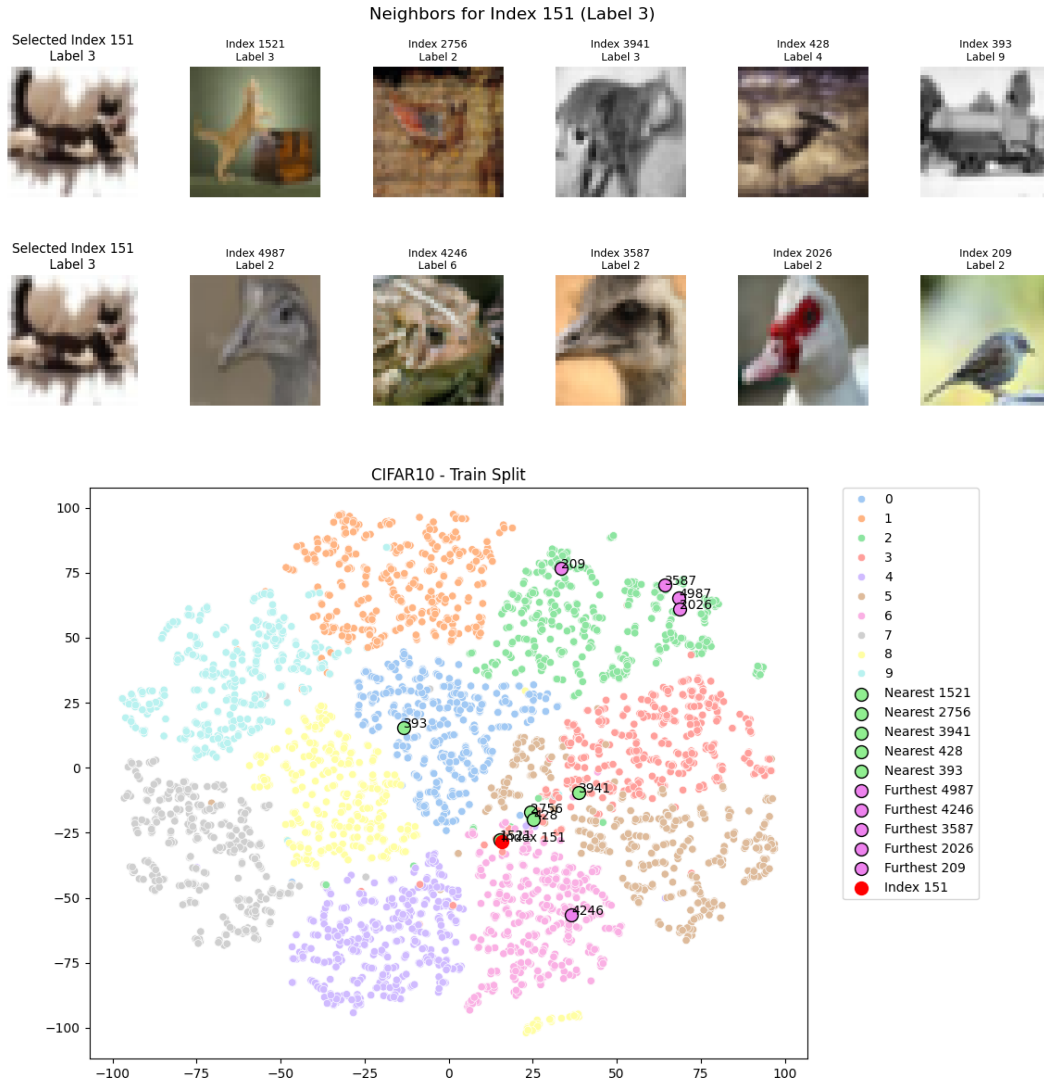


Figure 4: Top: Nearest neighbors for the point with index 151 in CIFAR-10 dataset, illustrating class overlap. Bottom: t-SNE plot of CIFAR-10 dataset showing clusters of different labels in a 2D space.

3.4 Step 2 - Compare SimpleKNN with Linar Layer

As working with kNNs is a novel task for us, our initial idea was to briefly introduce ourselves to this domain and use a simple approach to work with the given vector databases. For this, we created a simple kNN class as outlined in Section 2.2.1. Our goal was to evaluate different values of k (1, 3, 5, 7, 9, 11, 13, 15, 20, 30, 40) and assess the effectiveness of these values. For comparison, we also implemented a simple, one-layered linear model, which is described in Section 2.2.2. For this model, we evaluated several hyperparameters.

We started with the kNN model, training it on the training set and evaluating its performance on the validation set for each value of k . The validation accuracy was calculated for each k , and the optimal k value was determined based on the highest validation accuracy. The best k value was then used to evaluate the model on the test set. We recorded performance metrics such as accuracy, precision, recall, and F1 score to comprehensively assess the classifier's effectiveness. Additionally, we recorded the indices of misclassified samples but were unable to determine the reasons behind these misclassifications, as the internal learning process of the models does not provide clear insights into their decision-making.

In parallel, we trained the linear layer model. We performed hyperparameter tuning by conducting a grid search over the specified learning rates (0.001, 0.005), batch sizes (32, 64), and number of epochs (20, 40). The model was trained on the training set and validated on the validation set for each combination of hyperparameters. The best set of hyperparameters was selected based on the highest validation accuracy. The final model, trained with the best hyperparameters, was then evaluated on the test set, and performance metrics were recorded similarly to the kNN model.

A detailed comparison of the performance metrics for both methods across all datasets can be found in the Appendix (Table 3).

3.5 Step 3 - ForestKNN

The problem of implementing a ForestKNN for step 3 was a major challenge as we had to significantly improve our simple kNN approach. To understand the impact of each component described above, we gradually improved our original kNN model by adding functionality incrementally. This systematic approach allowed us to evaluate the contribution of each component to overall performance. First, we added bootstrapping to our kNN model. Bootstrapping involved creating multiple subsets of the training data through random sampling with replacement. Each subset was used to train a separate kNN classifier. This improvement was intended to improve the robustness of the model by reducing the variance. As the BreastMNIST and DermamNIST datasets were unbalanced, we also attempted to equalize the data using the Synthetic Minority Over-sampling Technique (SMOTE). However, this approach resulted in significantly lower accuracy, so we discarded it. Next, we explored various distance metrics to determine which would perform best for our kNN models. We experimented with a comprehensive set of metrics, including Euclidean, Cosine, Manhattan, Chebyshev (Cantrell, 2000), Minkowski (Zezula et al., 2006), Mahalanobis (Mahalanobis, 2018), and Canberra (Lance and Williams, 1966) distances. Each metric was evaluated, and we selected the most effective metrics for our final ForestKNN approach. In addition, we experimented with using different k-values for different subtrees within the ensemble. The evaluated k-values included groups such as [1, 3, 5], [3, 5, 7] and [5, 7, 9]. This approach allowed us to include diversity within the ensemble by using different k-values for different classifiers. The final step in our enhancement process involved combining all approaches using Bayesian optimization

Detailed results of the intermediate steps and the final model’s performance are provided in Appendix Table 4. It is worth noting that we only evaluated the results of the stepwise approach for the BreastMNIST dataset, as the overall complexity of the ForestKNN increased with additional features. Given our limited computational power locally, we were unable to compute the results for the other, larger datasets such as CIFAR-10, CIFAR-100, and DermamNIST. Especially when trying various distances, we quickly reached the limit and encountered GPU memory errors. We have therefore decided to limit ourselves to a smaller selection.

3.5.1 Tuning Hyperparameters

Table 1 outlines the key hyperparameters and their respective search spaces that are tuned in our Ensemble KNN model using Bayesian Optimization. These hyperparameters include the resampling method (SMOTE, Stratified Sampling, Bootstrapping), the projection method (Random Gaussian Projection, PCA), the number of components in random projection (50, 100, 150), the variance threshold in PCA (0.90, 0.99), the number of estimators (10, 30), the number of nearest neighbors (K) (3, 10), and the distance metric (Cosine, Euclidean, Manhattan). This comprehensive tuning process aims to optimize the model’s performance by exploring a wide range of parameter values.

4 Discussion

4.1 Step 2

The results from Step 2 highlight several insights into the performance of simple k-Nearest Neighbor (kNN) and Linear Probing methods across various datasets. Overall, Linear Probing generally outperformed kNN, particularly in more complex datasets.

For CIFAR-10, both methods performed well, with kNN achieving a slightly higher validation accuracy of 98.70% compared to 98.62% for Linear Probing. In the more complex CIFAR-100

Table 1: Hyperparameters tuning search space

Hyperparameter	Search space
Resampling method	[SMOTE, Stratified, Bootstrapping]
Projection method	[Random Gaussian Projection, PCA]
Number of components of random projection	[50, 100, 150]
Variance threshold in PCA	(0.90, 0.99)
Number of estimators	(10, 30)
K (number of nearest neighbors)	(3, 10)
Distance metric	[Cosine, Euclidean, Manhattan]

dataset, Linear Probing was superior, achieving 90.48% validation accuracy versus kNN’s 89.50%. DermaMNIST results were close, with kNN slightly ahead at 79.35% validation accuracy compared to Linear Probing’s 78.60%. For BreastMNIST, Linear Probing significantly outperformed kNN, with a validation accuracy of 85.26% compared to 79.49%.

These results suggest that both methods are suitable for datasets with distinct class boundaries, while Linear Probing is better at handling datasets with higher class diversity, likely due to its ability to learn feature weights. Despite kNN’s slight edge in some cases, the higher validation accuracy of Linear Probing indicates better generalization to unseen data, highlighting its potential for medical image classification.

Overall, the findings suggest that while the simple kNN method provided reasonable results, it was not optimized for high performance. Linear Probing, despite using only a single linear layer, showed promising results that could be significantly improved with more advanced architectures and hyperparameter optimization. The detailed comparison of performance metrics in the Appendix supports these conclusions and suggests directions for future enhancements.

4.2 Step3

Table 2: Forest-KNN results across datasets

	Cifar 10	Cifar100	DermaMNIST	BreastMNIST
Resampling method	Bootstrapping	Stratified	Stratified	Bootstrapping
Projection method	Random G	Random G	Random G	Random G
Number of components of random projection	139	100	72	94
Variance threshold in PCA	-	-	-	-
Number of estimators	28	20	16	26
K (number of nearest neighbors)	9	5	3	3
Distance metric	Euclidean	Euclidean	Euclidean	Euclidean
Accuracy	0.9873	0.899	0.880	0.871

Table 2 shows the selected best hyperparameters and the accuracy of our purpose model Forest-KNN across various datasets. For the CIFAR-10 dataset, using bootstrapping and random Gaussian projection with 139 components, 28 estimators, and 9 nearest neighbors, the model achieved an impressive accuracy of 98.73%. Similarly, for CIFAR-100 and BreastMNIST, stratified sampling and random Gaussian projection yielded accuracies of 89.9% and 87.1%, respectively, with optimal settings. For DermaMNIST, stratified sampling with 72 components, 16 estimators, and 3 nearest neighbors led to an accuracy of 88.0%. All datasets consistently performed best with the Euclidean distance metric. Furthermore, we conducted a Feature importance analysis to determine the impact of each hyperparameter on model accuracy. Figure 5 depicts the importance of various hyperparameters on model accuracy for the BreastMNIST, DermaMNIST, CIFAR-10, and CIFAR-100 datasets. For BreastMNIST and DermaMNIST, the sampling method is the most influential hyperparameter, followed by the number of nearest neighbors (K), indicating the significant impact of resampling techniques and local data structures on these medical imaging datasets. The number of components in the random

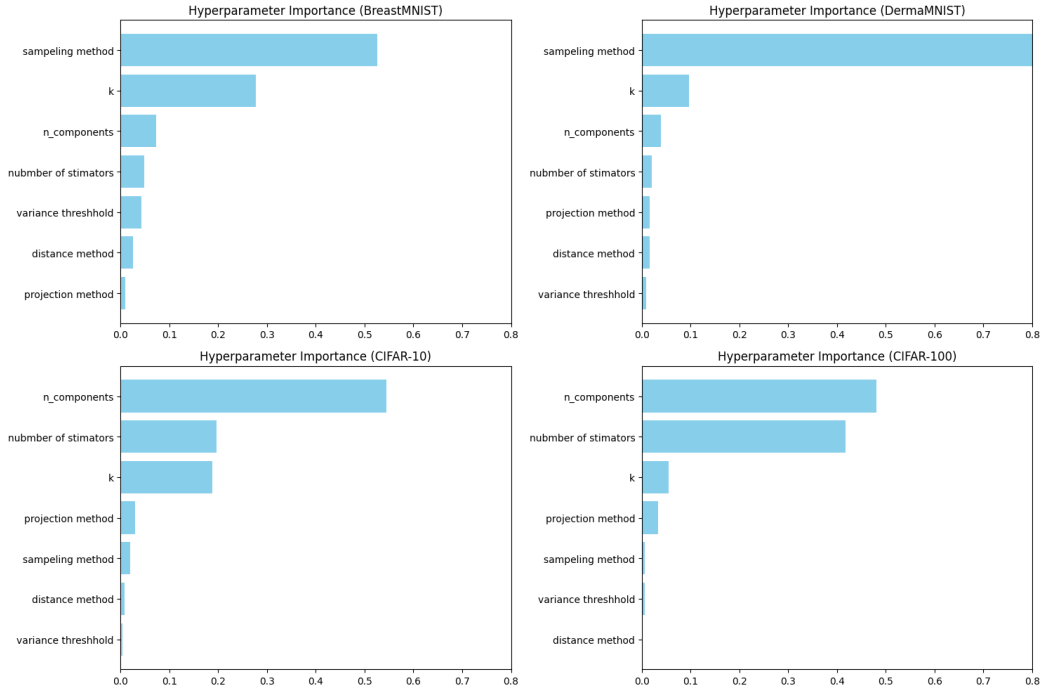


Figure 5: Forest-KNN Hyperparameters feature importance o

projection also plays a notable role. In contrast, for CIFAR-10 and CIFAR-100, the number of components and the number of estimators are the most important hyperparameters, highlighting the critical role of dimensionality reduction and ensemble size in these image classification tasks. Other hyperparameters, such as the distance method, variance threshold, and projection method, have minimal impact across all datasets. These insights emphasize the need for careful tuning of specific hyperparameters to optimize model performance for different types of data.

5 Conclusion

In this project, we developed and refined the Forest-KNN model by combining k-Nearest Neighbor (kNN) algorithms with principles from Random Forests for image classification tasks. Our approach was applied to various datasets, including CIFAR-10, CIFAR-100, DermaMNIST, and BreastMNIST. We focused on enhancing the robustness and accuracy of the kNN model by employing resampling methods to create multiple training subsets and exploring different distance metrics such as Manhattan, Euclidean, and Cosine. Bayesian optimization was used to fine-tune the model parameters, achieving optimal performance. Our results indicated that Forest-KNN is a robust and efficient alternative to traditional kNN and linear models, capable of handling complex datasets effectively. However, the added functionality comes with an increased computational cost. While there were slight improvements across various datasets compared to simpler approaches, they were not always the highest. Nonetheless, the Forest-KNN model demonstrated improvements in classification accuracy and robustness, particularly in medical image classification, where precision is crucial. Future work could involve further optimization, scaling to larger datasets, and exploring additional ensemble strategies and distance metrics. This project highlights the potential of ensemble methods to enhance classification accuracy and robustness, providing valuable insights into the practical application of machine learning in complex datasets.

References

- L. Breiman. Random forest. 45(1):5–32, 2001. ISSN 08856125. doi: 10.1023/A:1010933404324. URL <http://link.springer.com/10.1023/A:1010933404324>.

- C. D. Cantrell. *Modern mathematical methods for physicists and engineers*. Cambridge University Press, 2000.
- N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. 2011. doi: 10.48550/ARXIV.1106.1813. URL <https://arxiv.org/abs/1106.1813>. Publisher: arXiv Version Number: 1.
- M. P. Cohen. Stratified sampling. In M. Lovric, editor, *International Encyclopedia of Statistical Science*, pages 1547–1550. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-04897-5 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_574. URL http://link.springer.com/10.1007/978-3-642-04898-2_574.
- C. Gordon. Chatgpt and generative ai innovations are creating sustainability havoc. *Forbes*, 2024. URL <https://www.forbes.com/sites/cindygordon/2024/03/12/chatgpt-and-generative-ai-innovations-are-creating-sustainability-havoc/>. Accessed: 2024-07-27.
- Jia Wu, Xiu-Yun Chen, and Hao Zhang. Hyperparameter optimization for machine learning models based on bayesian optimization. 2019. doi: <https://doi.org/10.11989/JEST.1674-862X.80904120>. URL <https://www.sciencedirect.com/science/article/pii/S1674862X19300047>.
- A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009. URL <https://www.cs.toronto.edu/~kriz/cifar.html>. Accessed: 2024-07-27.
- G. N. Lance and W. T. Williams. Computer programs for hierarchical polythetic classification (“similarity analyses”). *The Computer Journal*, 9(1):60–64, 1966.
- P. C. Mahalanobis. On the generalized distance in statistics. *Sankhyā: The Indian Journal of Statistics, Series A (2008-)*, 80:pp. S1–S7, 2018. ISSN 0976836X, 09768378. URL <https://www.jstor.org/stable/48723335>.
- M. Nabil. Random projection and its applications, 2017. URL <http://arxiv.org/abs/1710.03163>.
- J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms, 2012. URL <http://arxiv.org/abs/1206.2944>.
- P. Tschandl. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions, 2018. URL <https://doi.org/10.7910/DVN/DBW86T>.
- L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. URL <http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf>.
- S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. 2(1):37–52, 1978. ISSN 01697439. doi: 10.1016/0169-7439(87)80084-9. URL <https://linkinghub.elsevier.com/retrieve/pii/0169743987800849>.
- J. Yang, R. Shi, and B. Ni. Medmnist classification decathlon: A lightweight automl benchmark for medical image analysis. In *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 191–195, 2021.
- J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.
- P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.

Declaration of Authorship

Ich erkläre hiermit gemäß § 9 Abs. 12 APO, dass ich die vorstehende Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Des Weiteren erkläre ich, dass die digitale Fassung der gedruckten Ausfertigung der Projektarbeit ausnahmslos in Inhalt und Wortlaut entspricht und zur Kenntnis genommen wurde, dass diese digitale Fassung einer durch Software unterstützten, anonymisierten Prüfung auf Plagiate unterzogen werden kann.

Bamberg, August 2, 2024

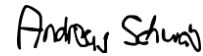
(Place, Date)

Bamberg, August 2, 2024

(Place, Date)



(Signature)



(Signature)

A Appendix

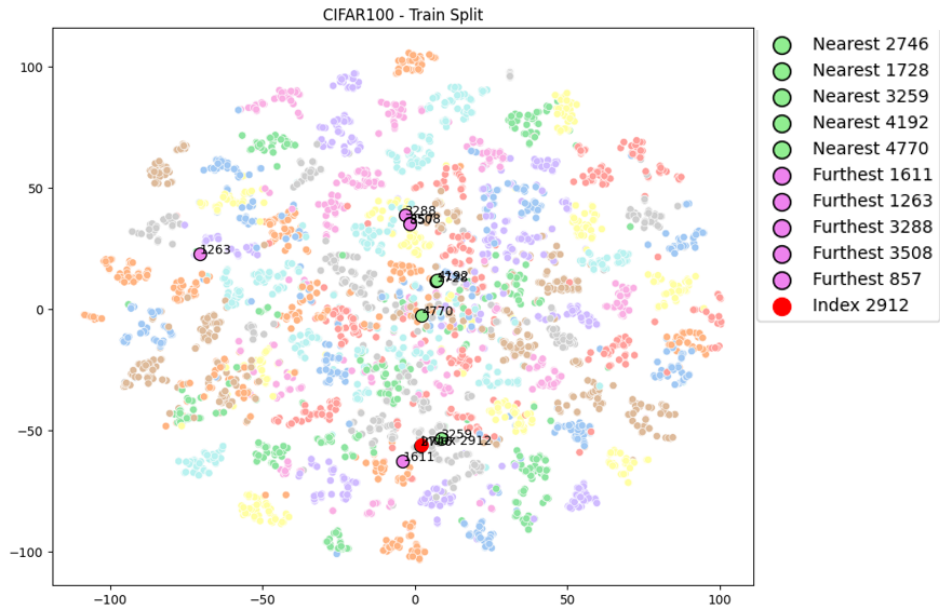


Figure 6: Nearest neighbors and t-SNE for index 2912 in the CIFAR-100 dataset.

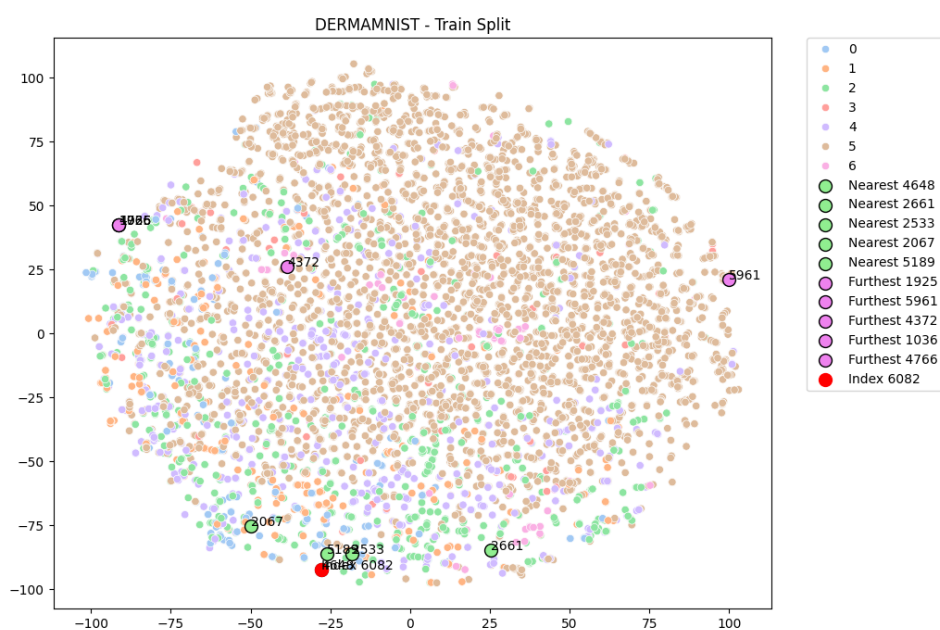
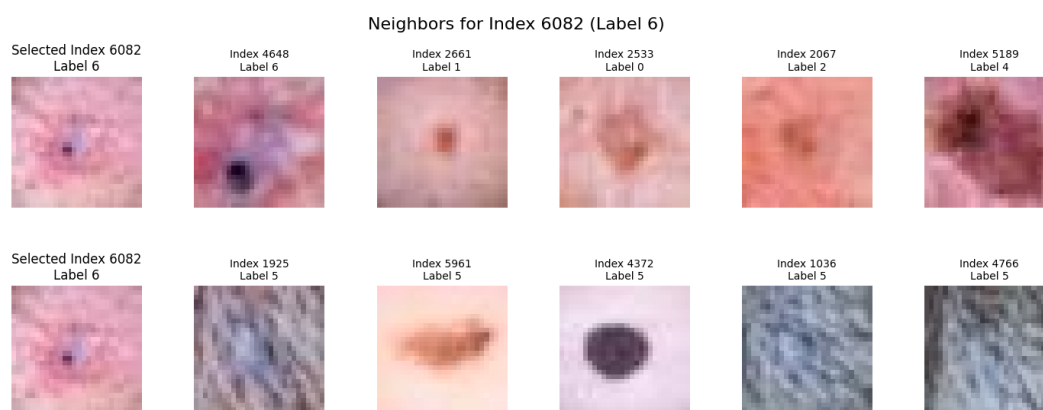


Figure 7: Nearest neighbors and t-SNE for index 6082 in DermaMNIST dataset

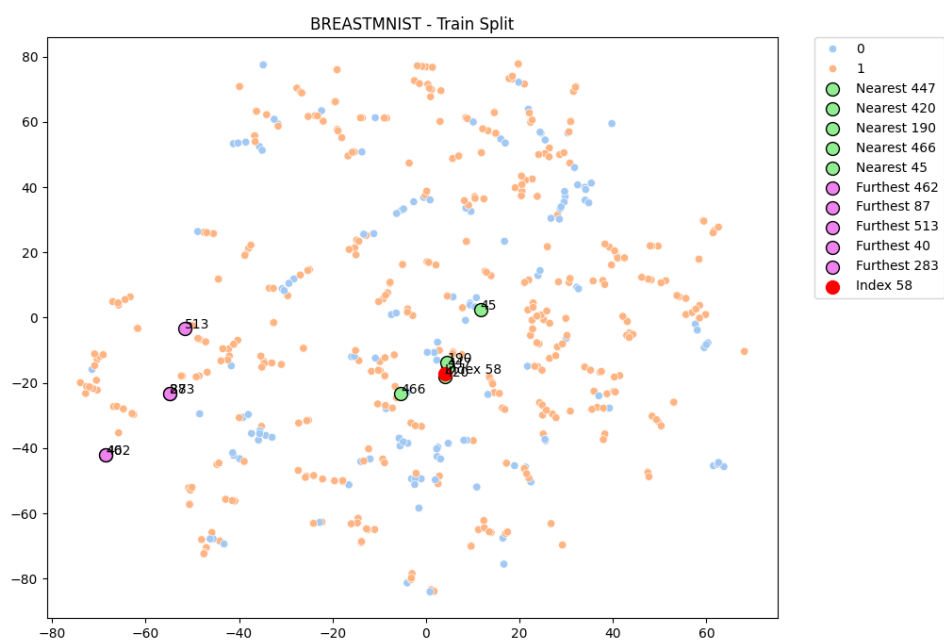
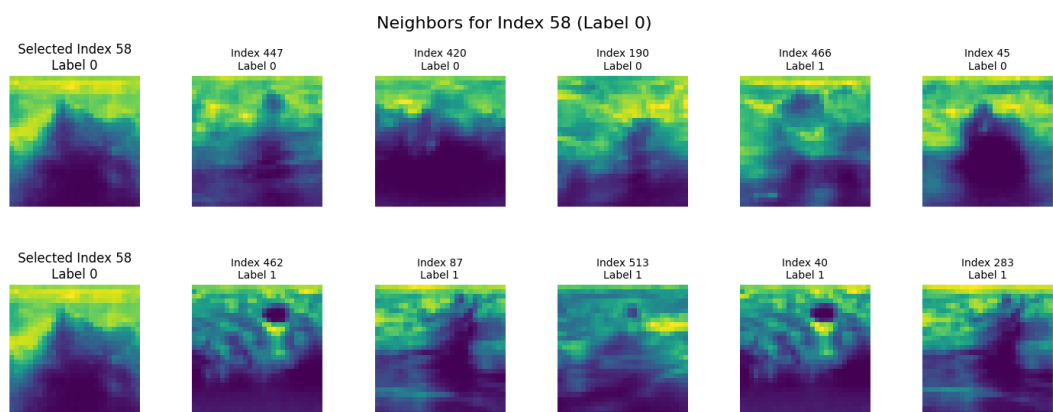


Figure 8: Nearest neighbors and t-SNE for index 58 in BreastMNIST dataset.

Table 3: Performance Comparison of Simple kNN and Linear Layer Methods for Step 2

CIFAR-10		
Method	kNN	Linear Layer
Best Parameter(s)	$k = 11$	lr=0.005, batch_size=32, epochs=40
Validation Accuracy (%)	98.90	98.84
Test Accuracy (%)	98.70	98.62
Test Precision (%)	98.71	98.62
Test F1 Score (%)	98.70	98.62
CIFAR-100		
Method	kNN	Linear Layer
Best Parameter(s)	$k = 15$	lr=0.005, batch_size=32, epochs=40
Validation Accuracy (%)	89.93	90.79
Test Accuracy (%)	89.50	90.48
Test Precision (%)	89.74	90.56
Test F1 Score (%)	89.46	90.46
DermaMNIST		
Method	kNN	Linear Layer
Best Parameter(s)	$k = 1$	lr=0.005, batch_size=32, epochs=40
Validation Accuracy (%)	78.27	79.46
Test Accuracy (%)	79.35	78.60
Test Precision (%)	80.56	77.17
Test F1 Score (%)	79.84	76.89
BreastMNIST		
Method	kNN	Linear Layer
Best Parameter(s)	$k = 9$	lr=0.005, batch_size=32, epochs=40
Validation Accuracy (%)	85.90	91.03
Test Accuracy (%)	79.49	85.26
Test Precision (%)	85.65	85.64
Test F1 Score (%)	81.53	85.41

Table 4: Validation Accuracy - BreastMNIST (Incremental Enhancement Approach)

Method	Trees	k	Additional Information	Accuracy
Only Bootstrapping	10	3		0.8846
Balance Datasets (use SMOTE)	5	3		0.5641
Different Distance Metrics	20	3	Distance: Chebyshev	0.9103
Different K on Subtrees	10	[5, 7, 9]	Distance: Canberra	0.9103