COMP 472 - Project

Steven Zrihen

November 25, 2024

1 Introduction

In this project, I worked on implementing, testing, and evaluating four well-known machine learning models: Naive Bayes, Decision Tree, Multi-Layer Perceptron , and Convolutional Neural Networks . My goal was to see how well these models perform on the CIFAR-10 dataset, a common benchmark for image classification, and to study how changing their design and settings affects their results.

The CIFAR-10 dataset includes 60,000 color images divided into 10 different categories, featuring objects like animals and vehicles. Each model was trained to recognize patterns in these images and classify them correctly. I experimented with different setups, such as changing the depth of the models, the size of their layers, and the size of their kernels, to better understand how these design and parameter changes impact their performance.

2 Naive Bayes Classifier

The Naive Bayes classifier is a simple model that uses Bayes' theorem. It assumes that the features in the data are independent of each other. This model works by calculating the probability of a class based on the given features. The formula it uses is:

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}, \tag{1}$$

where $P(C_k|\mathbf{x})$ is the posterior probability, $P(\mathbf{x}|C_k)$ is the likelihood, $P(C_k)$ is the prior probability of class C_k, and $P(\mathbf{x})$ is the evidence that normalizes probabilities across classes [1].

In this project, the Gaussian Naive Bayes (GNB) variant was utilized, which models the likelihood $P(x_i|C_k)$ for continuous features using a Gaussian distribution:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}\right), \tag{2}$$

where μ_k and σ_k^2 are the mean and variance of feature x_i for class C_k [18]. The predicted class is determined by selecting the class with the highest posterior probability:

$$\hat{C} = \arg\max_{C_k} \log P(C_k) + \sum_{i=1}^{n} \log P(x_i|C_k), \tag{3}$$

where logarithms are used for numerical stability [2].

2.1 Implementation and Variants

I considered two versions of Gaussian Naive Bayes in this project:

1. A custom Gaussian Naive Bayes classifier that I built from scratch. This version calculates the class probabilities, as well as the means and variances for each feature.

2. Scikit-learn's `GaussianNB`, which is a popular pre-built implementation. I used it as a benchmark to compare the performance of my custom model. [3].

Both versions of the Gaussian Naive Bayes classifier were tested on the PCA-reduced CIFAR-10 dataset, which has 10 classes. Since the Naive Bayes model is very simple, I didn't make any changes to its structure. Instead, I focused on evaluating its basic performance and comparing it with other models.

2.2 Training Methodology

Unlike models that rely on iterative optimization, Naive Bayes only needs one pass through the training data to calculate feature statistics. This makes training very fast and efficient since it doesn't require settings like a learning rate, epochs, or optimizers. During the training phase, the Gaussian Naive Bayes model calculated the following for each class:

- **Class Priors ($P(C_k)$):** The proportion of samples in each class.
- **Feature Means (μ_k) and Variances (σ_k^2):** Calculated for each feature conditioned on the class.

The testing phase involved computing the posterior probabilities for all classes and selecting the class with the highest probability.

2.3 Code Implementation

Below is a Python code snippet that shows how I implemented the custom version of the Gaussian Naive Bayes model:

```python
class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = {}
        self.variances = {}
        self.priors = {}

        for cls in self.classes:
            X_cls = X[y == cls]
            self.means[cls] = np.mean(X_cls, axis=0)
            self.variances[cls] = np.var(X_cls, axis=0) + 1e-9  #
                To avoid division by zero
            self.priors[cls] = X_cls.shape[0] / X.shape[0]

    def predict(self, X):
        posteriors = []
        for cls in self.classes:
            mean, var = self.means[cls], self.variances[cls]
            prior = np.log(self.priors[cls])
            likelihood = -0.5 * np.sum(np.log(2 * np.pi * var)) -
                0.5 * np.sum(((X - mean) ** 2) / var, axis=1)
            posterior = prior + likelihood
            posteriors.append(posterior)
        return self.classes[np.argmax(posteriors, axis=0)]
```

Listing 1: Custom Gaussian Naive Bayes Classifier

2.4 Advantages and Limitations

The Naive Bayes classifier assumes that all features are independent, which simplifies the calculations but can be problematic for datasets where features are strongly related, like images. However, using the PCA-reduced feature space in this study helped reduce these problems. This allowed the model to perform reasonably well compared to more advanced models. Its speed and efficiency made it a good starting point for comparison.

2.5 Conclusion

The Naive Bayes classifier served as a solid baseline for comparing other models in this study. Its simplicity and fast computation were significant benefits, but its performance also showed the limitations of its strong assumption that features are independent. [18].

3 Decision Tree Classifier

The Decision Tree model is a structured classifier that organizes data into a tree-like framework to make decisions. Each internal node selects a feature based on a splitting rule, dividing the data into smaller groups along branches that represent feature values. The process ends at leaf nodes, which assign the final class labels. In this project, I used the Gini impurity as the splitting rule, which is mathematically defined as:

$$G = 1 - \sum_{i=1}^{C} p_i^2, \tag{4}$$

where p_i is the proportion of samples belonging to class i, and C is the total number of classes. Gini impurity measures the likelihood of incorrect classification, with smaller values indicating purer splits [4].

The impurity of a split is calculated using the formula:

$$\text{Impurity}_{\text{split}} = \frac{N_L}{N} G_L + \frac{N_R}{N} G_R, \tag{5}$$

where N is the total number of samples, N_L and N_R are the number of samples in the left and right child nodes, and G_L and G_R are the Gini impurities of the respective nodes [5]. This greedy algorithm ensures locally optimal splits at each level of the tree.

The splitting process in the Decision Tree continues until a stopping condition is reached. This could be a maximum tree depth, a minimum number of samples in a leaf, or zero impurity at a node. In this project, I experimented with different maximum tree depths to study how they affect the model's performance and generalization. A deeper tree can learn more complex patterns but might overfit the data, while a shallower tree is less likely to overfit but could miss important patterns, leading to underfitting. [6].

3.1 Implementation and Variants

I used Scikit-learn's `DecisionTreeClassifier` to implement the Decision Tree model, with Gini impurity as the criterion for splitting nodes. The baseline model was set to a maximum depth of 50, enabling it to fully grow and learn complex patterns from the data. The configuration was defined as follows:

```python
from sklearn.tree import DecisionTreeClassifier

# Baseline Decision Tree
decision_tree = DecisionTreeClassifier(criterion='gini', max_depth
    =50, random_state=42)
decision_tree.fit(train_features_pca, train_labels)

# Evaluate the model
predictions = decision_tree.predict(test_features_pca)
```

Listing 2: Baseline Decision Tree Implementation

To study the trade-offs between model complexity and generalization, I experimented with different tree depths. I tested depths of 10, 20, 30, 40, and 50. The following code snippet shows how I set up these experiments:

```python
depths = [10, 20, 30, 40, 50]
results = []
```

```

for depth in depths:
    tree = DecisionTreeClassifier(criterion='gini', max_depth=depth
        , random_state=42)
    tree.fit(train_features_pca, train_labels)
    preds = tree.predict(test_features_pca)
    acc = accuracy_score(test_labels, preds)
    results.append((depth, acc))

# Print depth vs. accuracy
for depth, acc in results:
    print(f"Depth: {depth}, Accuracy: {acc:.2f}")
```

Listing 3: Decision Tree Depth Variants

These experiments helped me understand how the depth of the tree affects classification performance. They highlighted the balance between capturing complex patterns with deeper trees and maintaining generalization to avoid overfitting with shallower trees.

3.2 Training Methodology

The training process involved dividing the PCA-reduced CIFAR-10 dataset into training and testing sets. I used the Gini impurity criterion to minimize the chance of misclassification at each split. No pruning methods were applied, and the tree was trained using the entire training dataset. Since Decision Trees are deterministic and don't rely on iterative optimization, the training process was computationally efficient.

To evaluate the baseline and depth-variant models, I calculated metrics such as accuracy, precision, recall, and F1 score. These metrics were determined as follows:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}, \tag{6}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \tag{7}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \tag{8}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{9}$$

I also generated confusion matrices to examine the performance of the model for each class. These matrices helped me identify which classes were frequently misclassified and provided insights into potential weaknesses in the model.

3.3 Analysis of Depth Variations

The baseline model, with a maximum depth of 50, was able to capture complex patterns in the data. However, it showed signs of overfitting, as it modeled noise along with useful patterns. Reducing the tree depth encouraged the models to generalize better, limiting noise capture but at the cost of missing finer details. For example, a tree with a depth of 10 generalized well but lacked the ability to represent complex decision boundaries.

The results highlighted the trade-offs between model complexity and performance, consistent with established findings in decision tree optimization. While deeper trees achieved higher accuracy on the training data, they did not consistently perform better on the test data. This reinforced the need to balance tree depth for optimal generalization. [10].

3.4 Conclusion

The Decision Tree classifier showcased the typical trade-offs of tree-based models, where the depth of the tree plays a crucial role in balancing generalization and computational efficiency. By systematically testing different depths, I gained valuable insights into how the model behaves and learned the importance of fine-tuning parameters to achieve the best performance.

4 Multilayer Perceptron

MLP is a type of neural network where every neuron is connected to all neurons in the next layer. It maps input features to output predictions by applying a series of linear transformations followed by non-linear activation functions. Each layer in the MLP performs a transformation, which can be written as:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{10}$$

where $\mathbf{W}$ represents the weight matrix, $\mathbf{b}$ is the bias vector, $\mathbf{x}$ is the input vector, and f is the activation function. In this study, the ReLU activation function was employed in the hidden layers, mathematically defined as:

$$f(x) = \max(0, x). \tag{11}$$

The ReLU function introduces non-linearity into the model, enabling it to learn complex representations from the input data [8]. The final layer of the MLP uses the softmax function to convert logits into probabilities, defined as:

$$P(y = i|\mathbf{x}) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \tag{12}$$

where z_i is the logit for class i, and C is the total number of classes. This ensures that the output forms a valid probability distribution, suitable for classification tasks [18].

4.1 Baseline MLP Configuration

The baseline MLP model was implemented to process the PCA-reduced CIFAR-10 dataset. It features three fully connected layers being the first is an input layer to handle the 50-dimensional PCA-reduced features, next is a hidden layer with 128 neurons, and finally, an output layer for the 10 CIFAR-10 classes. The architecture was implemented as follows:

```python
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(50, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Listing 4: Baseline MLP Implementation

The ReLU activation function was used in the hidden layer to introduce non-linearity, while the softmax function was applied in the output layer to convert the outputs into probabilities for the 10 CIFAR-10 classes. This architecture served as a basic model to assess performance on the PCA-reduced CIFAR-10 dataset.

4.2 Experimental Variants

4.3 Deeper MLP

I implemented a deeper MLP variant by adding another hidden layer with 128 neurons. This increased the network's ability to capture complex patterns in the data but also raised the risk of overfitting, as deeper models are more prone to memorizing noise in the training data. [15].

4.4 Smaller MLP

The smaller MLP variant reduced the size of the hidden layers to 64 neurons, effectively limiting the model's capacity. This design aimed to improve generalization by preventing overfitting while also reducing computational complexity, making the model faster to train [10].

4.5 Training Methodology

All MLP variants were trained for 20 epochs using a mini-batch size of 64. The training process involved the following steps:

- **Forward Propagation:** Input features were passed through the network to compute predictions.
- **Loss Calculation:** The Cross-Entropy Loss function was used to measure the difference between predicted probabilities and true labels:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i), \tag{13}$$

  where y_i is the true label and $\hat{y}_i$ is the predicted probability for class i [18].
- **Backpropagation:** Gradients of the loss with respect to the model parameters were computed using automatic differentiation.
- **Optimization:** The SGD optimizer with a learning rate of 0.01 and momentum of 0.9 was used to update model parameters. The update rule is given by:

$$\theta_t = \theta_{t-1} - \eta \cdot (\nabla_\theta \mathcal{L} + \mu \cdot \Delta\theta_{t-1}), \tag{14}$$

  where η is the learning rate, μ is the momentum factor, and $\Delta\theta_{t-1}$ represents the parameter update from the previous iteration [17].

4.6 Implementation Process

The implementation used PyTorch, which provided flexibility for experimenting with different architectures. Each model was implemented as a separate class and trained independently to ensure fair comparisons. During training, the network was set to `train` mode, activating layers like dropout if present. For inference, the network was switched to `eval` mode to maintain consistent behavior.

The PCA-reduced CIFAR-10 dataset was used as input for all experiments. To evaluate the models, I computed metrics like accuracy, precision, recall, and F1 score, along with confusion matrices to assess performance for each class. This setup allowed me to thoroughly analyze how changes in architecture, such as varying depth and hidden layer size, impacted generalization and computational efficiency. [11].

4.7 Conclusion

The MLP experiments demonstrated the trade-offs between model capacity, generalization, and computational efficiency. The deeper MLP was better at capturing complex patterns but had a higher risk of overfitting. On the other hand, the smaller MLP generalized better but had limited ability to represent detailed patterns. These findings are consistent with established neural network research, highlighting the critical role of architectural design in achieving balanced performance.[15].

5 VGG11 Convolutional Neural Network

The VGG11 CNN is a deep learning model designed for image classification. It uses a hierarchical structure consisting of convolutional layers, pooling layers, and fully connected layers to extract and classify features from images. The convolutional layers apply localized operations on the input image using filters like kernels. These operations can be mathematically expressed as:

$$y[i,j] = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} x[i+m, j+n] \cdot w[m,n], \tag{15}$$

where $x[i,j]$ is the input at position (i,j), $w[m,n]$ represents the kernel weight, and k is the kernel size. These operations enable the extraction of spatial hierarchies, such as edges and textures, by sliding the kernel across the input image. Max-pooling layers further reduce spatial dimensions by selecting the maximum value within a window, enhancing computational efficiency and reducing sensitivity to positional changes in features [12].

Batch normalization is applied after each convolutional layer to stabilize learning by normalizing activations. This is computed as:

$$\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}, \tag{16}$$

where x is the input, μ is the mean, σ^2 is the variance, and ϵ is a small constant to prevent division by zero [13]. Non-linearity is introduced through the ReLU activation function, defined as:

$$f(x) = \max(0, x), \tag{17}$$

allowing the network to learn complex patterns [14]. Finally, fully connected layers aggregate high-level features into a final output, mapping these features to class probabilities using the softmax function:

$$P(y=i|\mathbf{x}) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \tag{18}$$

where z_i represents the logit for class i, and C is the total number of classes [18].

5.1 Baseline VGG11 Configuration

The baseline VGG11 model is structured with sequ, convolutional layers, batch normalization, and ReLU activation, combined with max-pooling layers for dimensionality reduction. The key architectural elements are:

- **Convolutional Layers:** Small 3×3 kernels capture fine details, maintaining uniformity throughout the network. The first convolutional layer transforms the 3-channel input into 64 feature maps with a stride of 1 and padding of 1.

- **Batch Normalization:** Applied after each convolutional layer to accelerate convergence and improve generalization.
- **Pooling Layers:** Max-pooling with a 2×2 kernel reduces spatial dimensions while retaining salient features.
- **Fully Connected Layers:** These layers map high-dimensional feature representations to class probabilities.

The baseline VGG11 model was implemented as follows:

```python
class VGG11(nn.Module):
    def __init__(self):
        super(VGG11, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # Additional layers follow a similar pattern...
        )
        self.classifier = nn.Sequential(
            nn.Linear(512, 4096),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, 10),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

Listing 5: Baseline VGG11 Implementation

5.2 Experimental Variants

5.3 Adding Layers

To evaluate the effect of increased depth, additional convolutional layers with 3×3 kernels were added. This variant was designed to enhance the network's ability to extract hierarchical features but introduced the risk of overfitting due to the increased model complexity. [15].

5.4 Removing Layers

In this variant, specific convolutional layers were removed to simplify the architecture. The aim was to explore how reducing depth impacts generalization and computational efficiency. Although shallower networks might lose some ability to capture fine-grained details, they are less likely to overfit. [11].

5.5 Adjusting Kernel Sizes

Separate experiments replaced the baseline \$3 \times 3\$ kernels with larger (\$5 \times 5\$) and smaller (\$2 \times 2\$) kernels. Larger kernels were used to capture broader features, such as shapes and textures, while smaller kernels focused on finer details. These experiments highlighted the trade-offs between spatial granularity and computational cost [16].

5.6 Training Methodology

The VGG11 models were trained on the CIFAR-10 dataset, which contains 50,000 training images and 10,000 test images, all resized to \$224 \times 224\$ pixels. Key training configurations included:

- **Loss Function:** Cross-Entropy Loss, defined as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i), \tag{19}$$

  where y_i and $\hat{y}_i$ are the true and predicted probabilities for class i [18].

- **Optimizer:** SGD with a learning rate of 0.01 and momentum of 0.9.
- **Batch Size:** 64 images per mini-batch.
- **Number of Epochs:** 20, with periodic validation.

The training process followed a standard pipeline which first was forward propagation to compute predictions, then loss computation using Cross-Entropy Loss. Also, backpropagation to calculate gradients, and parameter updates via SGD with momentum [17].

5.7 Implementation Process

The models were implemented using PyTorch's modular framework, which allowed for smooth experimentation. Training was performed on Google Colab with GPU acceleration to handle the computational demands efficiently. The \texttt{nn.Sequential} module made it easy to modify the architecture, such as adding layers or changing kernel sizes. Evaluation metrics, including accuracy, precision, recall, and F1 score, were calculated using Scikit-learn.

The experiments showed that adding layers enhanced the network's ability to extract hierarchical features but increased the risk of overfitting. Larger kernels were better at capturing broad patterns, while smaller kernels excelled at detecting fine-grained details. These findings are consistent with established research on convolutional networks. [12, 16].

5.8 Conclusion

The VGG11 experiments emphasized the significance of architectural changes in balancing model complexity, generalization, and computational efficiency. These results highlight the trade-offs involved in designing CNN architectures for image classification tasks. [11].

6 Evaluation of Naive Bayes Model

6.1 Evaluation Metrics

The evaluation of the Gaussian Naive Bayes model was performed using the CIFAR-10 dataset, which was reduced to 50 dimensions using PCA. Both the custom implementation and Scikit-learn's Gaussian Naive Bayes classifier delivered consistent results, with the following performance metrics:

- **Accuracy:** 0.79
- **Precision:** 0.80
- **Recall:** 0.79
- **F1 Score:** 0.79

The confusion matrix for both implementations was:

$$\begin{bmatrix}
80 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 12 & 4 \\
3 & 88 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 6 \\
7 & 0 & 62 & 8 & 7 & 4 & 11 & 0 & 1 & 0 \\
1 & 0 & 4 & 75 & 4 & 10 & 6 & 0 & 0 & 0 \\
1 & 0 & 4 & 7 & 77 & 3 & 1 & 7 & 0 & 0 \\
0 & 1 & 5 & 15 & 3 & 73 & 2 & 1 & 0 & 0 \\
2 & 0 & 4 & 6 & 6 & 3 & 78 & 1 & 0 & 0 \\
1 & 1 & 0 & 5 & 6 & 5 & 0 & 81 & 1 & 0 \\
8 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 87 & 3 \\
5 & 2 & 0 & 2 & 0 & 0 & 0 & 1 & 1 & 89
\end{bmatrix}$$

6.2 Insights into Performance

The Naive Bayes model performed well in identifying classes with distinct, easily separable features, such as classes 2 and 7. However, it faced challenges with classes that had significant overlap in their feature space, leading to frequent misclassifications, especially in classes 8 and 10. These issues stem from the Naive Bayes assumption of feature independence, which is not valid for complex datasets like CIFAR-10.

The model's precision as 0.80 is slightly higher than recall which is 0.79 shows a conservative tendency, prioritizing fewer but more accurate predictions. This aligns with the Gaussian Naive Bayes assumption of normally distributed features, which simplifies the decision boundary but reduces the model's flexibility.

6.3 Confusion Matrix Analysis

The confusion matrix showed these key insights:

- **Well-Recognized Classes:** Class 7 had the highest recognition accuracy, with very few misclassifications into other classes. This can be attributed to its distinct features, which were effectively modeled under the Gaussian assumption.
- **Frequently Confused Classes:** Class 8 was often misclassified as class 1 or class 10, likely due to overlapping features in the PCA-reduced space. These confusions highlight the limitations of the Naive Bayes model in distinguishing subtle variations among similar classes

6.4 Reflection on Model Assumptions

The performance of the Gaussian Naive Bayes model shows the trade-offs of its simplifying assumptions. While the assumption of feature independence and modeling with a Gaussian distribution make it efficient and effective on well-separated classes, these same assumptions make it harder to generalize on more complex datasets. This limitation was clear in the frequent misclassifications of visually similar classes, where the dependencies between features likely played an important role. [19].

6.5 Primary Findings

The Naive Bayes model was a good starting point for evaluating the PCA-reduced CIFAR-10 dataset. It is simple and fast, making it a good choice for datasets with clear, separated classes. However, its strong assumptions limited its performance compared to more flexible models like decision trees or neural networks. Even with these limits, its precision of 0.80 shows that it avoids false positives, which is useful in cases where more careful predictions are needed.

6.6 Conclusion

The Naive Bayes classifier is a useful baseline for classification tasks because it is efficient and easy to understand. While it worked well for the PCA-reduced CIFAR-10 dataset, its limitations show that more advanced models are needed for datasets with complex relationships between features. In the future, I could look into combining the simplicity of Naive Bayes with more flexible techniques for modeling features. [18] [19].

6.7 Code Implementation

Here is a Python code snippet that shows how I implemented the custom Gaussian Naive Bayes classifier:

```python
class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = {}
        self.variances = {}
        self.priors = {}

        for cls in self.classes:
            X_cls = X[y == cls]
            self.means[cls] = np.mean(X_cls, axis=0)
            self.variances[cls] = np.var(X_cls, axis=0) + 1e-9
            self.priors[cls] = X_cls.shape[0] / X.shape[0]

    def predict(self, X):
        posteriors = []
        for cls in self.classes:
            mean, var = self.means[cls], self.variances[cls]
            prior = np.log(self.priors[cls])
            likelihood = -0.5 * np.sum(np.log(2 * np.pi * var)) - \
                         0.5 * np.sum(((X - mean) ** 2) / var, axis
                            =1)
            posteriors.append(prior + likelihood)
        return self.classes[np.argmax(posteriors, axis=0)]
```

Listing 6: Custom Gaussian Naive Bayes Classifier

The Scikit-learn implementation gave the same results, confirming that the custom implementation was correct.

7 Evaluation of Decision Tree Model

7.1 Evaluation Metrics and Results

The Decision Tree model was tested with different depths to explore the trade-offs between complexity, generalization, and overfitting. The baseline model, set to a maximum depth of 50, achieved an accuracy of 0.58, precision of 0.59, recall of 0.59, and an F1 score of 0.58. These results show that the model can classify the PCA-reduced CIFAR-10 data fairly well, with balanced precision and recall. The confusion matrix for this baseline setup revealed frequent misclassifications in certain classes, as shown below:

Confusion Matrix with Max Depth = 50:

$$\begin{bmatrix}
56 & 4 & 8 & 4 & 3 & 0 & 1 & 1 & 19 & 4 \\
7 & 68 & 0 & 4 & 0 & 1 & 0 & 0 & 7 & 13 \\
3 & 1 & 42 & 11 & 8 & 10 & 17 & 7 & 1 & 0 \\
1 & 0 & 9 & 44 & 3 & 23 & 13 & 4 & 1 & 2 \\
5 & 1 & 12 & 6 & 45 & 7 & 4 & 18 & 0 & 2 \\
0 & 0 & 8 & 14 & 6 & 60 & 6 & 4 & 2 & 0 \\
1 & 0 & 11 & 5 & 3 & 2 & 76 & 2 & 0 & 0 \\
1 & 1 & 6 & 11 & 10 & 14 & 0 & 55 & 1 & 1 \\
22 & 3 & 2 & 0 & 0 & 0 & 2 & 1 & 65 & 5 \\
5 & 8 & 0 & 1 & 1 & 0 & 0 & 1 & 10 & 74
\end{bmatrix}$$

I tested different maximum depths for the Decision Tree, setting values of 10, 20, 30, 40, and 50. These experiments provided valuable insights into how tree depth affects generalization and model performance. A depth of 10 gave the best results, with accuracy of 0.61, precision of 0.62, recall of 0.61, and F1 score of 0.61. This suggests that limiting the depth helped prevent overfitting while still allowing the model to generalize well. However, deeper trees showed little improvement, with similar metrics and confusion matrices across these configurations.

Impact of Depth on Model Behavior Reducing the maximum depth of the Decision Tree helped prevent overfitting, as shown by the improved metrics at depth 10. At this depth, the model avoided memorizing noise or outliers in the training data, which allowed it to generalize better. On the other hand, deeper configurations captured more complex patterns in the training data, but this led to overfitting and resulted in performance metrics that leveled off.

The depth experiments highlighted the trade-offs in Decision Tree models. Shallow trees, with limited depth, acted as a form of regularization, promoting better generalization but losing the ability to capture complex decision boundaries. This is clear from the confusion matrix of the depth-10 configuration:

Confusion Matrix with Depth = 10:

$$\begin{bmatrix}
55 & 7 & 7 & 6 & 3 & 1 & 0 & 0 & 17 & 4 \\
6 & 70 & 0 & 3 & 0 & 2 & 2 & 0 & 5 & 12 \\
4 & 1 & 46 & 19 & 4 & 10 & 11 & 4 & 1 & 0 \\
1 & 0 & 9 & 61 & 4 & 17 & 6 & 1 & 0 & 1 \\
6 & 0 & 5 & 6 & 57 & 10 & 3 & 12 & 0 & 1 \\
0 & 0 & 6 & 25 & 4 & 57 & 3 & 3 & 2 & 0 \\
3 & 0 & 11 & 11 & 4 & 1 & 69 & 1 & 0 & 0 \\
1 & 1 & 4 & 13 & 11 & 13 & 0 & 55 & 1 & 1 \\
18 & 5 & 2 & 1 & 0 & 0 & 1 & 1 & 68 & 4 \\
7 & 9 & 0 & 2 & 1 & 0 & 0 & 1 & 8 & 72
\end{bmatrix}$$

In contrast, deeper trees were able to capture more detailed patterns but struggled with generalization. This was reflected in the confusion matrix, where frequent misclassifications across classes were observed.

7.2 Insights into Class-Wise Performance

Well-Recognized Classes: The Decision Tree model performed well for classes with clear and well-separated features in the PCA-reduced space. For example, class 9 consistently achieved high recall values across all configurations, including the default depth of 50, where it recorded a recall of 0.91. This success is due to the model's ability to make axis-aligned splits, effectively capturing clear boundaries in the feature space. Class 9's distinctiveness, likely from unique textures and edges in the CIFAR-10 dataset, helped the Decision Tree classify it accurately. Similarly, class 0 as vehicles showed high precision and recall, with a recall of 0.90 at depth 10, due to the clear separability of its features. These results highlight the Decision Tree's effectiveness in handling classes with linear boundaries and distinct representations.

Frequently Confused Classes: Despite its strengths, the Decision Tree struggled with classes that had overlapping feature distributions or similar attributes. Classes 3 as vehicles and 7 as structures were often misclassified, as shown by the confusion matrices across all depth configurations. For example, at the default depth of 50, the recall for class 3 dropped to 0.72, while class 7 saw a similar decline to 0.73. This confusion is due to the limitations of axis-aligned splits, which cannot capture complex, non-linear relationships in the data. The dimensionality reduction through PCA likely contributed to these misclassifications by compressing subtle but important differences between the classes. These challenges demonstrate the limitations of Decision Trees when dealing with high-dimensional and complex datasets like CIFAR-10.

7.3 Primary Findings

The Decision Tree experiments highlighted the critical relationship between depth, generalization, and model capacity. The shallow configuration with a depth of 10 achieved the best balance between generalization and accuracy, with an accuracy of 0.61, precision of 0.62, recall of 0.61, and F1-score of 0.61. This configuration effectively reduced overfitting while maintaining strong performance, especially for well-recognized classes like class 9, which achieved a recall of 0.92.

In contrast, deeper configurations, like the default depth of 50, captured more complex patterns but were prone to overfitting, as seen in increased misclassifications for overlapping classes. For example, accuracy dropped to 0.58, while both precision and recall stagnated at 0.59. These results suggest that deeper trees memorized training data instead of generalizing to new samples. Experiments with depths of 20, 30, and 40 showed incremental improvements in recall for some classes, but also introduced instability in others, illustrating the trade-off between depth and generalization.

Class-wise performance further emphasized the Decision Tree's reliance on axis-aligned splits. While this method worked well for distinct classes like class 0 and class 9, it struggled with more subtle distinctions, as seen in the ongoing confusion between classes 3 and 7. These findings underscore the importance of finding the right tree depth to optimize feature extraction without overfitting and highlight the Decision Tree's limitations in capturing non-linear feature interactions.

7.4 Conclusions

The Decision Tree model served as a strong and interpretable baseline for the CIFAR-10 classification task, performing well on classes with clear, linear boundaries. However, its inability to handle high-dimensional and non-linear data highlights the need to explore alternative models for more complex datasets. The depth experiments revealed a key trade-off: shallow configurations generalized well but struggled with intricate patterns, while deeper trees captured complex features at the cost of overfitting and misclassifications.

Overall, the Decision Tree provided valuable insights into how depth and linear splitting affect classification. Its performance emphasized the need for more advanced models, like MLPs and CNNs, which are better equipped to handle overlapping and non-linear feature distributions. These findings reaffirm the Decision Tree's role as a useful starting point for classification tasks while highlighting its limitations with complex datasets like CIFAR-10.

8 Evaluation of the Multilayer Perceptron

The evaluation of the MLP involved three configurations: the main model, a deeper model, and a smaller model. Each configuration was tested on PCA-reduced CIFAR-10 data to assess its performance using metrics such as accuracy, precision, recall, and F1-score. The results from these tests offer insights into how changes in the architecture affected the model's performance.

8.1 Metrics for the MLP Models

Main Model: The main MLP model achieved an accuracy of 0.82, with a precision of 0.83, recall of 0.82, and an F1-score of 0.83. This strong performance shows that the model is effective at identifying class-specific patterns and minimizing both false positives and false negatives. The confusion matrix below shows the distribution of correct and incorrect predictions:

$$\begin{bmatrix}
80 & 0 & 5 & 1 & 0 & 0 & 2 & 0 & 8 & 4 \\
3 & 86 & 0 & 2 & 0 & 0 & 1 & 0 & 2 & 6 \\
4 & 0 & 76 & 4 & 3 & 4 & 6 & 1 & 2 & 0 \\
0 & 0 & 3 & 76 & 3 & 8 & 9 & 0 & 1 & 0 \\
2 & 0 & 2 & 7 & 80 & 3 & 0 & 6 & 0 & 0 \\
0 & 0 & 6 & 14 & 2 & 73 & 3 & 1 & 1 & 0 \\
1 & 0 & 0 & 3 & 5 & 1 & 89 & 1 & 0 & 0 \\
1 & 0 & 1 & 5 & 7 & 4 & 0 & 82 & 0 & 0 \\
5 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 92 & 0 \\
2 & 3 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 91
\end{bmatrix}$$

The confusion matrix shows that class 9 was recognized with high accuracy, while overlapping features between classes 3 and 7 caused occasional misclassifications.

Deeper Model: The deeper MLP model achieved slightly better metrics, with an accuracy of 0.83, precision of 0.83, recall of 0.83, and an F1-score of 0.83. These results suggest that the added hidden layer enhanced the model's ability to capture more complex relationships in the dataset. The confusion matrix for the deeper model is shown below:

$$\begin{bmatrix}
81 & 0 & 5 & 2 & 1 & 0 & 0 & 0 & 8 & 3 \\
2 & 87 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 7 \\
4 & 0 & 80 & 5 & 2 & 3 & 4 & 1 & 1 & 0 \\
0 & 0 & 7 & 70 & 5 & 9 & 7 & 1 & 1 & 0 \\
2 & 0 & 5 & 4 & 78 & 3 & 1 & 7 & 0 & 0 \\
0 & 0 & 5 & 15 & 3 & 73 & 2 & 1 & 1 & 0 \\
1 & 0 & 5 & 1 & 2 & 2 & 88 & 1 & 0 & 0 \\
1 & 0 & 1 & 3 & 6 & 2 & 0 & 87 & 0 & 0 \\
4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 93 & 1 \\
2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 4 & 91
\end{bmatrix}$$

While the deeper model showed better overall performance, the confusion matrix reveals an increase in false positives for classes with overlapping features, like classes 3 and 4. This suggests that the model may be overfitting to specific patterns in the data.

Smaller Model: The smaller MLP model, designed for computational efficiency, achieved an accuracy of 0.82, with precision, recall, and F1-score all at 0.82. The confusion matrix for the smaller model is shown below:

$$\begin{bmatrix}
81 & 0 & 7 & 0 & 1 & 0 & 0 & 1 & 9 & 1 \\
2 & 85 & 1 & 1 & 0 & 0 & 1 & 0 & 3 & 7 \\
4 & 0 & 77 & 7 & 4 & 1 & 6 & 1 & 0 & 0 \\
0 & 0 & 3 & 72 & 5 & 12 & 7 & 1 & 0 & 0 \\
2 & 0 & 2 & 6 & 80 & 2 & 2 & 5 & 1 & 0 \\
0 & 0 & 4 & 19 & 2 & 70 & 2 & 2 & 1 & 0 \\
1 & 0 & 1 & 2 & 3 & 2 & 90 & 1 & 0 & 0 \\
0 & 0 & 2 & 5 & 9 & 6 & 0 & 78 & 0 & 0 \\
5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 93 & 0 \\
2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 4 & 91
\end{bmatrix}$$

This model showed consistent performance across most classes but was more likely to misclassify nuanced categories, such as classes 1 and 7.

8.2 Insights and Implications

The MLP models reveal key trade-offs between complexity, performance, and computational efficiency. The main model provided a strong baseline, balancing the ability to represent data with generalization. The deeper model captured more intricate patterns, as shown by higher metrics, but its slight overfitting reduced generalization in overlapping classes. On the other hand, the smaller model showed consistent performance across classes but struggled with nuanced features due to its limited capacity.

Class 9 consistently had the highest recognition across all models, thanks to its distinct features that reduced ambiguity. Classes with overlapping characteristics, like 3 and 7, had higher misclassification rates, highlighting the difficulty of distinguishing similar patterns in high-dimensional feature spaces.

These findings emphasize the importance of adjusting MLP architectures to the complexity of the dataset. While deeper networks provide better feature representation, they require careful regularization to prevent overfitting. Smaller networks, though more computationally efficient, may underfit complex data. Ultimately, the deeper model proved to be the best-performing version, showing the advantages of additional depth for learning features.

8.3 Primary Findings

The evaluation of the MLP models provided insights into how architectural changes affect classification performance. Each model—main, deeper, and smaller—had distinct characteristics that influenced its ability to generalize and classify CIFAR-10 images.

The main MLP model, with three fully connected layers, achieved an accuracy of 0.82, precision of 0.83, recall of 0.82, and an F1-score of 0.83. It performed well on well-defined classes like class 9 but showed misclassifications in overlapping classes like 3 and 7, indicating a good balance between capacity and generalization.

The deeper MLP model, which added an extra hidden layer, slightly outperformed the main model, achieving an accuracy of 0.83 and matching precision, recall, and F1-score of 0.83. While it captured more intricate patterns, it showed minor overfitting, especially in classes like 3 and 5, where false positives increased.

The smaller MLP model, with reduced hidden layers, had similar metrics to the main model, but struggled with overlapping classes like 4 and 7. Its simpler architecture prioritized generalization, though its reduced capacity limited detailed feature extraction. Its computational efficiency makes it ideal for resource-constrained tasks.

Class-wise analysis showed that all models successfully recognized well-defined classes like class 9. However, overlapping classes such as 3, 5, and 7 were more challenging, highlighting the need for regularization or data augmentation to improve separability.

8.4 Conclusion

The evaluation highlighted the importance of architectural choices in classification performance. The deeper model achieved the highest accuracy of 0.83, using its increased depth to capture complex relationships in the data. However, its slight overfitting showed the trade-off between capacity and generalization.

The main model served as a strong baseline, achieving similar precision and recall while maintaining solid performance across various classes. It effectively balanced computational cost and accuracy, making it a reliable choice for tasks with moderate complexity.

The smaller model focused on computational efficiency, maintaining accuracy and recall similar to the main model but struggling with nuanced class distinctions. Its reduced capacity led to higher misclassification rates in overlapping classes, making it more suitable for scenarios with limited computational resources, where some accuracy trade-offs are acceptable.

These findings emphasize the need to tailor neural network architectures to the task's demands. For complex datasets like CIFAR-10, deeper architectures may provide significant advantages. However, to avoid overfitting, future work could explore regularization techniques like dropout or weight decay, along with data augmentation. The results also stress the importance of balancing depth and computational efficiency when designing neural networks for real-world applications.

9 Evaluation of the VGG11 Models

The VGG11 CNN models were evaluated to understand how different architectural variations impacted their ability to classify CIFAR-10 images. Three configurations were tested: the main model, a deeper model with additional convolutional layers, and a smaller model with fewer convolutional layers or modified kernel sizes. Each configuration was thoroughly tested using metrics like accuracy, precision, recall, and F1-score, along with confusion matrix analysis to examine class-specific performance.

9.1 Metrics for the VGG11 Models

Main Model: The baseline configuration of the VGG11 model achieved balanced performance metrics:

- **Accuracy:** 0.82
- **Precision:** 0.82
- **Recall:** 0.82
- **F1 Score:** 0.82

These metrics show that the main model was effective in handling general classification tasks, successfully capturing the key features in the CIFAR-10 dataset.

The confusion matrix for the main model is:

$$\begin{bmatrix} 840 & 4 & 34 & 23 & 13 & 4 & 7 & 13 & 34 & 28 \\ 17 & 834 & 1 & 2 & 2 & 1 & 2 & 2 & 13 & 126 \\ 43 & 2 & 686 & 56 & 70 & 62 & 42 & 29 & 0 & 10 \\ 10 & 3 & 23 & 667 & 37 & 153 & 35 & 36 & 6 & 30 \\ 6 & 2 & 33 & 53 & 788 & 25 & 18 & 70 & 2 & 3 \\ 3 & 1 & 7 & 128 & 25 & 781 & 15 & 29 & 2 & 9 \\ 4 & 2 & 15 & 50 & 26 & 8 & 881 & 3 & 2 & 9 \\ 3 & 0 & 6 & 28 & 16 & 34 & 5 & 898 & 0 & 10 \\ 43 & 8 & 8 & 11 & 4 & 1 & 4 & 8 & 881 & 32 \\ 19 & 11 & 2 & 3 & 1 & 0 & 1 & 6 & 9 & 948 \end{bmatrix}$$

The confusion matrix reveals that the main model performed exceptionally well on class 10, with 948 correct predictions out of 1000, resulting in a recall of 0.95 for this class. However, it struggled with class 3, misclassifying many instances as classes 5 and 6, which suggests difficulty in distinguishing features among these classes.

Deeper Model: The deeper VGG11 model, augmented with additional convolutional layers, achieved improved metrics:

- **Accuracy:** 0.83
- **Precision:** 0.83
- **Recall:** 0.83
- **F1 Score:** 0.83

The slight increase in metrics indicates that the deeper model benefited from enhanced capacity to capture complex patterns.

The confusion matrix for the deeper model is:

$$\begin{bmatrix}
855 & 14 & 10 & 16 & 18 & 0 & 2 & 7 & 49 & 29 \\
7 & 919 & 2 & 6 & 2 & 0 & 3 & 3 & 21 & 37 \\
70 & 6 & 715 & 75 & 74 & 14 & 19 & 15 & 10 & 2 \\
22 & 3 & 41 & 755 & 56 & 57 & 32 & 11 & 12 & 11 \\
11 & 3 & 28 & 41 & 860 & 12 & 15 & 23 & 5 & 2 \\
9 & 7 & 22 & 246 & 49 & 618 & 16 & 25 & 2 & 6 \\
6 & 3 & 29 & 59 & 24 & 6 & 868 & 1 & 1 & 3 \\
10 & 2 & 13 & 44 & 31 & 20 & 2 & 868 & 1 & 9 \\
28 & 13 & 10 & 11 & 2 & 0 & 2 & 3 & 910 & 21 \\
12 & 53 & 4 & 8 & 4 & 3 & 5 & 4 & 16 & 891
\end{bmatrix}$$

The deeper model showed significant improvement in classifying class 2, increasing correct predictions from 686 to 715, and reducing misclassifications to classes 4 and 5. However, an increase in misclassifications for class 6 suggests that the model might be overfitting to certain patterns.

Smaller Model: The smaller VGG11 model, with reduced layers or altered kernel sizes, recorded slightly lower performance metrics:

- **Accuracy:** 0.80
- **Precision:** 0.81
- **Recall:** 0.80
- **F1 Score:** 0.80

This decline reflects the model's reduced ability to capture detailed features because of its simplified architecture.

The confusion matrix for the smaller model is:

$$\begin{bmatrix}
911 & 7 & 26 & 11 & 14 & 4 & 3 & 3 & 17 & 4 \\
32 & 889 & 3 & 2 & 2 & 0 & 6 & 0 & 34 & 32 \\
69 & 0 & 734 & 18 & 84 & 32 & 45 & 14 & 3 & 1 \\
25 & 2 & 76 & 491 & 126 & 121 & 100 & 25 & 20 & 14 \\
13 & 1 & 28 & 9 & 899 & 10 & 19 & 16 & 4 & 1 \\
15 & 4 & 53 & 77 & 96 & 677 & 35 & 26 & 9 & 8 \\
7 & 0 & 33 & 16 & 34 & 6 & 887 & 6 & 7 & 4 \\
19 & 1 & 26 & 12 & 75 & 21 & 6 & 833 & 3 & 4 \\
90 & 6 & 9 & 4 & 6 & 1 & 5 & 1 & 875 & 3 \\
82 & 74 & 8 & 5 & 5 & 2 & 4 & 4 & 24 & 792
\end{bmatrix}$$

The smaller model struggled with class 4, where correct predictions dropped to 491, compared to 667 in the main model, and misclassifications to classes 3 and 5 increased. This suggests that the reduced capacity limited the model's ability to capture important features for certain classes.

9.2 Analysis of Model Performance

Well-Recognized Classes: Across all models, class 1 and class 10 were consistently well-recognized. For example, in the deeper model, class 1 achieved 919 correct predictions out of 1000, with a recall of 0.92. This success is due to the distinct visual features of these classes, which the models could easily learn and distinguish.

Frequently Confused Classes: Classes with similar visual characteristics, like classes 3, 5, and 6, were often confused. The main model misclassified 153 instances of class 4 as class 6, suggesting difficulty in distinguishing features such as texture and shape. The deeper model reduced some of these misclassifications but introduced others, showing the complex relationship between depth and feature extraction.

9.3 Impact of Depth and Kernel Size

Well-Recognized Classes: Across all models, class 1 and class 10 were consistently well-recognized. For example, in the deeper model, class 1 achieved 919 correct predictions out of 1000, with a recall of 0.92. This success is due to the distinct visual features of these classes, which the models could easily learn and distinguish.

Frequently Confused Classes: Classes with similar visual characteristics, like classes 3, 5, and 6, were often confused. The main model misclassified 153 instances of class 4 as class 6, suggesting difficulty in distinguishing features such as texture and shape. The deeper model reduced some of these misclassifications but introduced others, showing the complex relationship between depth and feature extraction.

9.4 Primary Findings

The evaluation of the VGG11 models led to the following key findings:

Deeper Model Superiority: The deeper VGG11 model had the highest overall performance, with an accuracy of 0.83. It improved classification in challenging classes, such as increasing correct predictions for class 3 from 686 to 715, and reducing misclassifications in classes 2 and 4. The added layers allowed the model to capture more abstract and complex features, enhancing its ability to differentiate between classes.

Overfitting Concerns: While the deeper model showed better performance, it also displayed signs of overfitting. Misclassifications of class 5 increased, with 246 instances incorrectly predicted as class 4, compared to 128 in the main model. This suggests that while depth improves feature learning, it requires stronger regularization techniques to prevent overfitting.

Main Model Balance: The main model maintained a good balance between performance and generalization. It achieved a reliable accuracy of 0.82

without significant overfitting. The confusion matrix showed consistent performance across most classes, making it a solid baseline for further experimentation.

Smaller Model Limitations: The smaller model had reduced capacity, resulting in a drop in accuracy to 0.80. It struggled with classes requiring more detailed feature recognition, such as class 4, where correct predictions dropped from 667 to 491. This suggests that insufficient depth or smaller kernels limited the model's ability to capture essential features.

Class-Specific Challenges: All models faced difficulties with classes that had subtle visual differences or significant intra-class variation. High misclassification rates among classes 3, 5, and 6 emphasized the need for architectures that can better capture and differentiate complex features.

9.5 Conclusion

The evaluation of the VGG11 models shows how important architectural choices are for classification performance. The deeper model's higher accuracy shows the benefits of adding depth for complex datasets like CIFAR-10. However, the risk of overfitting means we need regularization strategies, like dropout or data augmentation, to improve generalization.

The main model provides a strong baseline, balancing performance and computational efficiency. It gives consistent accuracy without overfitting, making it good for tasks where resources are limited or overfitting must be controlled.

The smaller model, while faster to run, struggled to capture complex patterns because of its reduced depth or smaller kernels. This shows the trade-off between model size and recognition ability, highlighting the need for enough complexity to handle high-resolution images.

The results suggest that for datasets with complex features and classes that are hard to tell apart, deeper CNNs work better. But we must be careful to avoid overfitting so the model works well on new data. Future work could look into advanced regularization techniques and fine-tuning kernel sizes to improve performance even more.

10 Comparison of Models

The evaluation of Naive Bayes, Decision Tree, MLP, and VGG11 models on the CIFAR-10 dataset helps us understand their strengths, weaknesses, and suitability for different classification tasks. Each model has its own advantages, with performance being affected by factors like dataset complexity, model design, and how the experiments were set up.

10.1 Overall Performance Across Metrics

The VGG11 model consistently outperformed the others in accuracy, precision, recall, and F1-score. The baseline VGG11 achieved an accuracy of 0.82, while the deeper variant improved it to 0.83. The MLP models also performed well, with the baseline and deeper MLPs achieving accuracy scores of 0.82 and 0.83, respectively. In contrast, the Decision Tree and Naive Bayes models lagged behind, with the best Decision Tree configuration with depth 10 reaching an accuracy of 0.61, and Naive Bayes achieving 0.79. These results show that neural network models, especially the CNN-based VGG11, are better at capturing complex, non-linear patterns in image data.

10.2 Insights into Specific Scenarios

Handling Overlapping Classes: The VGG11 and MLP models showed better performance in distinguishing overlapping classes, such as classes 3 and 7, where the Decision Tree and Naive Bayes models had difficulty. For example, in the Decision Tree's depth 50 configuration, class 3 had a recall of only 0.72 due to significant confusion between these classes. In contrast, the neural network models used non-linear transformations and hierarchical feature extraction to capture subtle differences more effectively, leading to higher recall and precision for these challenging classes.

Interpretability vs. Performance: While neural networks excel in predictive performance, their complexity reduces interpretability. The Decision Tree, with its axis-aligned splits and clear decision-making process, offered valuable insights into class boundaries. For example, it recognized classes with distinct features, like class 9, where it achieved a recall of 0.91 in the depth 10 configuration. Naive Bayes, although less accurate, provided simplicity and robustness, as shown by its balanced recall and precision of 0.79.

Impact of Depth and Complexity: Experiments with varying depth in Decision Trees and MLPs showed important trade-offs. The deeper Decision Tree model with depth 50 overfitted, memorizing training patterns and misclassifying test samples, as its precision and recall stagnated at 0.59. On the other hand, the deeper MLP with three hidden layers balanced capacity and generalization well, achieving an accuracy of 0.83. For VGG11, increasing depth improved feature extraction but also increased the risk of overfitting, which was managed using batch normalization and dropout.

Efficiency and Scalability: The Naive Bayes model was the most computationally efficient due to its simple probabilistic framework. It processed PCA-reduced CIFAR-10 data quickly, making it ideal for rapid prototyping. The Decision Tree also trained efficiently but struggled with high-dimensional data, where axis-aligned splits were not enough. In contrast, the MLP and VGG11 models required more computational resources, especially during training, but their scalability and performance made them essential for handling complex datasets.

10.3 Suitability for Specific Applications

Naive Bayes: It's ideal for quick baseline evaluations and tasks with well-separated feature distributions. Its robustness in handling noisy data ensures reliable performance, especially in scenarios with limited computational resources.

Decision Tree: The Decision Tree is best suited for applications where interpretability is crucial, as it clearly shows decision boundaries. However, its performance drops for high-dimensional and overlapping datasets, as seen with CIFAR-10.

MLP: The MLP excels in datasets with moderate complexity, balancing performance and computational cost. The deeper variant offered better representational power while maintaining generalization, making it a versatile choice for mid-level image classification tasks.

VGG11: The VGG11 model is unmatched in handling complex, high-dimensional datasets and is the preferred choice for advanced image classification tasks. Its ability to capture hierarchical features ensures superior accuracy, though it comes with higher computational demands and reduced interpretability.

10.4 Primary Findings and Key Takeaways

The comparative evaluation highlights the trade-offs involved in model selection. VGG11 and deeper MLP variants achieved the highest accuracy, precision, and recall, demonstrating their ability to capture complex patterns in CIFAR-10 images. In terms of generalization, shallow Decision Trees and smaller MLP models performed well but with reduced representational power. When it comes to computational efficiency, Naive Bayes proved to be the most efficient, making it ideal for quick evaluations, while neural networks required more resources but delivered superior performance. Lastly, all models struggled with overlapping classes, but VGG11 and MLP models were able to mitigate this issue through advanced feature extraction techniques.

10.5 Conclusions

The choice of model ultimately depends on the task's complexity, interpretability needs, and computational constraints. For complex datasets like CIFAR-10, neural networks, especially CNNs like VGG11, offer unmatched performance. However, simpler models like Naive Bayes and Decision Trees are still valuable for quick evaluations and tasks that prioritize interpretability. These findings highlight the importance of aligning model selection with specific application requirements to ensure a balance between accuracy, efficiency, and complexity.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Pearson, 2010. [Online]. Available: https://icourse.club/uploads/files/3a4f5c1d49ceb1dfe137c645882848bf5e0f6d09.pdf

[2] T. Mitchell, *Machine Learning*, McGraw Hill, 1997. [Online]. Available: https://www.cin.ufpe.br/~cavmj/Machine%20-%20Learning%20-%20Tom%20Mitchell.pdf

[3] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Advances in Neural Information Processing Systems (NIPS)*, 2001. [Online]. Available: https://direct.mit.edu/books/edited-volume/2485/Advances-in-Neural-Information-Processing-Systems

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Routledge, 2017. [Online]. Available: https://doi.org/10.1201/9781315139470

[5] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986. [Online]. Available: https://doi.org/10.1023/A:1022643204877

[6] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, pp. 78–87, 2012. [Online]. Available: https://doi.org/10.1145/2347736.2347755

[7] Scikit-learn documentation: Decision Trees. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html

[8] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010. [Online]. Available: https://icml.cc/Conferences/2010/papers/432.pdf

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer, 2009. [Online]. Available: https://link.springer.com/book/10.1007/978-0-387-84858-7

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint*, 2015. [Online]. Available: `https://arxiv.org/abs/1502.03167`

[14] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," *Proc. 27th Int. Conf. Mach. Learn.*, 2010. [Online]. Available: `https://icml.cc/Conferences/2010/papers/432.pdf`

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015. [Online]. Available: `https://doi.org/10.1038/nature14539`

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proc. IEEE CVPR*, 2016. [Online]. Available: `https://arxiv.org/abs/1512.03385`

[17] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint*, 2016. [Online]. Available: `https://arxiv.org/abs/1609.04747`

[18] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006. [Online]. Available: `https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf`

[19] Scikit-learn Documentation: Gaussian Naive Bayes. [Online]. Available: `https://scikit-learn.org/stable/modules/naive_bayes.html`