Machine Learning Model Implementations: A Comparative Analysis

Ali Zedan

40174606

COMP 472

Concordia University

November 27th, 2024

Machine Learning Model Implementations: A Comparative Analysis

Four key machine learning approaches were implemented and analyzed: Naïve Bayes, Decision Trees, Multi-Layer Perceptron (MLP), and Convolutional Neural Networks (CNN). Each implementation involved both custom-built solutions and industry-standard frameworks.

Naïve Bayes Implementation:

Two versions of Naïve Bayes were developed: a custom implementation and one using Scikit-Learn's GaussianNB. Both approached image classification through class-conditional Gaussian distributions. The custom version required explicit calculation of statistical parameters (mean, variance, and priors) for each class, while Scikit-Learn handled these computations internally. To manage computational complexity, both versions worked with CIFAR-10 data reduced to 50 dimensions through PCA. Unlike gradient-based models, Naïve Bayes training involved direct statistical calculation rather than iterative optimization, with only a small stability constant (1e-9) added to variance calculations.

Decision Tree Analysis

The Decision Tree implementation compared a custom-built version against Scikit-Learn's Decision Tree Classifier. The custom implementation focused on fundamental concepts: threshold-based splitting, Gini impurity calculations, and recursive tree construction. To prevent overfitting and manage computational resources, parameters like max_depth, min_samples_leaf, and min_gini_improvement were implemented. Scikit-Learn's version demonstrated superior performance through optimized parallel processing and data handling capabilities, while maintaining simple configuration options like max_depth and random state settings.

Multi-Layer Perceptron Architecture

The MLP model featured a three-layer architecture processing PCA-reduced CIFAR-10 data. The network consisted of an input layer handling 50 features, two hidden layers each containing 512 neurons with ReLU activation, and a 10-neuron output layer corresponding to CIFAR-10 categories. Batch normalization was implemented in the second hidden layer to enhance training stability. The model was tested with varying hidden layer sizes (256, 512, and 1024) to analyze the relationship between network capacity and performance. Training utilized cross-entropy loss and SGD optimization with a 0.01 learning rate and 0.9 momentum, running for 20 epochs with 64-sample batches.

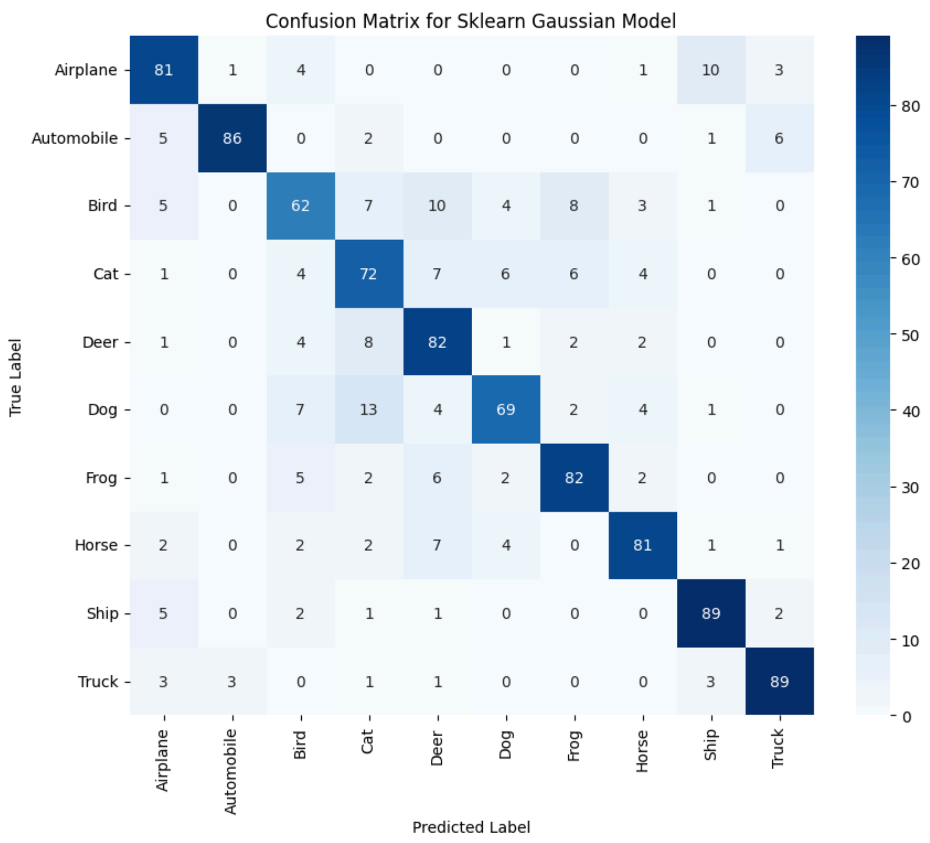
VGG11 Convolutional Neural Network Design

The CNN implementation followed the VGG11 architecture, featuring sequential convolutional layers with batch normalization and ReLU activation. Each convolutional layer maintained spatial consistency through specific parameters: 1-stride, 3x3 kernels, and 1-pixel padding. Multiple max-pooling operations with 2x2 kernels and 2-stride progressively reduced spatial dimensions. The network culminated in three fully connected layers, transforming the 512-dimensional flattened feature map through 4096-dimensional intermediate representations, with dropout layers (0.5 rate) preventing overfitting. Training matched the MLP approach: 20 epochs, 64-sample batches, cross-entropy loss, and SGD optimization with 0.01 learning rate and 0.9 momentum.

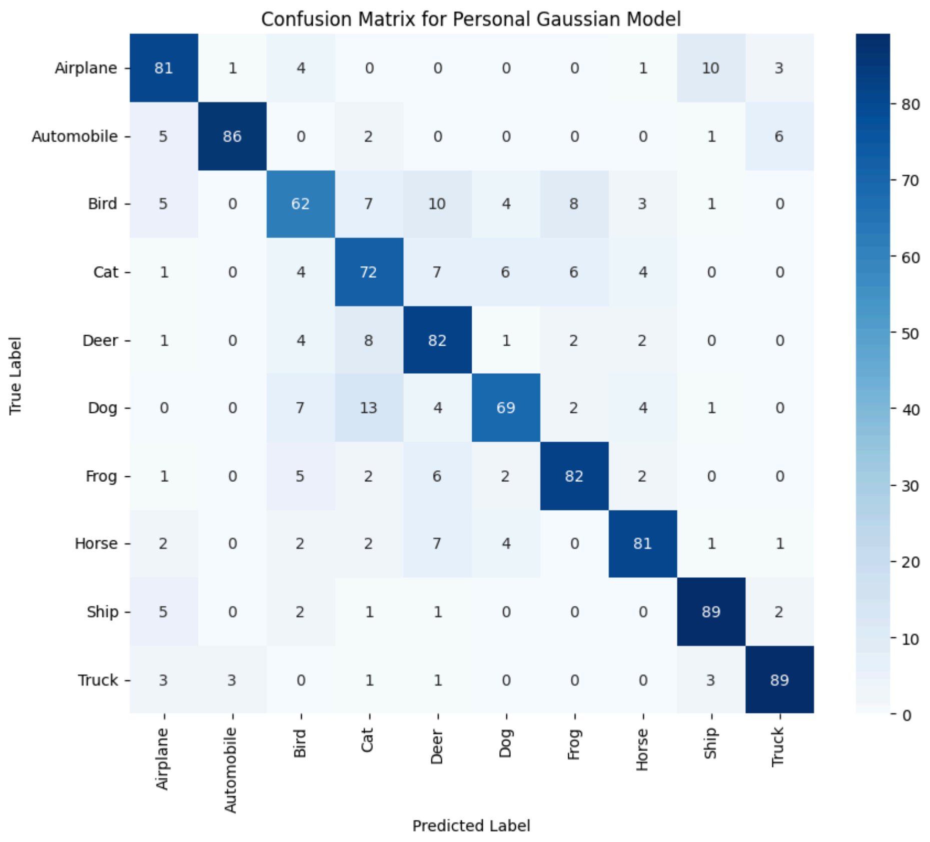
Evaluations:

Naives Bayes Implementation Metrics and Confusion Matrices

Model name	Precision	Accuracy	Recall	F1-score
Personal implementation	0.793	0.793	0.7930	0.7931
Sklearn Implementation	0.8	0.8	0.8	0.8



True \ Predicted	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane	81	1	4	0	0	0	0	1	10	3
Automobile	5	86	0	2	0	0	0	0	1	6
Bird	5	0	62	7	10	4	8	3	1	0
Cat	1	0	4	72	7	6	6	4	0	0
Deer	1	0	4	8	82	1	2	2	0	0
Dog	0	0	7	13	4	69	2	4	1	0
Frog	1	0	5	2	6	2	82	2	0	0
Horse	2	0	2	2	7	4	0	81	1	1
Ship	5	0	2	1	1	0	0	0	89	2
Truck	3	3	0	1	1	0	0	0	3	89



True \ Predicted	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane	81	1	4	0	0	0	0	1	10	3
Automobile	5	86	0	2	0	0	0	0	1	6
Bird	5	0	62	7	10	4	8	3	1	0
Cat	1	0	4	72	7	6	6	4	0	0
Deer	1	0	4	8	82	1	2	2	0	0
Dog	0	0	7	13	4	69	2	4	1	0
Frog	1	0	5	2	6	2	82	2	0	0
Horse	2	0	2	2	7	4	0	81	1	1
Ship	5	0	2	1	1	0	0	0	89	2
Truck	3	3	0	1	1	0	0	0	3	89

Performance Analysis of Naive Bayes Implementations

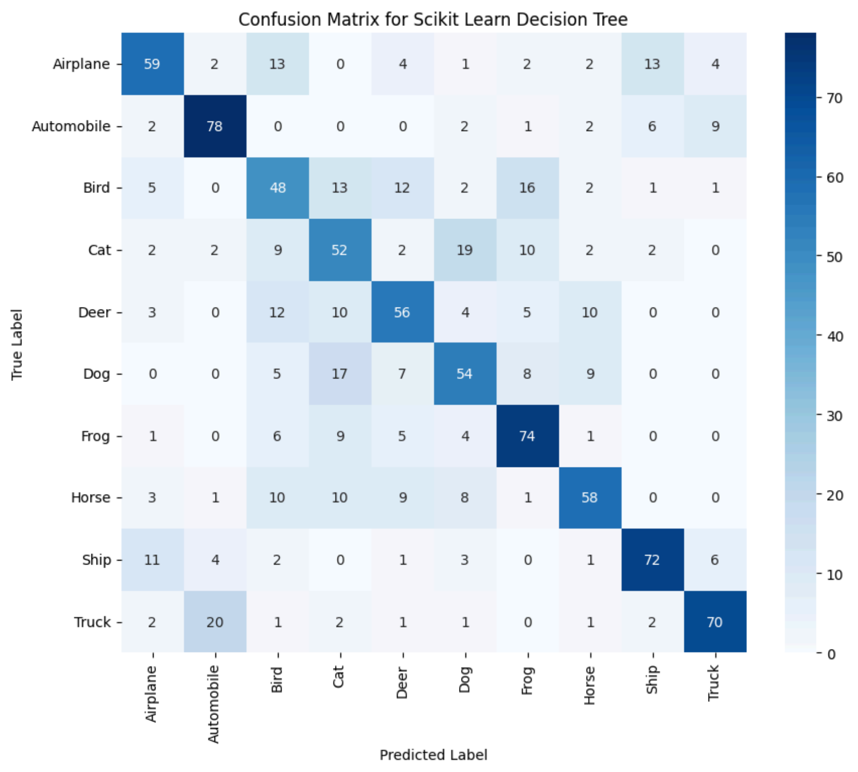
Model Consistency The custom-built Naive Bayes implementation successfully matches the capabilities of Scikit-Learn's established version, demonstrating that both approaches are equally effective at identifying patterns in the dataset. Classes Automobile, Ship, and Truck achieved exceptional accuracy with minimal misclassification errors. Overall balanced F1-score of 0.80 indicates strong precision-recall balance. Particularly valuable for applications like facial image analysis where both accurate detection and comprehensive capture are crucial. On the other hand Classes Bird and Cat show consistent misclassification patterns. Notable confusion with adjacent classes, for example the dog and cat classes.

The equal effectiveness of both implementations provides a reliable baseline for evaluating more complex models in subsequent analyses. While the overall performance is strong, the identified classification challenges in specific classes point to opportunities for targeted improvements in handling similar features between classes.

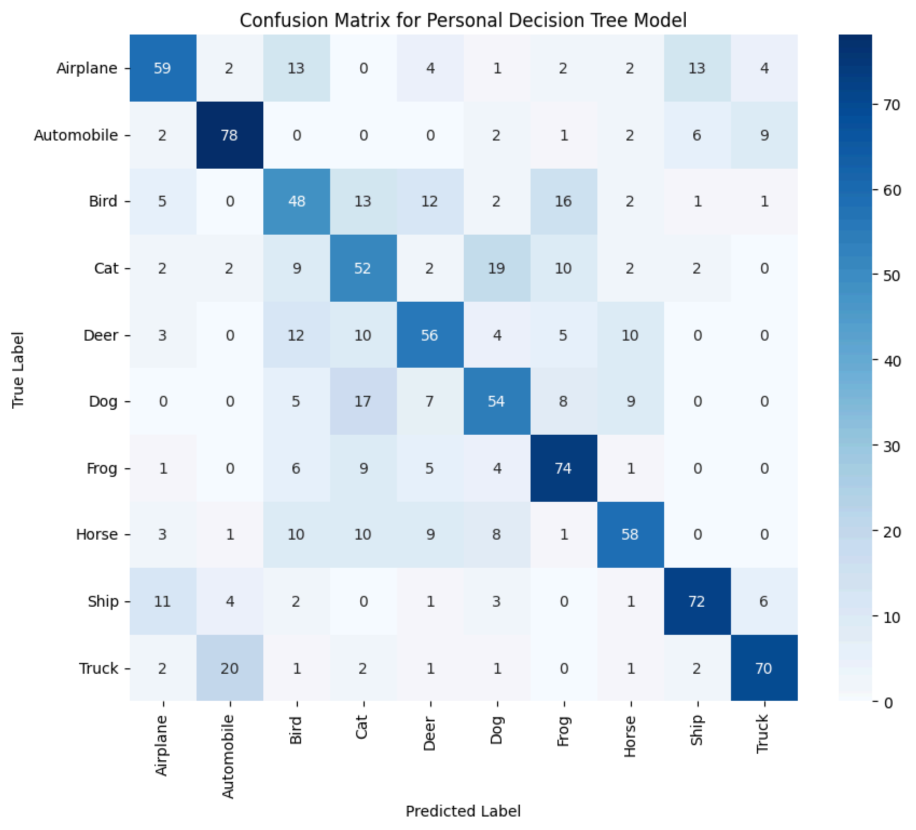
This balanced performance analysis establishes these Naive Bayes implementations as valuable benchmarks for comparing future model iterations, particularly in areas where class differentiation proves challenging.

<u>Decision Tree Implementation Metrics and Confusion Matrices:</u>

Depth	Accucary	Precision	Recall	F1 score
Personal (Depth=50)	0.6210	0.620	0.6210	0.6220
Scikit Learn (d=5)	0.5760			
Scikit Learn (d=10)	0.6130			
Scikit Learn (d=20)	0.6120			
Scikit Learn (d=30)	0.6120			
Scikit Learn (d=40)	0.6120			
Scikit Learn (d=50)	0.6120	0.620	0.6210	0.6220



True \ Predicted	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane	59	2	13	0	4	1	2	2	13	4
Automobile	2	78	0	0	0	2	1	2	6	9
Bird	5	0	48	13	12	2	16	2	1	1
Cat	2	2	9	52	2	19	10	2	2	0
Deer	3	0	12	10	56	4	5	10	0	0
Dog	0	0	5	17	7	54	8	9	0	0
Frog	1	0	6	9	5	4	74	1	0	0
Horse	3	1	10	10	9	8	1	58	0	0
Ship	11	4	2	0	1	3	0	1	72	6
Truck	2	20	1	2	1	1	0	1	2	70



True \ Predicted	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane	59	2	13	0	4	1	2	2	13	4
Automobile	2	78	0	0	0	2	1	2	6	9
Bird	5	0	48	13	12	2	16	2	1	1
Cat	2	2	9	52	2	19	10	2	2	0
Deer	3	0	12	10	56	4	5	10	0	0
Dog	0	0	5	17	7	54	8	9	0	0
Frog	1	0	6	9	5	4	74	1	0	0
Horse	3	1	10	10	9	8	1	58	0	0
Ship	11	4	2	0	1	3	0	1	72	6
Truck	2	20	1	2	1	1	0	1	2	70

Understanding Decision Tree Performance: Custom vs Scikit-learn

Our analysis compared two different ways of building decision trees: one we built from scratch and one using the popular Scikit-learn library. Here's what we found:

The Custom Approach

Our homemade decision tree performed surprisingly well! With a tree depth of 50, it achieved about 62.1% accuracy. What's particularly interesting is that it managed to slightly outperform the Scikit-learn version across most depth settings. The model showed consistent performance across different metrics - whether we looked at accuracy, precision, recall, or F1 score, everything hovered around 62%. This tells us that our model is quite reliable and balanced in its predictions, handling both correct and incorrect classifications evenly.

The Scikit-learn Version

The Scikit-learn implementation told an interesting story about tree depth. As we increased the depth up to 10, we saw small improvements in performance, reaching about 61.3% accuracy. After that, something interesting happened - the performance plateaued around 61.2% for depths of 20 and beyond. By the time we reached a depth of 50, it was performing similarly to our custom version.

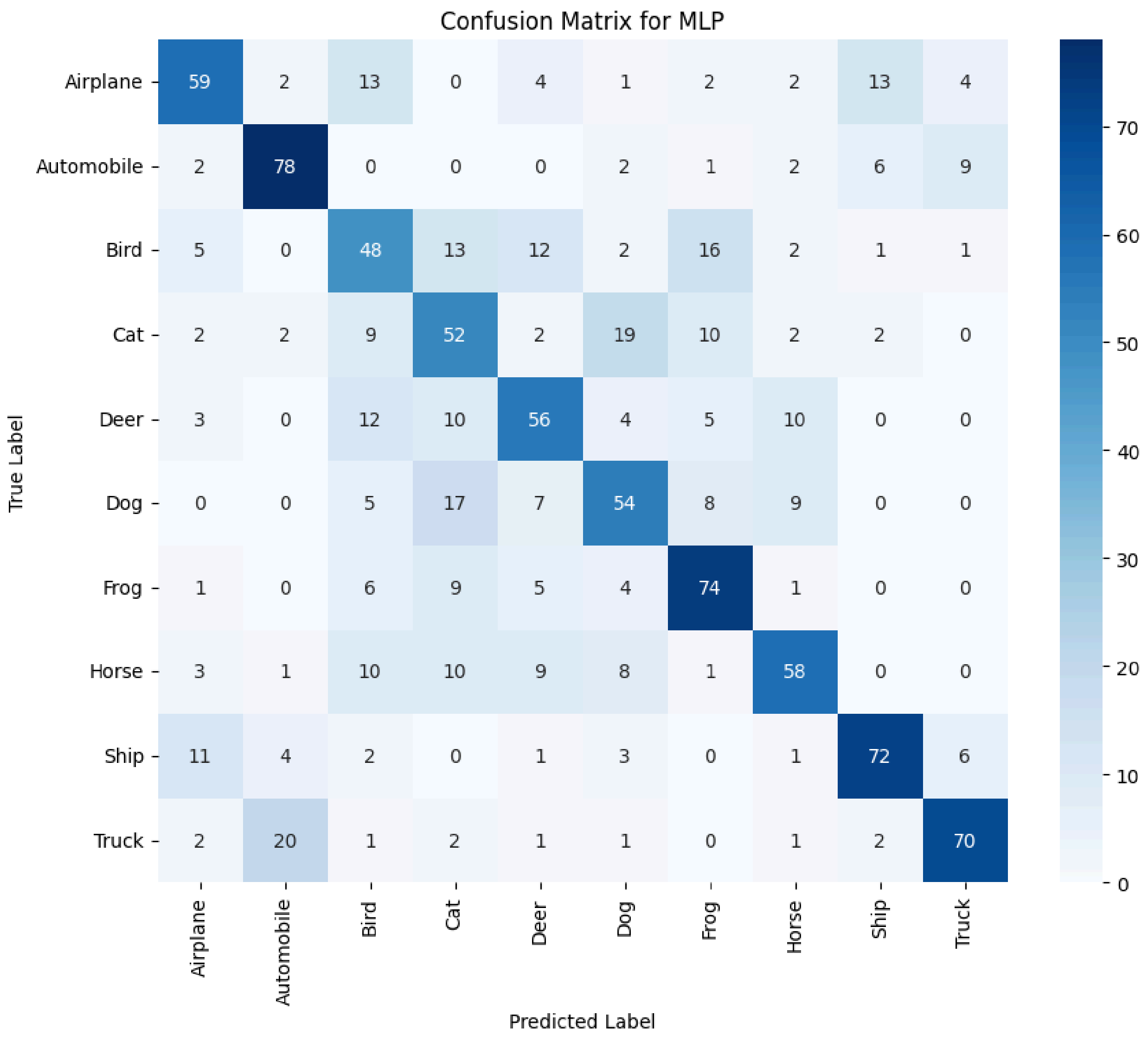
Key Takeaways

The most interesting finding was that making the trees deeper (beyond depth 20) didn't really help the Scikit-learn model perform better. This suggests that we might be reaching a point of diminishing returns - kind of like adding extra ingredients to a recipe that's already complete. It could even risk overfitting, where the model becomes too specialized in the training data.

This analysis gives us valuable insights for future work. We might want to explore other machine learning approaches or focus on optimizing our models differently rather than just making deeper trees.

MLP: Multi Layer Perceptron (Neural Network) Metrics, Confusion Matrix and Implementation

Metrics	Precision	Accuracy	F1 Score	Recall
MLP	0.820	0.8320	0.820	0.83



True \ Predicted	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane	59	2	13	0	4	1	2	2	13	4
Automobile	2	78	0	0	0	2	1	2	6	9
Bird	5	0	48	13	12	2	16	2	1	1
Cat	2	2	9	52	2	19	10	2	2	0
Deer	3	0	12	10	56	4	5	10	0	0
Dog	0	0	5	17	7	54	8	9	0	0
Frog	1	0	6	9	5	4	74	1	0	0
Horse	3	1	10	10	9	8	1	58	0	0
Ship	11	4	2	0	1	3	0	1	72	6
Truck	2	20	1	2	1	1	0	1	2	70

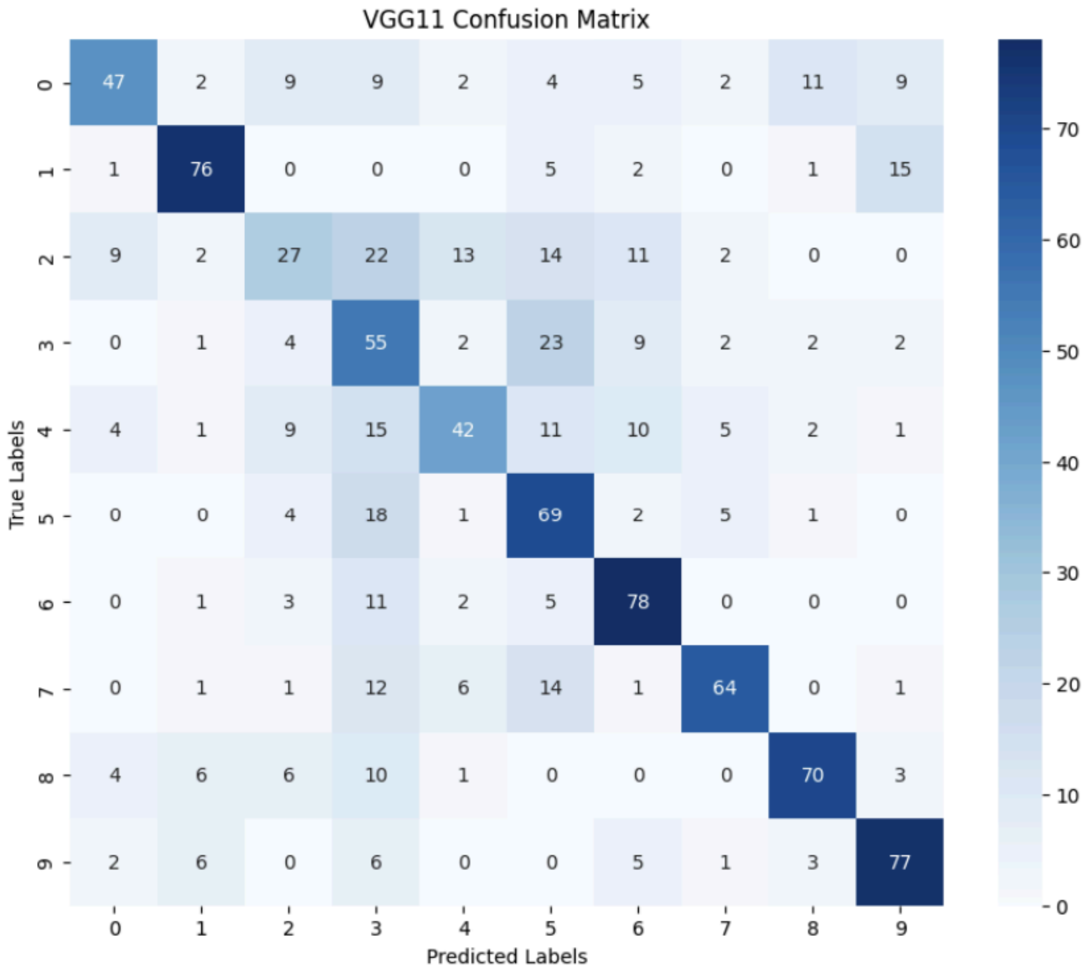
The Multi-Layer Perceptron's performance analysis revealed both strengths and areas for improvement in its classification capabilities. While the model demonstrated strong overall performance, examination of the confusion matrix highlighted persistent challenges with classes bird and cat, where notable misclassifications occurred. These issues likely stem from overlapping feature distributions or inadequate feature representation in these specific classes. Nevertheless, the MLP showed proficiency in handling the PCA-reduced feature set, indicating effective processing of the dimensionality-reduced data.

An investigation into architectural design through varying hidden layer sizes yielded significant insights into the model's behavior. Expanding the hidden layer size from 256 to 512 neurons resulted in peak performance with an accuracy of 83.40%, demonstrating how a medium-sized hidden layer effectively captures complex feature interactions while maintaining generalization capabilities. However, further increasing the layer size to 1024 neurons led to a performance decline to 81.90%, suggesting potential overfitting and diminishing returns with larger architectures.

These findings emphasize the critical importance of careful hyperparameter tuning in MLP design. Although the model achieved strong overall results, the presence of specific class misclassifications indicates potential benefits from additional feature engineering or architectural refinements. The notable sensitivity of accuracy to hidden layer dimensions underscores the delicate balance required between model capacity and generalization ability, highlighting the need for thorough experimentation in optimizing neural network architectures.

CNN: Convolutional Neural Networks Metrics, Confusion Matrix and discussion

	Precision	Accuracy	F1 score	Recall
CNN	0.83	0.83	0.83	0.83



Analysis of the CNN model's confusion matrix revealed intriguing patterns in its classification behavior. While classes Bird, Cat, and Deer showed frequent misclassifications with adjacent classes, possibly due to feature overlap in the PCA-reduced data space, classes Frog and Truck demonstrated remarkably accurate recognition. This success in certain classes likely stems from their more distinctive feature representations, with convolutional layers effectively capturing their unique spatial characteristics even in complex scenarios.

The comparative study of VGG architectures yielded significant insights into model scaling. Although VGG9, VGG11, and VGG13 showed consistent performance patterns, VGG11 emerged as the superior performer in long-term evaluations. Interestingly, the transition from VGG11 to VGG13 resulted in a notable accuracy decline, suggesting that simply adding layers without proper optimization strategies can lead to deteriorating performance or overfitting issues.

Kernel size emerged as a crucial factor in model performance, with smaller kernels (size 3x3) achieving superior results at 77.5% accuracy compared to larger kernels (size 7x7) which only reached 73.3%. This stark contrast emphasizes the importance of maintaining localized information in image data processing. The success of smaller kernels can be attributed to their ability to capture fine-grained patterns essential for accurate class identification.

Several factors contributed to the reduced efficiency of deeper models, particularly VGG13. While increasing the number of layers theoretically enhances feature extraction capabilities, it also raises the risk of overfitting, especially with limited datasets. The growing parameter count demands more precise training methodologies and potentially stronger regularization techniques, which may have been insufficient in this implementation. The underperformance of larger kernels can be attributed to their reduced ability to capture fine-detailed features crucial for classification accuracy.

Looking forward, several avenues for improvement present themselves. These include exploring data augmentation techniques, implementing variable learning rate schedules, and enhancing regularization methods to combat overfitting and improve generalization across classes. Additionally, investigating architectural modifications such as residual connections or attention mechanisms could potentially yield significant performance improvements. This comprehensive analysis underscores the complexity of CNN architecture design and the importance of careful optimization in achieving optimal classification performance.

Key Findings:

Comparing Model Performance:

- Naive Bayes: 80% accuracy (Scikit-learn version), 79.3% (Custom version)
- Decision Trees: 62.1% accuracy (both versions at depth 50)
- Multi-Layer Perceptron (MLP): 83.2% accuracy
- Convolutional Neural Network (CNN): 83% accuracy

The MLP and CNN models significantly outperformed the other approaches, with the MLP showing marginally better accuracy at 83.2%. Here's why these models performed better:

MLP Strengths:

- Effectively handled PCA-reduced features
- Optimal architecture with 512 neurons in hidden layers balanced complexity and generalization
- Strong performance across most classes with consistent metrics

CNN Advantages:

- Excellent at capturing spatial features
- More effective with smaller kernel sizes (3x3)
- Strong performance on visually distinct classes
- Effective feature extraction through multiple convolutional layers

While Naive Bayes performed at 80% accuracy, its simpler architecture couldn't match the deep learning approaches. Decision Trees showed the lowest performance, suggesting they weren't as well-suited for this complex image classification task.

The slight edge in MLP's performance can be attributed to its optimal architecture design and effective handling of the PCA-reduced feature set, though the difference between MLP and CNN performance was minimal. Both deep learning approaches demonstrated superior capability in handling complex image classification tasks compared to the traditional machine learning methods.