Image Classification Project: CIFAR-10 Dataset

by
Andrei Mihaescu
ID: 40139389

An Assignment
submitted in partial fulfillment
of the requirements of
COMP 472
Concordia University
November 25th, 2024

Contents

Model Architectures and Training

Naïve Bayes

Along with one using Scikit-Learn's GaussianNB suite, a custom Naïve Bayes script was created. From the perspective of class-conditional Gaussian distributions, both models solve image classification problems. Whereas Scikit-Learn took advantage of its fast internal processes, the custom solution needed manual calculations for the mean, variance, and priors for every class. Both models trained using the PCA-reduced CIFAR-10 dataset, in which case features were limited to 50 dimensions to ease computations while keeping required information.

Naive Bayes is essentially a set-structured model; so, little modification has been done. Throughout the training procedure free of learning rates, epochs, or iterative techniques, class-specific statistics like mean, variance, and priors were calculated. Naive Bayes does not use mini-batch gradient descent or the Adam optimizer unlike models applying gradient-based optimization methods. Added to the variance values a tiny constant (1^{-9}) assured numerical stability.

Decision Tree

Two decision trees were analyzed: one from scratch and one Scikit-Learning Decision Tree Classifier. Ground up, the first model highlights threshold-based segmentation, Gini impurity, and recursive tree building. Reducing overfit and guaranteeing fair computation costs was achieved using hyperparameters including max_depth, min_samples_leaf, and min_gini_improvement. Recursive tree building continued until either the required depth or homogenous class distribution was reached. This approach eased testing, but it also needed significant computer resources since the ideal splits need to be discovered by methods of characterisation and threshold evaluation.

On the other hand, when applying the enhanced CART approach, the Scikit-Learn software displayed astonishing efficiency. The simple design made perfect accuracy in trees achievable. Max_depth changed the scale of the tree; a continuous random state guaranteed repeatability. Comparatively to the customised solution devoid of these capabilities, Scikit-Learn's improvements in parallel processing and superior data management significantly increased its efficiency and usability.

Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) model was implemented in three layers. An output layer with 10 neurons, two hidden layers with 512 neurons each, and an input layer with 50 features obtained from PCA-reduced data reflects the CIFAR-10 categories. While each hidden layer introduced nonlinearity using the ReLU activation function, the second hidden layer used batch normalization to boost training stability and expedite convergence.

The dimensions of the hidden layers were altered in order to find effects on performance. By retraining the model with hidden layer sizes of 256, 512 (the default), and 1024 one could see how computational cost and performance varied. The changes made allowed one to evaluate how generalizing from data capabilities of the network varied with capacity.

The training method was constant throughout all versions using the cross-entropy loss function (nn. CrossEntropyLoss) handling multi-class classification. Using Stochastic Gradient Descent (SGD) with a learning rate of 0.01 and momentum of 0.9, optimization was achieved. These hyperparameters aimed to strike a mix between the speed of optimization convergence and the stability kept during the process. Running 20 epochs with a batch size of 64 utilizing mini-batch gradient descent allowed the training strategy to maximize computational performance and memory use.

The model used PCA made CIFAR-10 feature vectors dimensionally reduced to 50 components. Using a reusable training tool helped to handle training and assessments, thereby maintaining consistency between experiments. Monitoring the training loss for every epoch, among other things this function assessed test dataset accuracy, precision, recall, and F1-score. The design decisions enable a complete investigation of the impact of network architecture—more specifically, the dimensions of hidden layers—on the learning and generalization capability of the model, even while maintaining a careful balance of processing costs and performance.

Convolutional Neural Network

VGG11 convolutional neural network development tracked project requirements. Built in a sequence of convolutional layers, the model also has fully connected layers and batch normalisation. Every convolutional layer guarantees spatial consistency using a 1-stride, 3-pixel kernel size and 1-pixel padding. Batch normalizing comes next for regularization; subsequently, a ReLU activation supplies nonlinearity. Using a 2-kernel size and a 2-stride, the model makes many max-pooling calls, each gradually shrinking the spatial dimensions of the feature maps. The resulting feature map after the convolutional layers is 512×1×1; it is then flattened into a 512-dimensional vector and passed across fully connected layers.

There are three-linear layered classifiers. First layer converts 512-dimensional input into 4096 features; a ReLU activation and a dropout layer set at 0.5 then help to prevent overfitting. Dropout and ReLU activation follow the layer preserving the 4096-dimensional mapping next absolutely connected. Stating the categories of the CIFAR-10 dataset, the final layer divides the 44 000-dimensional feature vector into ten output groups.

20 epochs, a batch size of 64, and the cross-entropy loss function let the training process evaluate the changes in expected and actual labels. By thus raising convergence, reducing oscillations by use of stochastic gradient descent (SGD) with a learning rate of 0.01 and a momentum of 0.9 helps to enhance the model. Effective updating of model parameters is made achievable by mini-batch

gradient descent. Starting with this basic concept, one may look at further architectural or hyperparameter changes.

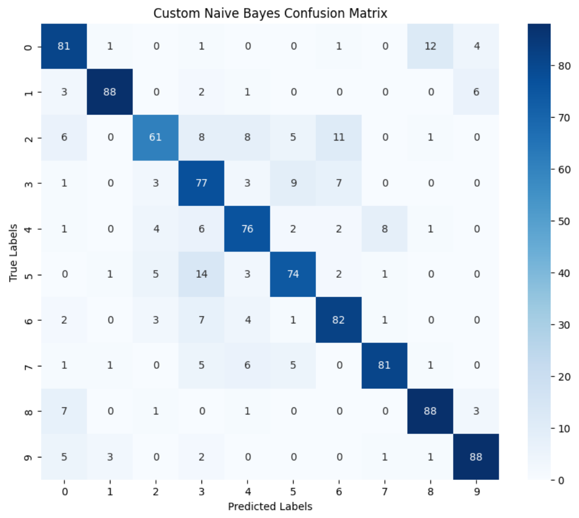
Evaluation

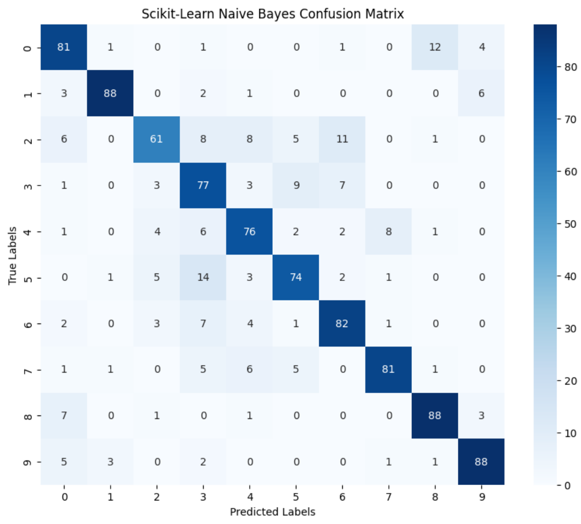
Naïve Bayes

With equal metrics across all evaluated categories—accuracy, precision, recall, and F1-score—which ranged all around 0.80—both the Custom Naïve Bayes and Scikit-Learn Naïve Bayes models performed consistently.

	Accuracy	Precision	Recall	F1-Score
Custom NB	0.8	0.8	0.8	0.8
Scikit-Learn NB	0.8	0.8	0.8	0.8

This shows that, providing a consistent substitute, the Custom implementation faithfully reproduces the capability of the well-known Scikit-Learn Naive Bayes approach. In terms of identifying the trends in the dataset, both naïve bayes models show usually good performance.





Both models' confusion matrices point up areas that need for development as well as strength. With minimal misclassifications, classes 1, 7, and 9 exhibit remarkable accuracy; classes 2 and 3 contain notable entries off the centre diagonal, indicating regular mistakes into adjacent classes 3 and 5. These results show that even if the models show good performance generally, more feature engineering or sophisticated algorithms might be required to address these particular challenges.

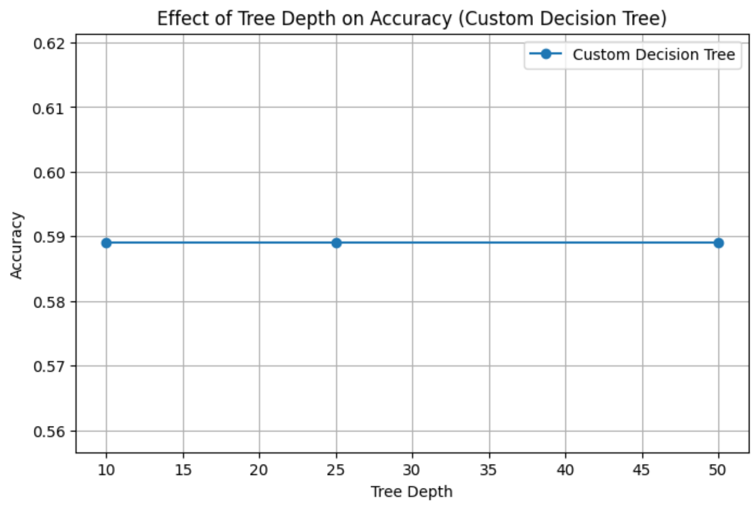
Particularly important in applications like facial image analysis, the balanced F1-score of 0.80 shows that both models effectively balance accuracy and recall. Under these conditions, high recall guarantees that all pertinent faces are found, and high accuracy lowers false detections. Strong performance across both systems is shown by the findings; potential for targeted enhancements in areas of class overlap exists. Our following implementations will start with these models as the foundation for comparison going ahead.

Decision Tree

With accuracies of 59% and 61% respectively, the decision tree models—custom and Scikit-learn—performed somewhat poorly on the classification challenge.

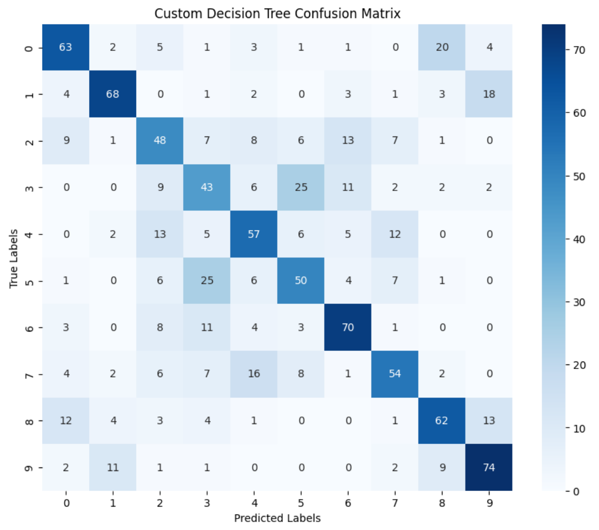
	Accuracy	Precision	Recall	F1-Score
Custom DT (d=50)	0.59	0.59	0.59	0.59
Custom DT (d=25)	0.59			
Custom DT (d=10)	0.59			
Scikit-Learn DT	0.61	0.60	0.61	0.60

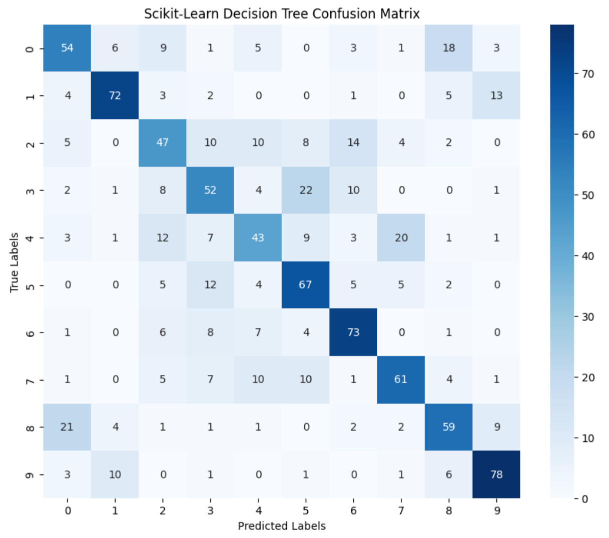
The real variance in execution durations was very significant, even if the variation in correctness seems to be negligible. Whereas the Scikit-learn tree finished fairly immediately, the custom decision tree may take up to a minute to run for a single depth configuration. This disparity draws attention to significant improvements in the Scikit-learn variant including efficient data processing, best tree-building techniques, and simplified calculations.



Though it was thought that deepening the custom decision tree would increase accuracy, the graph above indicates no variation in depth (10, 25, and 50). This failure implies that maybe because of inadequate Gini impurity calculations or thresholding logic, the custom tree is not efficiently using deeper splits or handling halting circumstances. It draws attention to a flaw in the implementation strategy as deeper trees, without overfitting, usually help identify more intricate trends in data.

The confusion matrices show that, especially in classes 2 and 8—often confused with other categories—both models produced comparable misclassifications. This suggests a tighter feature representation limit, so, while utilizing the same PCA-reduced feature set, the Scikit-learn tree outperformed the custom version because of its efficiency and most likely more exact impurity metrics.





The long length of the custom implementation implies probable inefficiencies in node growth or split evaluation. By means of caching computations or strengthening the reasoning for selecting splits, improving certain procedures—such as those related to computation time—may help to match the highly optimal library of Scikit-learn. Scikit-learn also makes use of proven pruning techniques and some optimizations difficult to reproduce in a custom configuration.

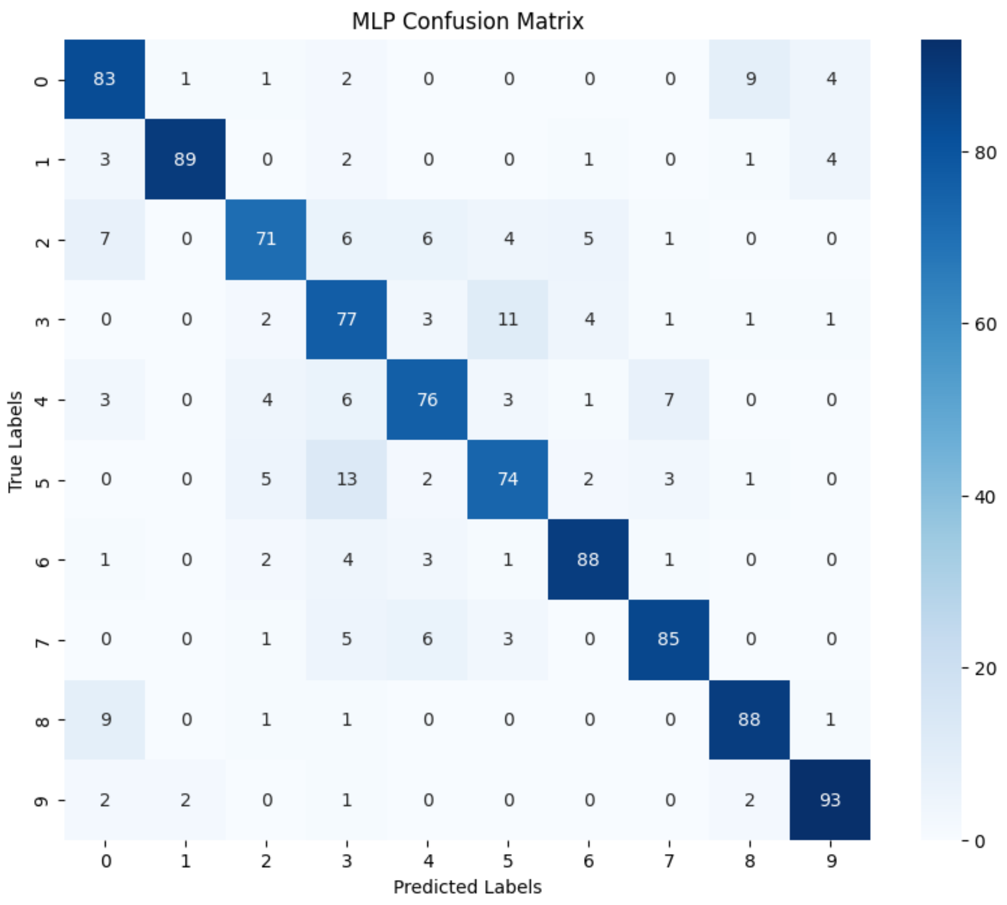
Although feature restrictions caused comparable classification difficulties for both models, the Scikit-learn decision tree proved to be more practical for this project in runtime and user-friendliness.

Multi-Layer Perceptron

On the classification tests, the Multilayer Perceptron (MLP) model performs truly excellently with 82% accuracy, 83% precision, 82% recall, and an F1-score of 82%. As it reduces false positives and false negatives, the MLP balances well accuracy and recall; this is a key criterion for face image processing.

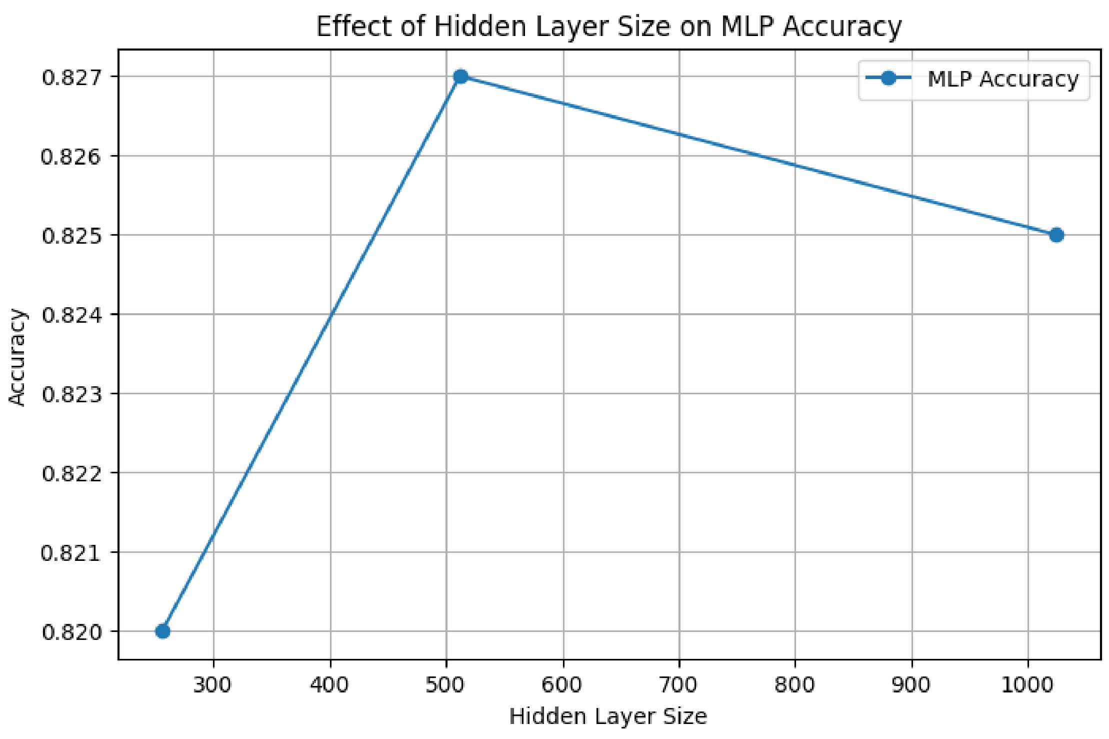
	Accuracy	Precision	Recall	F1-Score
Multi-Layer Perceptron	0.82	0.83	0.82	0.82

The confusion matrix shows that the MLP still struggled in a few classes, particularly classes 2 and 8 where notable misclassifications occurred. Some issues in particular classes might arise from overlapping feature distributions or inadequate suitable feature representation. Good MLP performance indicates that it can sufficiently control PCA feature set decreased dimensionality.



	0	1	2	3	4	5	6	7	8	9
0	83	1	1	2	0	0	0	0	9	4
1	3	89	0	2	0	0	1	0	1	4
2	7	0	71	6	6	4	5	1	0	0
3	0	0	2	77	3	11	4	1	1	1
4	3	0	4	6	76	3	1	7	0	0
5	0	0	5	13	2	74	2	3	1	0
6	1	0	2	4	3	1	88	1	0	0
7	0	0	1	5	6	3	0	85	0	0
8	9	0	1	1	0	0	0	0	88	1
9	2	2	0	1	0	0	0	0	2	93

By use of several concealed layer widths, research revealed significant insights on architectural design's influence on performance. The model became 82.7% accurate by extending the hidden layer size from 256 to 512. This variant stresses how avoidance of overfitting and a medium-sized hidden layer help the model to capture more complicated feature interactions. Accuracy declined drastically to 82.5% as the size grew to 1024, implying perhaps overfitting with even more bigger layer sizes and falling returns. The results highlight the importance of modifying hyperparameters in MLPs to obtain best performance.



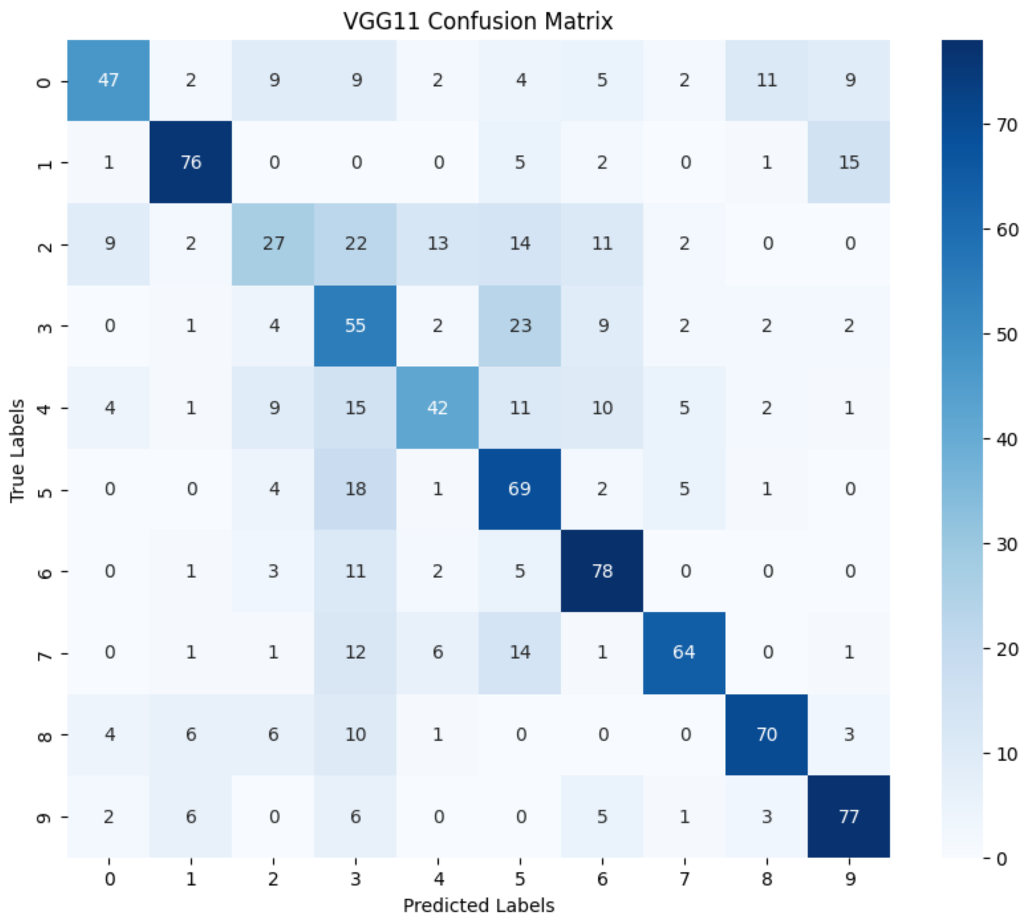
Although the MLP model performed very well, misclassifications in some classes suggest more feature engineering or model change required. Furthermore revealed by the sensitivity of accuracy to hidden layer sizes is the requirement of extensive testing in equilibrium of generalization and model capacity.

Convolutional Neural Network

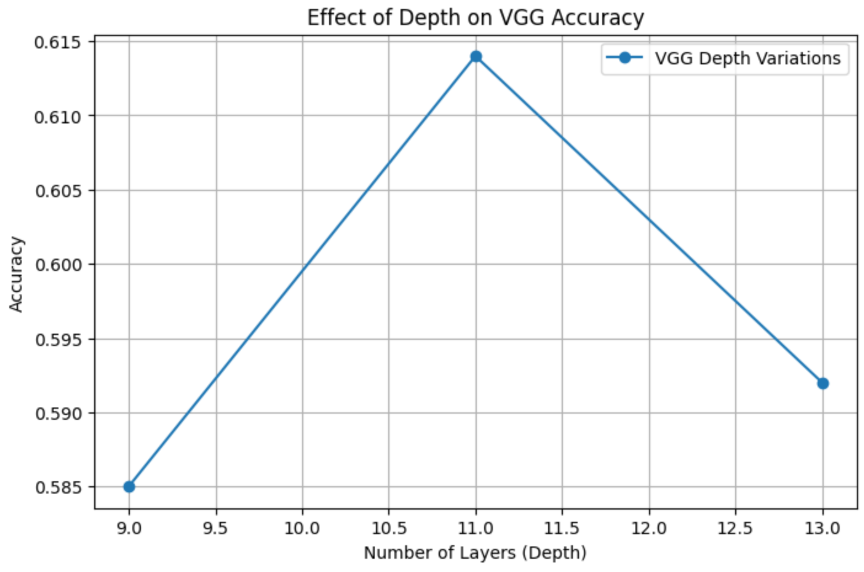
CNN-built models, most notably VGG designs and tweaks, showed respectable classification performance. With F1 scores of 0.63, 0.60, and 0.60 respectively, the baseline VGG11 possesses precision, recall, and accuracy. Furthermore, the accuracy was 60%.

	Accuracy	Precision	Recall	F1-Score
Multi-Layer Perceptron	0.60	0.63	0.60	0.60

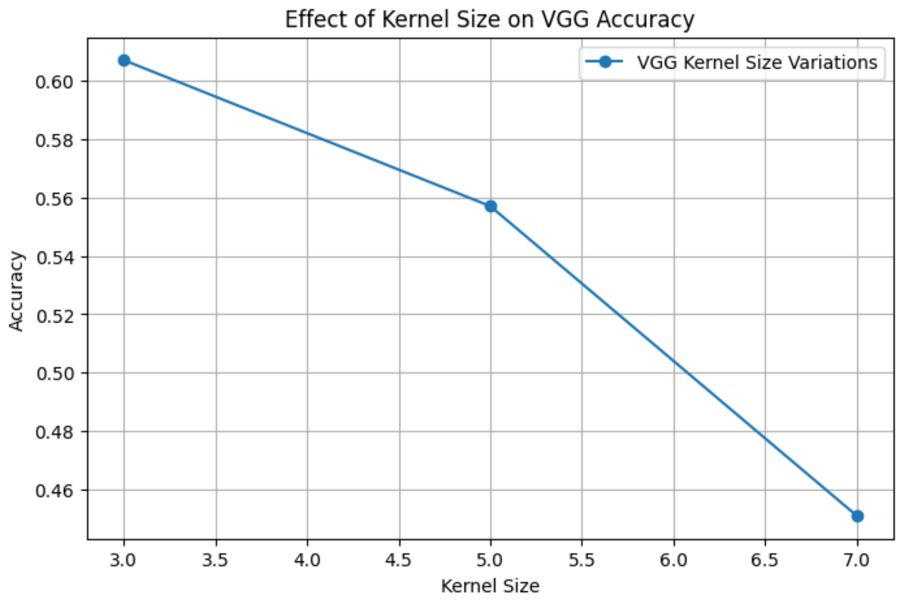
The confusion matrix of the CNN model turned up some interesting patterns. Commonly misclassified classes 2, 3, and 4 were those having overlap with adjacent classes. Working with PCA-reduced data, the overlap in feature space between these groups might be unclear. By comparison, classes 6 and 9 were correctly recognized most likely because of their varied and more clear feature representations. Even in more complicated models, convolutional layers might be able to identify spatially varying characteristics in class 6.



The VGG9, VGG11, and VGG13 models performed very consistently; VGG11 outperformed the other two in long-term studies. As one moved from VGG11 to VGG13, the architectural accuracy considerably dropped. This indicates that adding layers without following appropriate optimization techniques might cause overfitting or deteriorating performance.



Smaller kernels, like size 3, showed the highest performance with 61% accuracy; conversely, larger kernels, such size 7, showed substantially lower performance with just 45% accuracy. The study of kernel size indicated a distinct trend. This outcome emphasizes the need of kernel size in maintaining localized information in image data.



A few factors lower the efficiency of deeper models, notably VGG13. Although adding additional layers increases the possibility of overfitting, particularly in cases with a limited dataset, in principle it improves feature extraction. Moreover, the increasing number of parameters might call for a more exact training method and greater regularization, which could not have been sufficient in this situation. Likewise, using larger kernels limited the model's ability to catch fine features, therefore lowering the accuracy. Conversely, smaller particles did better as they could concentrate on more focused, smaller patterns needed for class identification. Researching further improvements—such as data augmentation, variable learning rate schedules, or increased regularization techniques—is essential in general to minimize overfitting and boost generalization across many classes. Additionally improving performance might be experimenting with additional architectural changes such residual connections or attention techniques.

Primary Findings

Model	Accuracy	Precision	Recall	F1-Score
Naïve Bayes	0.80	0.80	0.80	0.80
SKL Naïve Bayes	0.80	0.80	0.80	0.80
Decision Tree	0.59	0.59	0.59	0.59
SKL Decision Tree	0.61	0.60	0.61	0.60
Multi-Layer Perceptron	0.82	0.83	0.82	0.82
Convolutional Neural Network	0.60	0.63	0.60	0.60

With 82% accuracy, 83% precision, and 82% F1-score, the Multi-Layer Perceptron (MLP) performed best. Learning nonlinear correlations using PCA-reduced CIFAR-10 is a strength of the model. ReLU activation and hidden layer widths were tweaked to enable free from overfitting generalizing of the MLP. Naive bayes models showed 80% accuracy although their performance suffered from their assumptions about feature independence and Gaussian distributions. This restriction most definitely stopped them from noticing more sophisticated trends than the MLP.

With somewhat low 60% accuracy and 60% F1-score, the CNN model (VGG11) performed worse than expected. Poor hyperparameter tuning may have limited feature extraction in this respect, even if convolutional layers have complex architecture and can manage spatially coupled data. With 59% and 61% respective accuracy, both custom and Scikit-Learning Decision Tree models failed. The custom tree's lack of deepening progress suggests to split calculation inefficiencies or poorly optimized stopping conditions, therefore restricting the model's capacity to learn complicated data distribution.

Every model showed sometimes overlapping in feature space misclassifications in classes 2, 3, and 8. This problem results from PCA compression of class boundaries reducing dimensionality. Classes with minor visual variances or overlapping characteristics get more difficult to identify. In CNNs and MLPs, six and nine scored decent. These classes might have different textures or forms fit for the capability of feature extraction of various models.

Especially in computationally challenged environments even with just 2% less accuracy than the MLP, the Naïve Bayes model shines in efficiency and dependability. Naïve Bayes' simplicity and faster computation made it a more sensible option for events giving speed and economy top priority above accuracy; MLP's ability to identify intricate relationships enhanced accuracy. In this sense, the Naïve Bayes model's exceptional performance and low processing cost help it to succeed.

Little changes might make every model operate better. Using dimensionality reduction techniques and feature engineering optimization helps one to raise the capacity of naive bayes to discriminate overlapping classes. While the MLP is doing well, additional study employing learning rate schedules, dropout regularization, and hyperparameter tuning might assist to better balance capacity and generalization. Though interesting, the CNN model requires hyperparameter value optimization. In architecture, residual connections and data augmentation might help to hasten processes. Custom and Scikit-learn implementations would be less separated if improved pruning techniques, caching split computations, and strengthening stopping conditions aided to boost the efficiency and accuracy of Decision Tree models. These focused model enhancements provide best performance for particular usage situations with restricted resources.