Analysis of Machine Learning Algorithms on CIFAR-10 Image Classification

Prepared by: Hanae Razafindrakoto (40249714)
Course: COMP 472 Artificial Intelligence
Instructor: Kefaya Qaddoum

Table of Contents

I. Introduction

The goal of this project was to classify images from the CIFAR-10 dataset using machine learning models and evaluate their performance. The CIFAR-10 dataset is a collection of 32x32 RGB images separated into 10 classes, which were then limited to only 500 training images and 100 test images per class. ResNet-18 was used to extract 512-dimensional feature vectors, and which was then converted into 50 dimensions using Principal Component Analysis (PCA).

Four models were trained and tested: a Gaussian Naive Bayes classifier, a Decision Tree classifier, a Multi-Layer Perceptron (MLP), and a Convolutional Neural Network based on the VGG11 architecture. For each model, metrics like confusion matrix, accuracy, precision, recall, and F1-score were used in the evaluation. Lastly, variations of the Decision Tree, MLP, and CNN models were tested to analyze how depth (and kernel size for CNNs) affected computational cost and accuracy.

II. Model Architecture and Training

A. Gaussian Naive Bayes

The Gaussian Naive Bayes (GNB) algorithm is a classifier that operates under the assumption that features are independent in each class and that they follow a Gaussian distribution. It calculates the likelihood of a feature x given a class C_k using the formula:

$$P(x|C_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ and σ are the mean and standard deviation of x for class C_k. Hence, combining these likelihoods with class priors will tell which class to select from among them as the probability of it being the correct class for a sample.

For this project, I built and trained a custom GNB model that classifies the CIFAR-10 in contrast to that of a Scikit-learn implementation for comparison. My custom model calculated the means, standard deviations, and priors for each class during the training phase. Predictions were made by calculating the posterior probabilities for each class using the formula above and selecting the class with the highest score. Both implementations had the same accuracy (77.9%), and the custom GNB logged an effective training time of 0.00 seconds. This speed reflects how all parameters are computed in a single pass through the data.

B. Decision Tree

Decision tree classifiers are algorithms that predict class labels based on recursively splitting of the dataset according to feature values. At each node, the algorithm selects that feature and threshold to minimize the Gini impurity, which can be defined as:

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

where p_i is the proportion of samples belonging to class i. A Gini score of 0 represents “perfect purity,” while higher values reflect mixed-class distributionis. The process continues until the maximum depth is reached or until the data becomes perfectly classified.

In this project, I developed a custom decision tree model and compared it with a Scikit-learn implementation. The custom model builds a tree recursively that picks the best feature and threshold at each split using the Gini coefficient.

Both implementations got very similar results, with the custom tree scoring a test accuracy of 58.5% and the Scikit-learn model scoring 57.7%. I also examined how varying the tree's depth affects performance by training on 7 depths: 1, 2, 5, 10, 20, 50, 100. The results made it clear that while deeper trees improved training accuracy, they often overfit the data, leading to less increases in test accuracy. A more detailed analysis of these variations is presented later in this report.

C. Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) is an example of a feed-forward neural network that assigns class labels by processing inputs through several layers of connected nodes. Each layer applies a mathematical transformation to the input and uses a non-linear activation function to learn complex patterns in the data. For this project, the default MLP model consisted of three layers:

- The first layer converted 50 input features through the initial layer into 512 dimensions and activated with ReLu.
- The second layer retained the 512 dimensions, applied batch normalization for stability, and used another ReLU activation.
- The final layer mapped the 512-dimensional data to 10 outputs, associated with each CIFAR-10 class.

The above-mentioned model was trained for 100 epochs using a Cross-Entropy Loss function and Stochastic Gradient Descent (SGD) optimizer, with the learning rate being 0.01 and momentum of 0.9. During training, I tracked both the loss and test accuracy at regular intervals to evaluate performance.

To analyze how the depth of the network influences its accuracy, I also experimented with models having between 1 and 6 hidden layers. Deeper networks generally improved test accuracy, but at some point there would be no further effect. A more detailed analysis of these variations is presented later in this report.

D. Convolutional Neural Network (VGG11)

A Convolutional Neural Network (CNN) is trained over a number of different layers (convolutions, pooling, activations) to perform spatial feature extraction, then followed by fully connected layers for classification. The VGG11 model used in this project included:

- Feature extraction layers: Five blocks of convolutional layers, each followed by ReLU activations and max-pooling layers. The feature depth increased across the blocks, starting with 64 channels and ascending to 512 in the final three blocks.

- Fully connected layers: The fully connected layers passed flattened input data through 4096 neurons ReLU activations dropout at each through two layers. Probabilities are given for 10 classes by the last layer.

The model was trained for 50 epochs using a cross-entropy loss function and stochastic gradient descent (SGD) optimizer with a learning rate of 0.01 and momentum of 0.9. Whenever the accuracy of the tests plateaued, the learning rate was decreased by using a learning rate scheduler. Training progress was evaluated using test accuracy and loss after each epoch.

In addition to training the default VGG11 model, variations in the number of convolutional layers (depth) and kernel sizes were analyzed. Depths ranged from 2 to 10 even layers, and kernel sizes tested included 2x2, 3x3, 5x5, and 7x7. Adaptive average pooling was added to standardize feature sizes, which optimizes training time. Results showed that increasing depth and kernel size generally improved the accuracy, but the improvements stagnated beyond certain thresholds and resulted in much longer training times. A more detailed analysis of these variations is presented later in this report.

III. Analysis and Evaluation

A. Evaluation Metrics

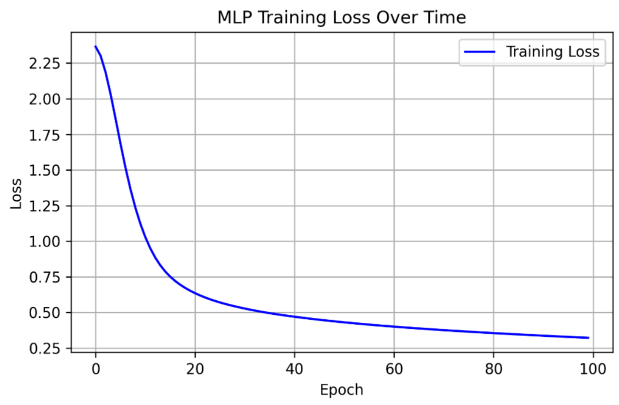
Algorithm	Accuracy	Precision	Recall	F1
Gaussian Naive Bayes	0.779	0.783	0.779	0.778
Scikit GNB	0.779	0.783	0.779	0.778
Decision Tree	0.585	0.586	0.585	0.585
Scikit Decision Tree	0.577	0.581	0.577	0.578
MLP	0.840	0.842	0.840	0.840
CNN	0.715	0.712	0.715	0.712

Figure 1. Table of Algorithm Evaluation Metrics

The Gaussian Naive Bayes models, both custom and Scikit-learn, got an accuracy of 77.9% with matching precision, recall, and F1-scores. The consistency of these metrics indicates a reliable performance of the model across all classes. Nevertheless, the rigidity of feature independence makes it inferior to the use of more advanced models like MLPs.

The custom Decision Tree yielded an accuracy of 58.5 %, which is a slight margin better off that of Scikit-learn at 57.7 %, though both presented similar precision, recall, and f1-scores reinforcing the decision tree's incapability in handling data complexities. It overfit the training data, even with depth limitations, which produces lower test accuracy and inconsistent class predictions.

The MLP performed the best, achieving 84.0% accuracy and well-balanced precision, recall, and F1-scores. This indicates that the MLP model is efficient in learning the data patterns and deals better with class imbalances than simpler models. Its deeper structure also allowed it to generalize better; thus, this stands as the best overall model.



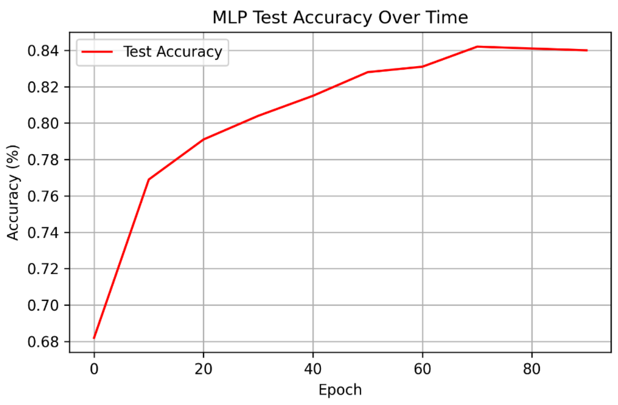


Figure 2. Graphs of MLP Training Loss and Test Accuracy Over Time

The CNN, using the VGG11 architecture, reached 71.5% accuracy with slightly lower precision and recall. This indicates it had more false positives and didn't do well with consistent class predictions. While the CNN did well at capturing spatial features, its overall performance lagged behind the MLP, likely due to the shorter training duration (50 epochs), which was set due to computational capacity of my system.

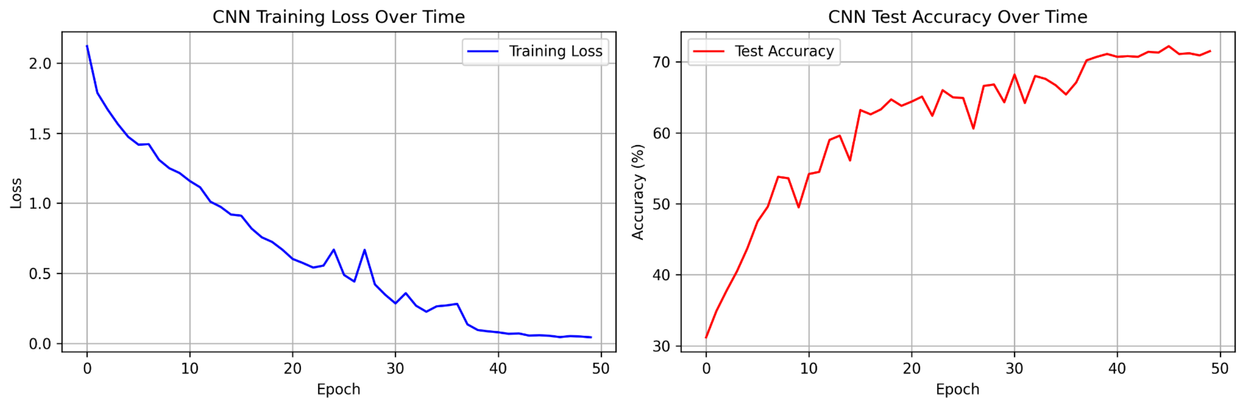


Figure 3. Graphs of CNN Training Loss and Test Accuracy Over Time

B. Confusion Matrices

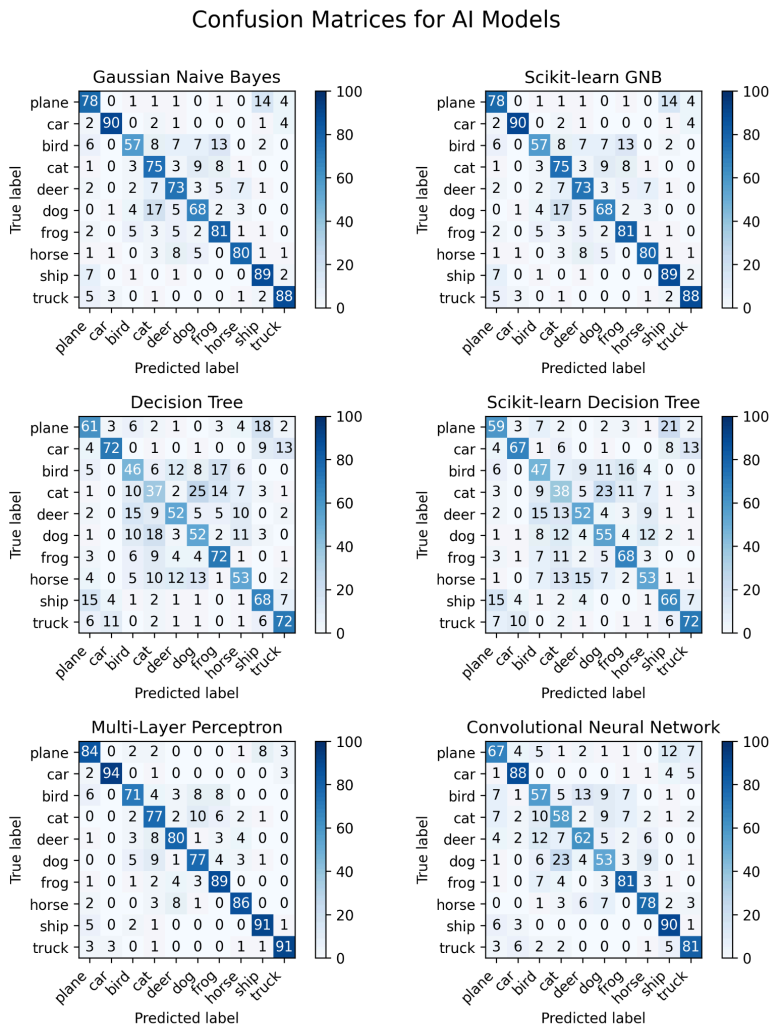


Figure 4. Confusion Matrices for 6 models

Model-Specific Accuracy Trends

Figure 5 shows that the Decision Tree models did poorly compared to GNB, MLP, and CNN models. They had trouble with the "bird" (46%) and "cat" (37%) classes and showed confusion between similar classes like "dog" and "cat." This is likely because Decision Trees overfit to the training data, making their predictions less generalizable.

GNB, despite being simple, performed well for distinct classes like "car" (90%), "ship" (89%), and "truck" (88%). It worked well for these clear-featured classes, but it struggled with classes like "bird" (57%), where features overlap with other classes.

The CNN didn't perform as well as expected. It had lower accuracy on classes like "dog" (53%) and "bird" (57%). This is likely due to the fact that it was trained for only 50 epochs, compared to 100 epochs for the MLP. Therefore, disallowing it to fully learn the patterns made available in the dataset. CNNs also need careful tuning, which might explain the lower performance.

Among all the classifiers, the MLP performed best overall, with the highest accuracy on most classes, including "car" (94%), "ship" (91%), and "truck" (91%). It handled complex patterns well and was better at separating similar classes.

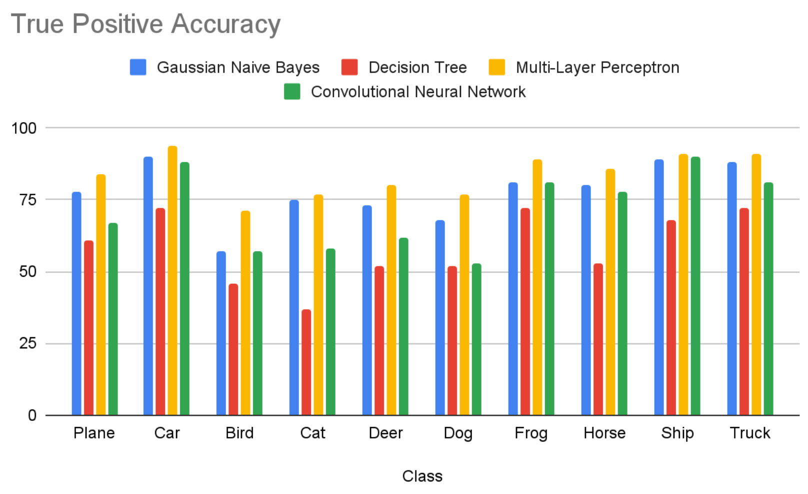


Figure 5. Graphs of True Positive Accuracy By Class Per Model

Class-Specific Performance Trends

Figure 6 shows the average accuracy of each class across all models. "Car" had the highest average accuracy, followed by "ship" and "truck." These classes are relatively easy to classify because they have distinct shapes and textures. On the other hand, "bird" and "cat" had the lowest average accuracy. These classes are harder to classify since they could share features with other classes, like "dog" or "frog."

There was much confusion between some classes, like "cat" and "dog" or "frog" and "bird." For example, "cat" and "dog" share many similar features, like fur color, which can confuse the models. Advanced models like CNN and MLP reduced these errors but didn't remove them entirely.

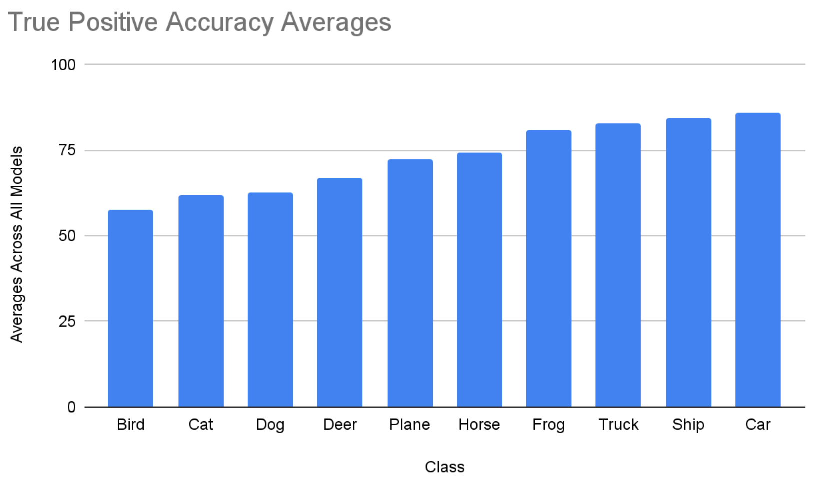


Figure 6. Graphs of True Positive Average Accuracy By Class

C. Depth and Kernel Size Analysis

Depth Analysis

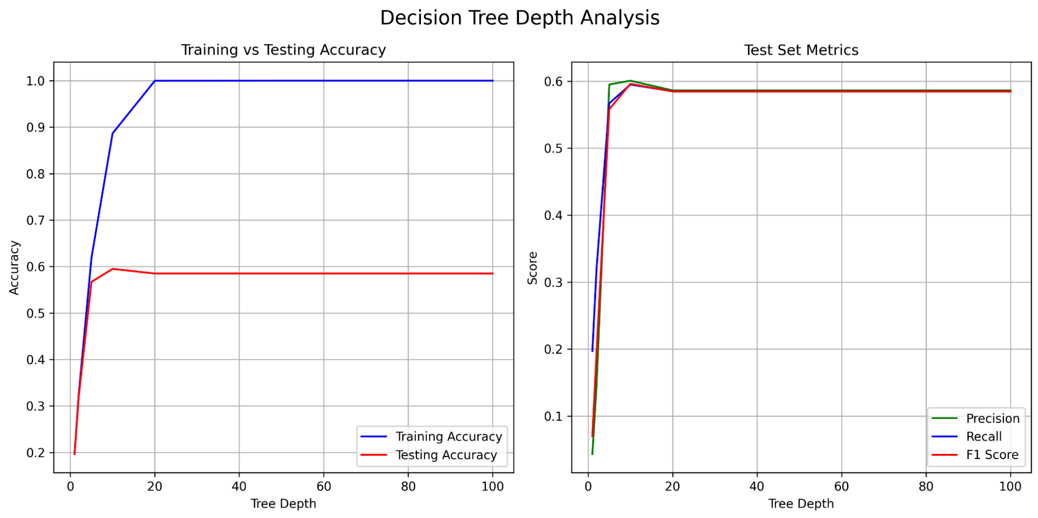


Figure 7. Graphs of Decision Tree Depth Analysis

The Decision Tree analysis (Figure 7) shows that increasing depth initially improves performance but quickly leads to overfitting. At lower depths (e.g., depth 1 or 2), both training and testing accuracy are low, with a test accuracy of 19.7% and 32.1%, respectively. As depth increases to 10, the training accuracy reaches 88.7%, but test accuracy stays the same at around 59.5%. Past depth 20, the model memorizes the training data entirely (100% training accuracy), but test accuracy is still unchanged at 58.5%, suggesting overfitting.

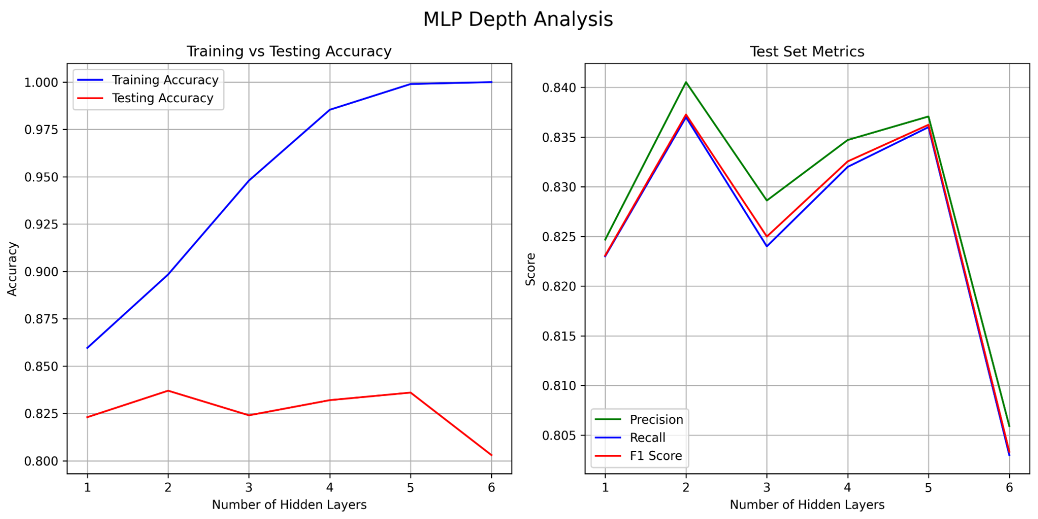


Figure 8. Graphs of Multi-Layer Perceptron Depth Analysis

The MLP depth analysis (Figure 8) shows a more nuanced relationship between depth and performance. Adding more layers initially improves test accuracy, peaking at two hidden layers with 83.7%. Training accuracy increases with more layers, reaching 100% with six layers, but test accuracy also drops to 80.3%. This indicates overfitting when too many layers are added.

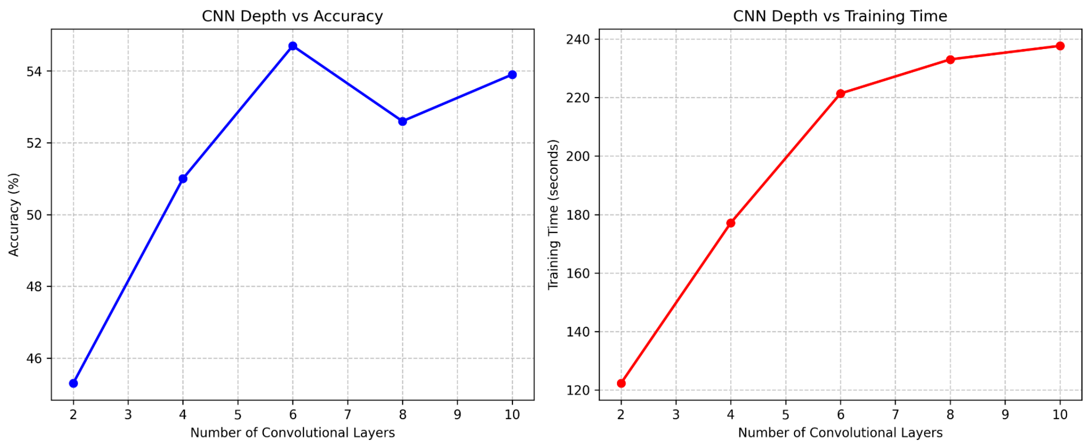


Figure 9. Graphs of Convolutional Neural Network Depth Analysis

The CNN analysis (Figure 9) shows that accuracy improves as depth increases from 2 layers (45.3%) to 6 layers (54.7%), but deeper networks like 8 or 10 layers show slight drops in accuracy (52.6% and 53.9%, respectively). This suggests that while additional depth allows the model to have more features initially, potentially overfitting happens at greater depths. Training time increases significantly, almost doubling from 122.33 seconds at 2 layers to 237.67 seconds at 10 layers, making deeper models computationally costly without much better performance.

Kernel Size Analysis

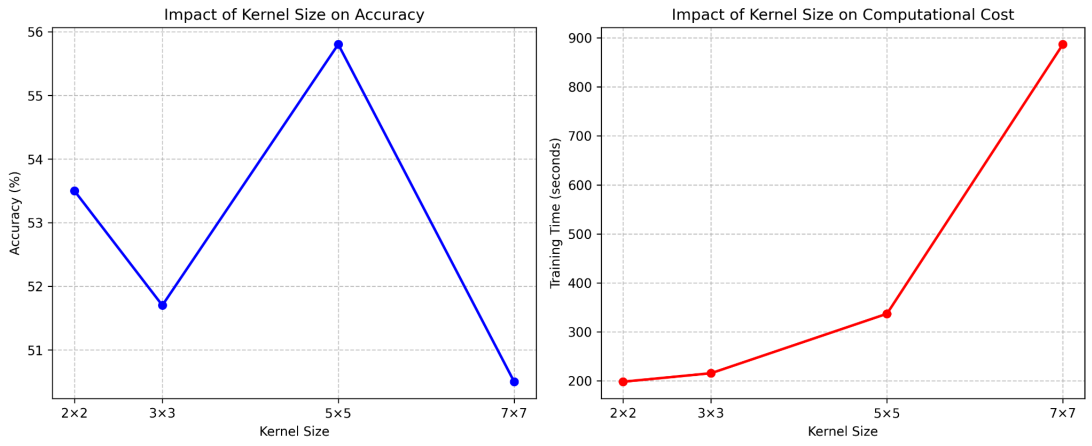


Figure 10. Graphs of Convolutional Neural Network Kernel Size Analysis

The CNN kernel analysis (Figure 10) highlights how the choice of kernel size significantly impacts its ability to capture features and its computational cost. Smaller kernels like 2x2 and 3x3 focus on fine-grained details, but they may miss larger patterns, which can explain their lower accuracy (53.5%, 51.7%) compared to the 5x5 kernel. The 5x5 kernel has the best accuracy (55.8%), likely because it balances the extraction of both detailed and broader features, making it more effective at recognizing complex spatial structures. In contrast, the 7x7 kernel loses the finer detail by oversmoothing, which results in the lowest accuracy (50.5%).

From a computational perspective, larger kernels come with a large increase in training time. While the 5x5 kernel is best in terms of accuracy, it needs significantly more resources than smaller kernels like 2x2 or 3x3. The 7x7 kernel's high computational cost combined with its poor performance demonstrates diminishing returns for larger kernels.

IV. Conclusion

This project compared the performance of four models – Gaussian Naive Bayes, Decision Tree, Multi-Layer Perceptron, and Convolutional Neural Network – on the CIFAR-10 dataset. The best performance was from the MLP, with an accuracy of 84%, as it learned complex patterns and could generalize well, but needed optimization in depth not to overfit.

The CNN returned an accuracy of 71.5%, performing well because of its strength in extracting spatial features. However, its performance was limited by a shorter training duration of 50 epochs, which likely prevented it from fully learning the dataset's patterns. Gaussian Naive Bayes showed surprising effectiveness with an accuracy of 77.9%, despite its simplicity, did very well in classes with distinct features but badly with overlapping ones. The Decision Tree performed worst, with an accuracy of 58.5%, due to the effect of overfitting at larger depths, which seriously reduced its ability to generalize.

In conclusion, the MLP was more powerful than the other models due to its ability to balance the learning of complex features without overfitting, making it the best choice regarding this classification task.

V. References

[1] Pytorch, “Training a Classifier — PyTorch Tutorials 1.5.0 Documentation,” *pytorch.org*. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

[2] Nandita Bhaskhar, “Intermediate Activations — the Forward Hook,” *Nandita Bhaskhar*, Aug. 17, 2020. https://web.stanford.edu/~nanbhas/blog/forward-hooks-pytorch/?source=post_page-----ed79da32c950-------------------------------- (accessed Nov. 26, 2024).

[3] A. Tam, “Principal Component Analysis for Visualization,” *Machine Learning Mastery*, Oct. 20, 2021. https://machinelearningmastery.com/principal-component-analysis-for-visualization/

[4] deepmancer, “GitHub - deepmancer/resnet-cifar-classification: A step-by-step implementation of a ResNet-18 model for image classification on the CIFAR-10 dataset,” *GitHub*, 2022. https://github.com/deepmancer/resnet-cifar-classification (accessed Nov. 26, 2024).

[5] ctrnngtrung, “A Simple VGG11 Model with Tensorflow & PyTorch,” *Kaggle.com*, Aug. 27, 2023. https://www.kaggle.com/code/ctrnngtrung/a-simple-vgg11-model-with-tensorflow-pytorch

[6] S. Jaiswal, “Multilayer Perceptrons in Machine Learning: A Comprehensive Guide,” *Datacamp.com*, Feb. 07, 2024. https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

[7] GeeksforGeeks, “Multi-Layer Perceptron Learning in Tensorflow,” *GeeksforGeeks*, Nov. 03, 2021. https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/

[8] Scikit-learn, “1.17. Neural network models (supervised),” *scikit-learn*, 2024. https://scikit-learn.org/1.5/modules/neural_networks_supervised.html

[9] Scikit-learn, “1.10. Decision Trees,” *scikit-learn*, 2024. https://scikit-learn.org/1.5/modules/tree.html

[10] Scikit-learn, “GaussianNB,” *scikit-learn*, 2024. https://scikit-learn.org/dev/modules/generated/sklearn.naive_bayes.GaussianNB.html

[11] ChatGPT, “ChatGPT,” *ChatGPT*, 2024. https://chatgpt.com/

[12] Jason Brownlee, “How Do Convolutional Layers Work in Deep Learning Neural Networks?,” *Machine Learning Mastery*, Apr. 16, 2019. https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/