A Concordia University

Prepared for
Professor. Kefaya Qaddoum

COMP 472 - Introduction to Artificial Intelligence
Section NN 2242

Github: https://github.com/genericlearner/COMP472-Proj.git

We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality.

By
Zaouhair Chich Katana 40263998 and Mohammed Umaruddin 40204345

1. Introduction

This project uses different machine-learning models to classify images from the CIFAR-10 dataset. The CIFAR-10 dataset is a popular benchmark in computer vision, containing 60,000 images spread across ten categories. Using this dataset, we aim to test how well different models can classify images and identify the best methods.

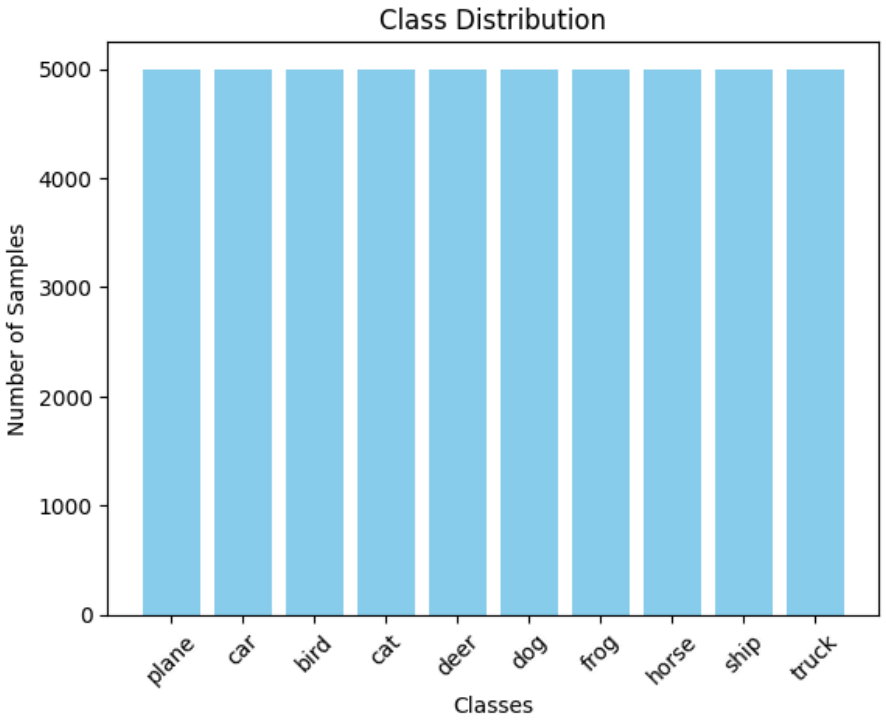
We experimented with four models: Naive Bayes, Decision Tree, Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN). Each model employs a unique method to tackle the classification task. Naive Bayes relies on probabilities and assumes feature independence, whereas Decision Tree creates decision rules from the training data to make predictions. MLP and CNN, both neural network models, take a more advanced approach, with CNN being optimized explicitly for image data, making it the most specialized and complex among the four.

To prepare the data for these models, we applied several preprocessing steps. This included resizing the images, normalizing their pixel values, and using a pre-trained ResNet-18 model to extract useful features from the images. Since the extracted features had many dimensions, we used Principal Component Analysis (PCA) to reduce them to a more manageable size, making the models faster and easier to train.

This project aims to evaluate these models' performances in terms of accuracy and additional metrics like precision, recall, and F1-score. By examining their advantages and disadvantages, we can learn more about which models are best suited for related tasks in the future.

2. Dataset and Preprocessing

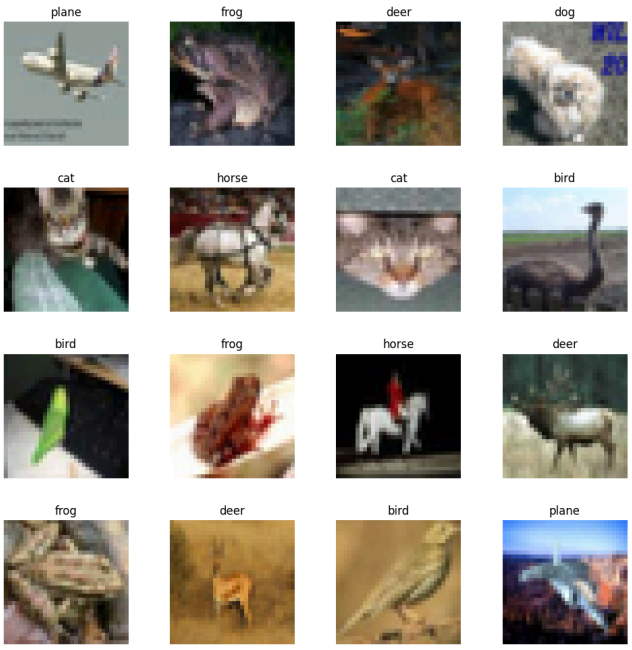
For this project, the CIFAR-10 dataset included very small images, just 32x32 pixels, split into 50,000 for training and 10,000 for testing.



The datasets include ten classes, including animals, birds, and transportation methods. The images are equally divided into classes, with 5000 images for each class for training. The variety of classes in the dataset highlights that our model will be tested on different backgrounds, positions, and other specific details.

- **Dataset Description**:
    - Reduced dataset: 500 training and 100 test images per class.

Example Images:



A key point in this section is that we are using 10% of the data for each class. This poses a potential challenge of overfitting. We aim to train VGG11 for CNN, which has a deep architecture. Additionally, we use features extracted from the ResNet-18 model. The complexity of these models and lack of data introduces overfitting in our project. Regardless of this challenge, we aim to train the data on various models, including NaiveBayes, Decision Trees, Varying MLP models and modified VGG11 architectures.

To ensure consistency and compatibility across all models(except CNN), we followed a unified preprocessing pipeline (as detailed in the **General Preprocessing Section**). This pipeline involved four steps:

- **Resizing Images**
- **Normalizing Pixel Values**
- **Feature Extraction**
- **Dimensionality Reduction**

By following these preprocessing procedures, we were able to effectively prepare the dataset for every model that was put into use, guaranteeing uniformity in both training and evaluation.

General Preprocessing Section:

We started by getting the dataset ready before putting the models into practice. Preprocessing the data was the first thing we did. The images were resized to 224 x 224 pixels, their pixel values were normalized using the mean and standard deviation of CIFAR-10, and a pre-trained ResNet-18 model was used to extract features. Our models' input was greatly simplified using ResNet-18 to transform the raw image data into helpful feature vectors. Figure 1 below provides a summary of the preprocessing pipeline.

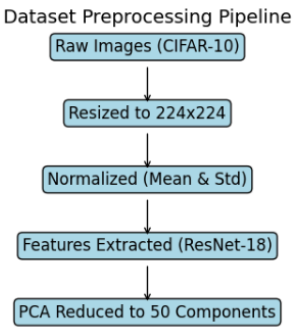


Figure 1: Dataset Preprocessing Pipeline

Next, we reduced the dimensionality of these feature vectors using Principal Component Analysis (PCA). This step helped make the training process more efficient by reducing each feature vector to 50 components.

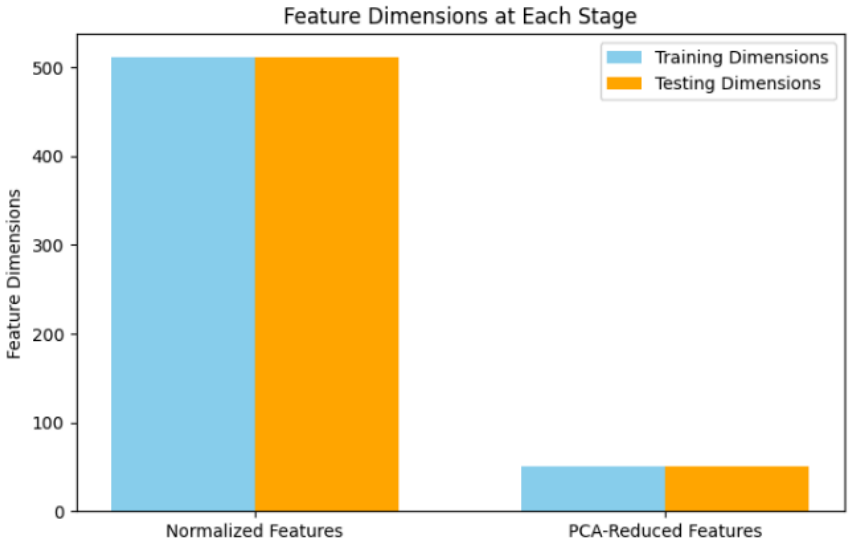


Figure 2: Feature dimensions at preprocessing stages. The initial 512 dimensions from ResNet-18 are reduced to 50 using PCA, streamlining the Naive Bayes training process.

The chart above highlights the reduction in feature dimensions during preprocessing. The ResNet-18 model extracts 512-dimensional feature vectors, which are then reduced to 50 dimensions using PCA. This step ensures computational efficiency while retaining critical information and preparing the dataset for practical training and testing with the Naive Bayes model.

3. Model Implementations

3.1 Naive Bayes

Overview

Naive Bayes was one of the first models we implemented in this project. Its simplicity helped us understand classification and probability-based predictions, providing valuable insights into using statistical assumptions for machine learning tasks. Even though its feature independence assumption is often unrealistic, it serves as a good starting point for understanding classification algorithms.

This model allowed us to build a solid foundation and set a baseline for comparison with the other, more advanced models in our project.

Implementation Approach

At this stage, with the dataset preprocessed as outlined in the General Preprocessing Section, we implemented two versions of Naive Bayes:

1. Manual Implementation:

For the manual implementation of Naive Bayes, we followed the algorithm's mathematical foundations to classify the CIFAR-10 dataset. The steps included:

- Calculating Priors: Determining the proportion of training samples belonging to each class to compute class probabilities.
- Calculating Likelihoods: Using the Gaussian probability density function to calculate the likelihood of each feature given the class.
- Calculating Posteriors: Applying Bayes' theorem to compute the posterior probabilities for each class and assigning the label with the highest posterior.

This process allowed us to understand how the model uses statistical measures like means, variances, and probabilities to make predictions. It also highlighted the importance of preprocessing steps like PCA, which reduced the dimensionality of the input and simplified the computations. The confusion matrix in Figure 3 below shows how the manual Naive Bayes performed. Diagonal values represent correct predictions, while off-diagonal values show errors. This model struggled with similar classes like 'cat' and 'dog.'

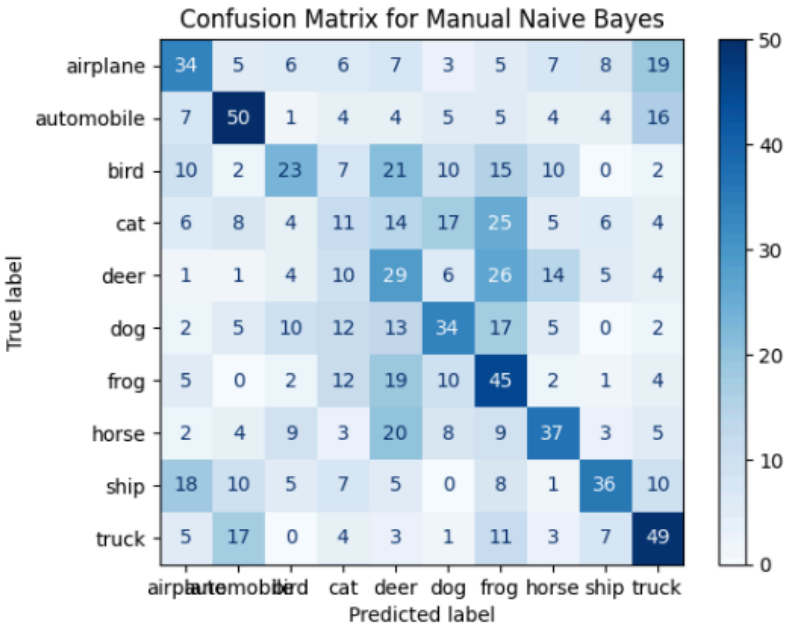


Figure 3: Confusion Matrix for Manual Naive Bayes

1. Scikit-learn Implementation:

For the Scikit-learn implementation, we used the GaussianNB class from the Scikit-learn library to classify the CIFAR-10 dataset. This implementation provided a quick and reliable way to validate the results of our manual Naive Bayes classifier. The key steps included:

- Model Initialization: Using the GaussianNB class.
- Training: Fitting the model on the preprocessed training dataset with PCA-reduced feature vectors.
- Testing: Evaluating the model on the test dataset to measure its accuracy and generate performance metrics.

The Scikit-learn implementation offered the advantage of being optimized for speed and accuracy, allowing us to focus on comparing its performance with our manual implementation. This comparison helped us understand the strengths and weaknesses of a manual versus library-based approach to Naive Bayes classification.

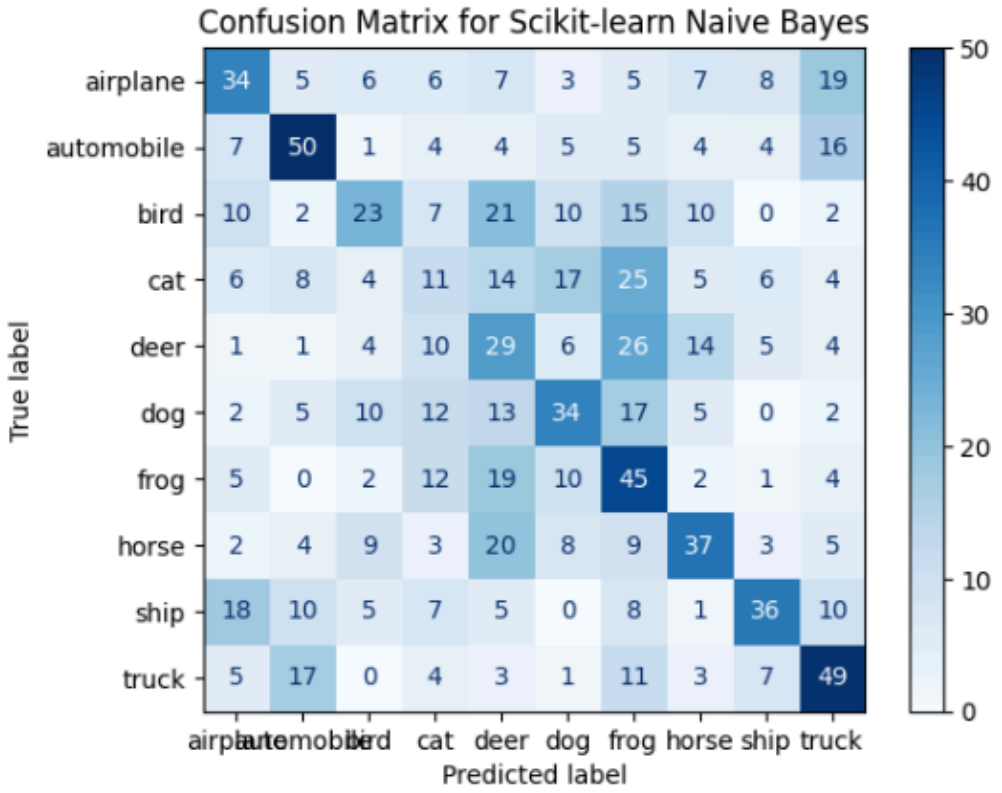


Figure 4: Confusion Matrix for Scikit-learn Naive Bayes

The confusion matrix in Figure 4 highlights strengths and weaknesses similar to those of manual implementation.

For both implementations, we trained the model on the reduced dataset of 5000 training samples across 10 classes and evaluated it on the test dataset of 1000 samples. This dual approach gave us a hands-on understanding of how Naive Bayes works while also allowing us to validate our results.

Accuracy and Training Time Analysis

Both implementations achieved an accuracy of 35%, confirming the correctness of our manual implementation. While this accuracy is low, it shows the complexity of the CIFAR-10 dataset and the limitations of the Naive Bayes model's feature independence assumption. The model performed well in classes like 'automobile' and 'ship' but struggled with visually similar categories such as 'cat' and 'dog.'

The manual implementation took 0.29 seconds, while Scikit-learn completed training in just 0.02 seconds. This highlights the difference between manual coding for deeper understanding and the speed of library-based implementations.

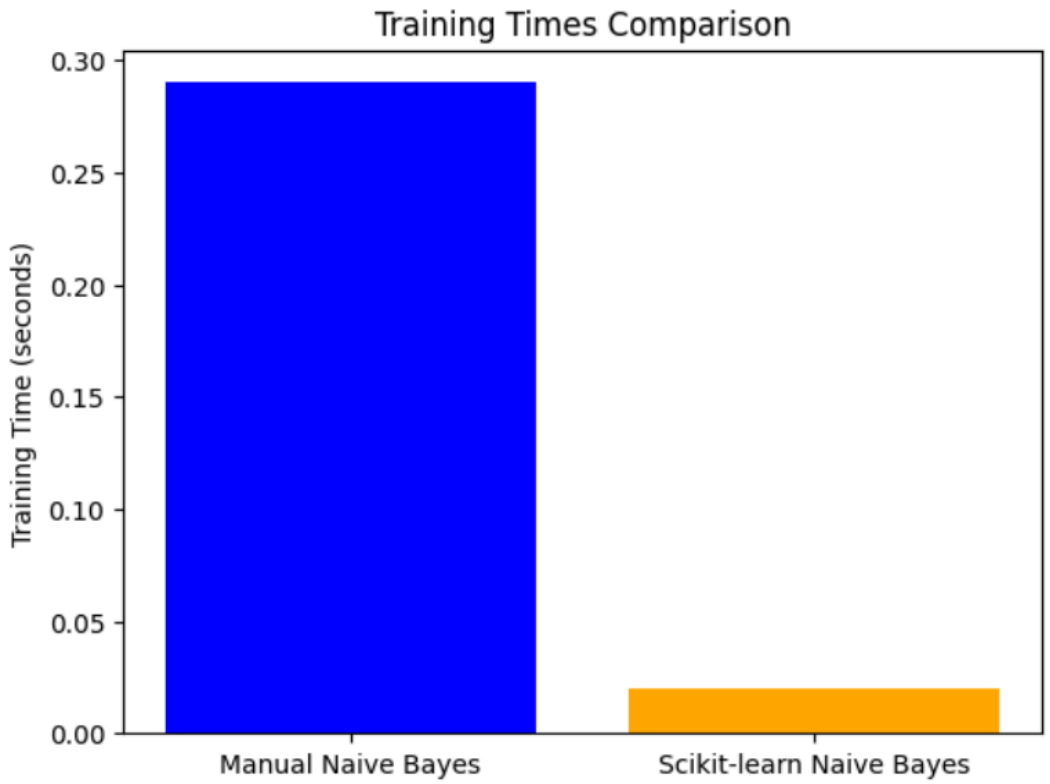


Figure 5: Training time comparison for Manual and Scikit-learn Naive Bayes, showing Scikit-learn's faster performance due to optimization.

Comparison Between Manual and Scikit-learn Implementations

Metric	Manual Naive Bayes	Scikit-learn Naive Bayes
Accuracy	35%	35%
Training Time	Slower	Faster

Naive Bayes gave us a good starting point for understanding classification and the challenges of the CIFAR-10 dataset. The manual and Scikit-learn versions showed that while Naive Bayes is simple and efficient, it struggles with complex datasets due to its feature independence assumption. This highlights the need for more advanced models to capture deeper patterns in the data.

3.2 Decision Tree

Overview

The second model we implemented was the Decision Tree. Unlike Naive Bayes, Decision Trees are more flexible as they do not assume feature independence. They split the dataset recursively based on feature thresholds, creating a tree-like structure to model complex, non-linear relationships.

While Decision Trees are easy to interpret and widely used, they have some challenges, such as a tendency to overfit the training data, especially with small or noisy datasets. We implemented the Decision Tree model using two approaches with the preprocessed dataset outlined in the General Preprocessing Section.

Implementation Approach

1. Manual Implementation:
   We recursively split the dataset by evaluating features and thresholds for our custom Decision Tree to find the best split that minimized Gini impurity. The recursion stopped when the maximum depth was reached, the subset became pure, or the subset size was too small to split further. This process enhanced our understanding of how splits, stopping conditions, and overfitting influence model performance.

2. Scikit-learn Implementation:

   We also implemented the Decision Tree using Scikit-learn's DecisionTreeClassifier. The key steps included:

   - Model Initialization: Usage of DecisionTreeClassifier with parameters such as max_depth and min_samples_split to control overfitting.
   - Training: Fit the model on the preprocessed training dataset.
   - Testing: Evaluate the model on the test dataset to measure its accuracy and generate a confusion matrix.

Confusion Matrices:

Figures 6 and 7 visualize the performance of both implementations, showing the confusion matrices for the manual and Scikit-learn implementations, respectively. These matrices illustrate the model's strengths and weaknesses, like its ability to classify specific categories accurately while struggling with others.

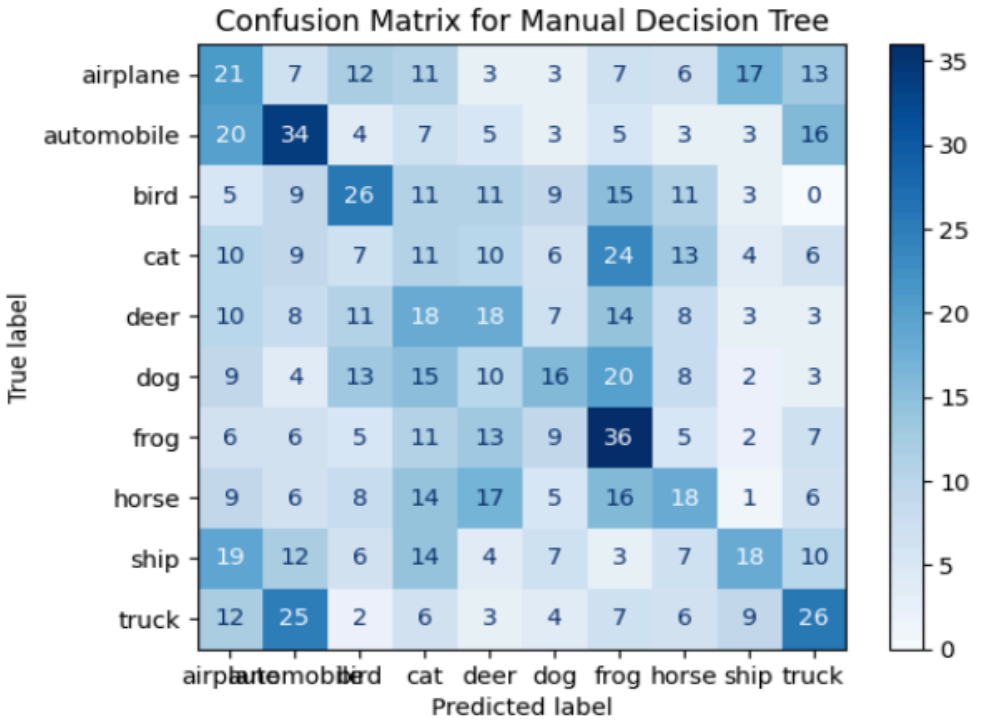


Figure 6: Confusion Matrix for Manual Decision Tree.

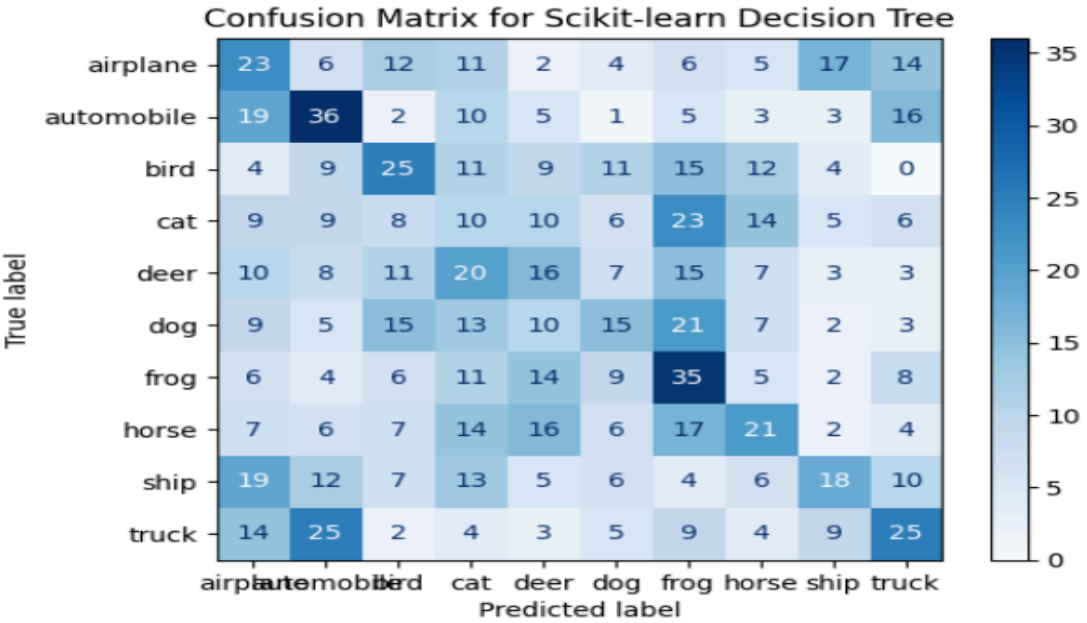


Figure 7: Confusion Matrix for Scikit-learn Decision Tree.

Figures 6 and 7 show the performance of the manual and Scikit-learn Decision Tree models. Like Naive Bayes, both models performed well for distinct classes but struggled with overlapping classes. While the confusion matrices share some patterns with Naive Bayes, Decision Trees' ability to model non-linear relationships provides slightly better flexibility for handling complex data.

Accuracy and Training Time Analysis

Both the manual and Scikit-learn Decision Tree implementations achieved an accuracy of **22%** on the test dataset. This modest performance reflects the complexity of the CIFAR-10 dataset and the challenge of distinguishing visually similar classes, especially with a model like the Decision Tree, which tends to overfit small or noisy datasets.

Training Time:

- Manual Decision Tree: Training took 256.24 seconds, significantly longer than the Scikit-learn implementation. This is due to the manual computation of splits and the implementation's recursive nature.
- Scikit-learn Decision Tree: Training completed in just 0.53 seconds, demonstrating the efficiency of optimized library functions.

Prediction Time:

- Manual Decision Tree: Prediction time was 0.01 seconds, comparable to Scikit-learn's performance.
- Scikit-learn Decision Tree: Prediction time was 0.00 seconds, highlighting its speed advantage for inference tasks.

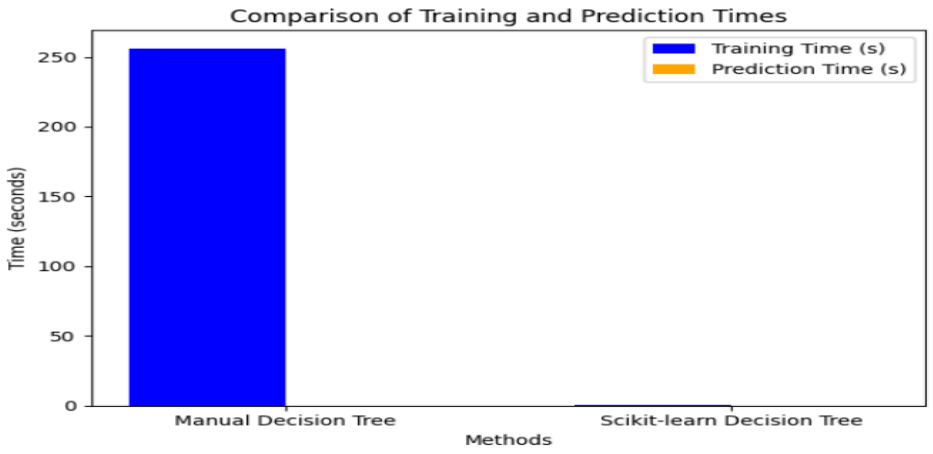


Figure 8: Training and Prediction Times for Manual and Scikit-learn Decision Tree Implementations.

Overall, the Decision Tree implementations showed how non-linear relationships could be modelled without assuming feature independence. However, they also highlighted challenges like overfitting and slow training times in the manual version. This reinforces the value of using optimized libraries like Scikit-learn for efficient machine learning, while manual implementations help understand the basics.

3.3 Multi-Layer Perceptron (MLP)

Overview

The preprocessing steps for all the models except CNN are performed in steps highlighted in section section 2. MLP is short for Multi Layer Perceptron, it is a neural network composed of multiple layers of interconnected neurons. Each of these neurons are further connected to other neurons and work capturing details in each layer. MLP is a feedforward neural network that maps input data to the output labels or values by learning the relationship in the data through supervised learning.

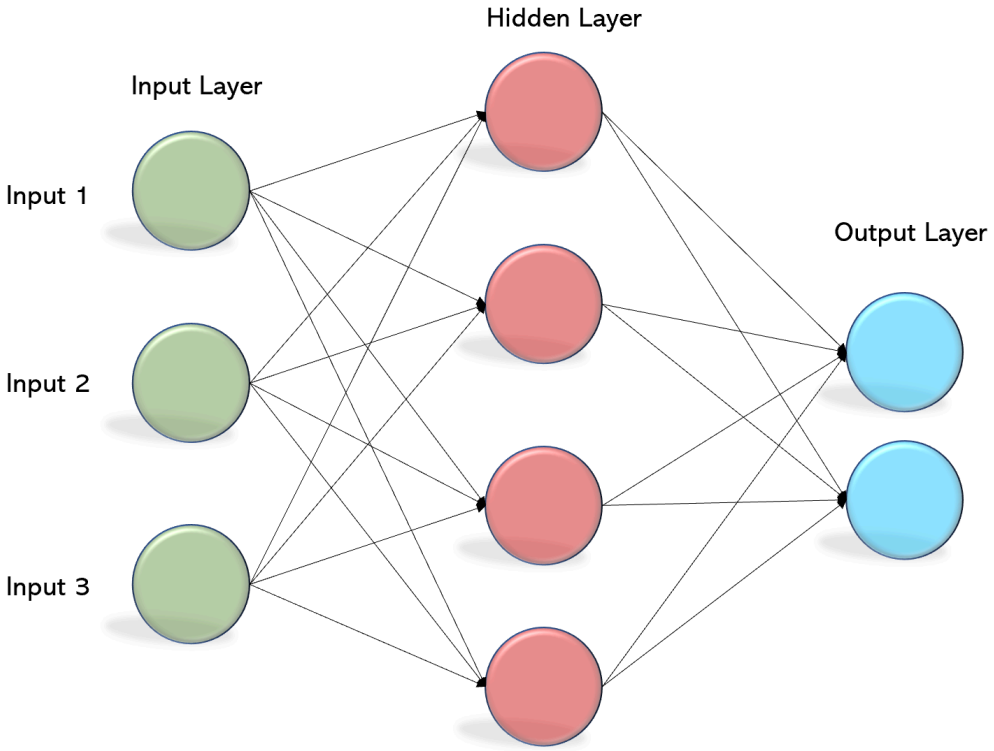


Figure 9: Basic MLP model [1]

On the higher level, it comprises three layers: the input layer, the hidden layer, and the output layer. The input is a vector of input features, and the hidden layers are the intermediate layers with neurons that compute the transformations. The output is a vector of the predictions. Each neuron in one layer is connected to every neuron in the next layer, which is called the fully connected layer. Non-linear functions are applied at each neuron to introduce non-linearity and learn the features of the complex patterns. The objective of the model is to learn effectively. This is underlined by minimizing the loss function; in our case, the Cross-Entropy Loss function is normally used for classification tasks. As MLP accepts vectors of input.[1]

Let's analyze the base MLP model provided to us. The first fully connected layer takes 50 input features (processed and passed by PCA) and outputs 512 neurons.
We have another fully connected layer in the second layer with 512 input features and 512 output neurons. This layer transforms the output of the first layer into a new representation while performing the same operation. Batch normalization is then applied to the layer. Batch

Normalization normalizes the activations from the linear layer across the whole batch. This helps stabilize the training and helps our gradient convergence. The RelU activation function is applied again, enabling the layer to model the transformations.

The final layer of the model takes in 512 input features and produces ten output neurons. This layer maps the 512-dimensional representation from the previous layer to a 10-dimensional output(our classes).

Architecture

The models and their configurations are highlighted in the table below. The goal of this exploration is to assess how the depth of the model and layer depth affect the predictions. Therefore, V1 has two additional layers, and V2 has one less layer than our base model. On the other hand, V3 and V4 have the same architecture as our Base model. However, the number of neurons decreased and increased respectively. This approach lets us train each model and get their respective metrics for further evaluation.

Model	Number of Linear Layers	Input Layer	Neurons in Layer 2	Neurons in Layer 3	Neurons in Layer 4	Output Layer
Base Model	3	(in=50, out= 512)	(in=512, out= 512)	-	-	(in=512, out=10)
MLP_V1	5	(50,512)	(512,512)	(512,512)	(512,512)	(512, 10)
MLP_V2	2	(50, 512)	-	-	-	(512. 10)
MLP_V3	3	(50, 256)	(50, 256)	-	-	(256, 10)
MLP_V4	3	(50, 1024)	(1024, 1024)	-	-	(1024, 10)

Table 1: Model Variations for MLP

Training Methodology:

In the MLP model's training process, we run the models for 100 epochs. The learning rate is 0.01. The optimizer used for MLP is SGD with a momentum of 0.9. The objective function of this model is the Categorical Cross Entropy, as our goal is to perform classification. We modify the baseline model to assess how the depth of the model(the number of hidden layers) and the size of the hidden layers(the number of neurons in the hidden layers affect the model).

Model Outputs and Evaluation

1. MLP (Baseline Model)

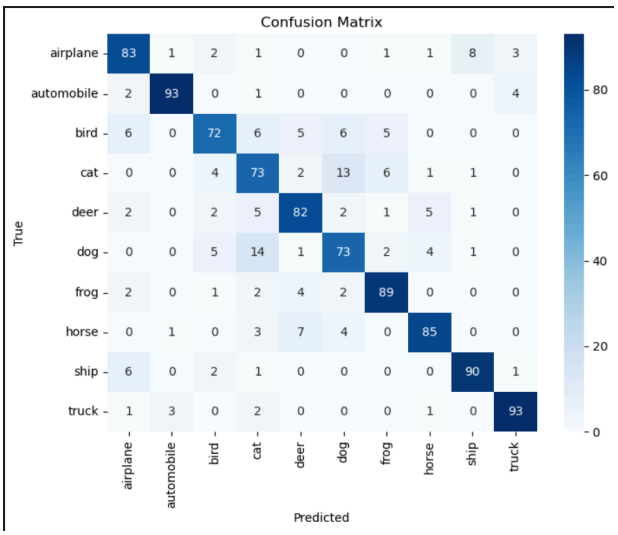


Figure 10:Confusion Matrix for MLP

```
Test Loss: 0.47554782032966614
Classification Report:
              precision    recall  f1-score   support

    airplane       0.81      0.83      0.82       100
  automobile       0.95      0.93      0.94       100
        bird       0.82      0.72      0.77       100
         cat       0.68      0.73      0.70       100
        deer       0.81      0.82      0.82       100
         dog       0.73      0.73      0.73       100
        frog       0.86      0.89      0.87       100
       horse       0.88      0.85      0.86       100
        ship       0.89      0.90      0.90       100
       truck       0.92      0.93      0.93       100

    accuracy                           0.83      1000
   macro avg       0.83      0.83      0.83      1000
weighted avg       0.83      0.83      0.83      1000
```

Figure 11:Classification report for MLP

The base model performs well, obtaining an accuracy of 83%. The cross-entropy loss of 0.475 indicates that the model is close to the true labels. This highlights that the model makes correct predictions but is not always very confident about these predictions. In terms of classes, the automobiles are best in terms of accuracy, f1 score and recall. On the other hand, the class of cats was the hardest for the model to predict. This could be because of the distinct features in classes(ships, automobiles), which make it easier for the model to learn. The macro average for all the metrics is the same, which is 83%. The confusion matrix with diagonal intensity shows that the model predicts the classes correctly. However, there is a misclassification between dogs and cats; 14 dogs were predicted as cats and 13 cats were predicted as dogs. This is possible due to similar-looking features.

2. MLP_V1

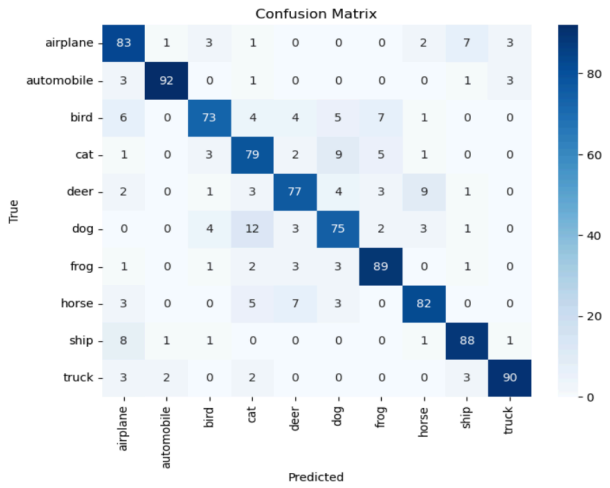


Figure 12:Confusion Matrix for MLP_V1

```
Test Loss: 0.5093013048171997
Classification Report:
              precision    recall  f1-score   support

    airplane       0.75      0.83      0.79       100
  automobile       0.96      0.92      0.94       100
        bird       0.85      0.73      0.78       100
         cat       0.72      0.79      0.76       100
        deer       0.80      0.77      0.79       100
         dog       0.76      0.75      0.75       100
        frog       0.84      0.89      0.86       100
       horse       0.83      0.82      0.82       100
        ship       0.86      0.88      0.87       100
       truck       0.93      0.90      0.91       100

    accuracy                           0.83      1000
   macro avg       0.83      0.83      0.83      1000
weighted avg       0.83      0.83      0.83      1000
```

Figure 13:Classification report for MLP_V1

This MLP_V1 model (extra layers) obtained an accuracy of 83% and a test loss of 0.51. The macro average and weighted average for precision, recall, and F1-score are 83%. The model

performs exceptionally well for classes like automobiles, frogs, and ships. This is indicated by high precision and recall. Precision defines how correct we are, and recall means how often our model identifies true positives. The model struggles with the cat class (F1 = 76%) and the dog class (F1 = 75%). The F1-score averages precision and recall metrics. The models struggle in the cat and dog classes because of their similar features. The confusion matrix backs this up as 12 dogs are misclassified as cats, and nine cats are misclassified as dogs.

3. MLP_V2

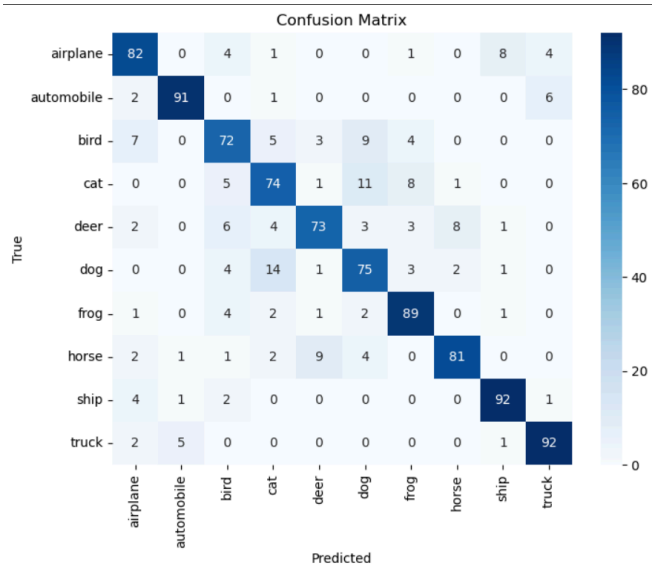


Figure 14:Confusion Matrix for MLP_V2

```
Test Loss: 0.514582097530365
Classification Report:
              precision    recall  f1-score   support

    airplane       0.80      0.82      0.81       100
  automobile       0.93      0.91      0.92       100
        bird       0.73      0.72      0.73       100
         cat       0.72      0.74      0.73       100
        deer       0.83      0.73      0.78       100
         dog       0.72      0.75      0.74       100
        frog       0.82      0.89      0.86       100
       horse       0.88      0.81      0.84       100
        ship       0.88      0.92      0.90       100
       truck       0.89      0.92      0.91       100

    accuracy                           0.82      1000
   macro avg       0.82      0.82      0.82      1000
weighted avg       0.82      0.82      0.82      1000
```

Figure 15:Classification report for MLP_V2

The MLP_V2 (Fewer layers than the base model) obtained an accuracy of 82%, while the test loss is 0.51. This shows that our model is pretty accurate and contains a few errors. The macro average for precision, recall and f1-score is 82%.The model performed well in the classes: automobile(F1 =92%), truck(F1= 91%) and ship(F1= 90%). However, it struggles with the cat class (F1 = 73%) and, to a lesser extent, the dog class (F1 = 74%). There seems to be a trend in the classes performing well and badly. These weaknesses suggest that the model has difficulty distinguishing among classes with subtle differences. The confusion matrix portrays the misclassification, where 14 dogs are misclassified as cats, and 11 are misclassified the other way around.

4. MLP_V3

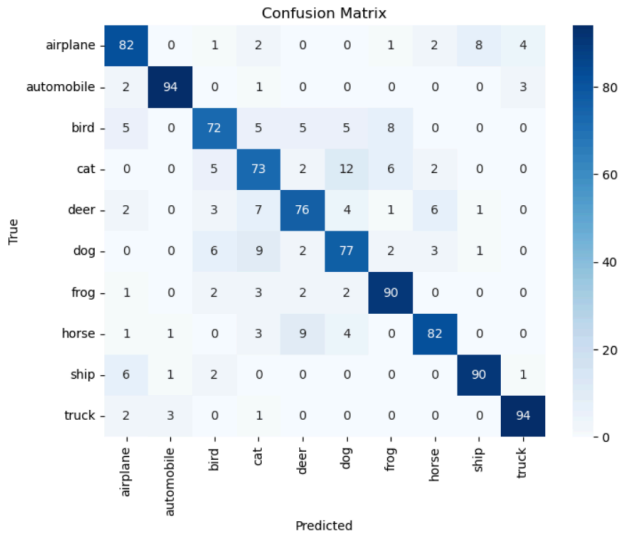


Figure 16:Confusion Matrix for MLP_V3

```
Test Loss: 0.515907883644104
Classification Report:
              precision    recall  f1-score   support

    airplane       0.81      0.82      0.82       100
  automobile       0.95      0.94      0.94       100
        bird       0.79      0.72      0.75       100
         cat       0.70      0.73      0.72       100
        deer       0.79      0.76      0.78       100
         dog       0.74      0.77      0.75       100
        frog       0.83      0.90      0.87       100
       horse       0.86      0.82      0.84       100
        ship       0.90      0.90      0.90       100
       truck       0.92      0.94      0.93       100

    accuracy                           0.83      1000
   macro avg       0.83      0.83      0.83      1000
weighted avg       0.83      0.83      0.83      1000
```

Figure 17:Classification report for MLP_V3

The model achieved an overall accuracy of 83%, with macro and weighted averages of precision, recall, and F1-score also at 83%. This highlights consistent performance through all the classes. The high-performing classes include "automobile" (F1-score: 94%) and "truck" (F1-score: 93%). The same classes with distinct features continue to perform well. However, the model struggles with "cat" and "bird," which have the lowest F1 scores (72% and 75%, respectively). The relatively low recall for "cat" (73%) suggests the model misses many true positives in this category. The confusion matrix highlights the misclassification where 12 cats are misclassified as dogs, and birds have been falsely classified as dogs, deer and cats.

5. MLP_V4

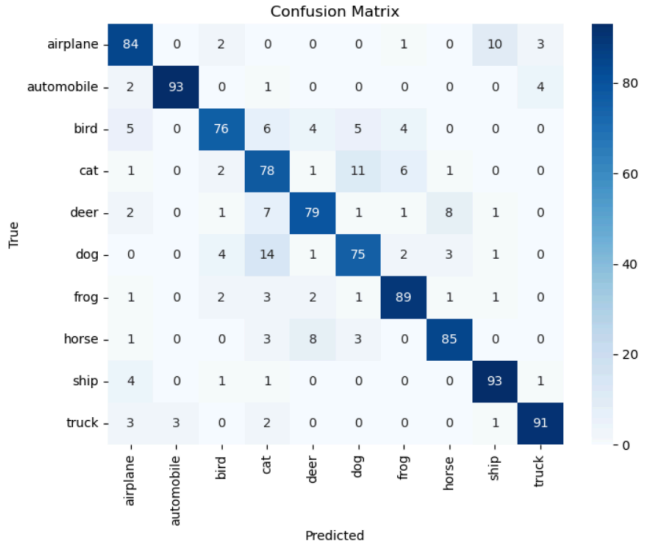


Figure 18:Confusion Matrix for MLP_V4

```
Test Loss: 0.4715401828289032
Classification Report:
              precision    recall  f1-score   support

    airplane       0.82      0.84      0.83       100
  automobile       0.97      0.93      0.95       100
        bird       0.86      0.76      0.81       100
         cat       0.68      0.78      0.73       100
        deer       0.83      0.79      0.81       100
         dog       0.78      0.75      0.77       100
        frog       0.86      0.89      0.88       100
       horse       0.87      0.85      0.86       100
        ship       0.87      0.93      0.90       100
       truck       0.92      0.91      0.91       100

    accuracy                           0.84      1000
   macro avg       0.85      0.84      0.84      1000
weighted avg       0.85      0.84      0.84      1000
```

Figure 19:Classification report for MLP_V4

The MLP model V4 has an overall accuracy of 84% and macro/weighted precision averages of 85%, recall, and F1-score at 84%. The highest-performing classes are "automobile" and "truck," each achieving F1 scores of 95% and 91%, respectively, indicating strong distinction and consistent predictions for these categories. However, the model continues to underperform for "cat" (F1-score: 73%) due to low precision (68%), suggesting frequent misclassifications. Similarly, "bird" and "dog" have modest F1 scores of 81% and 77%, showing room for improvement. The confusion matrix supports these scores as 14 dogs are misclassified as cats, and 11 cats are misclassified as dogs. Bird classification is still in multiple classes, such as frogs, dogs, and cats.

Overall analysis for MLP

Computation time:

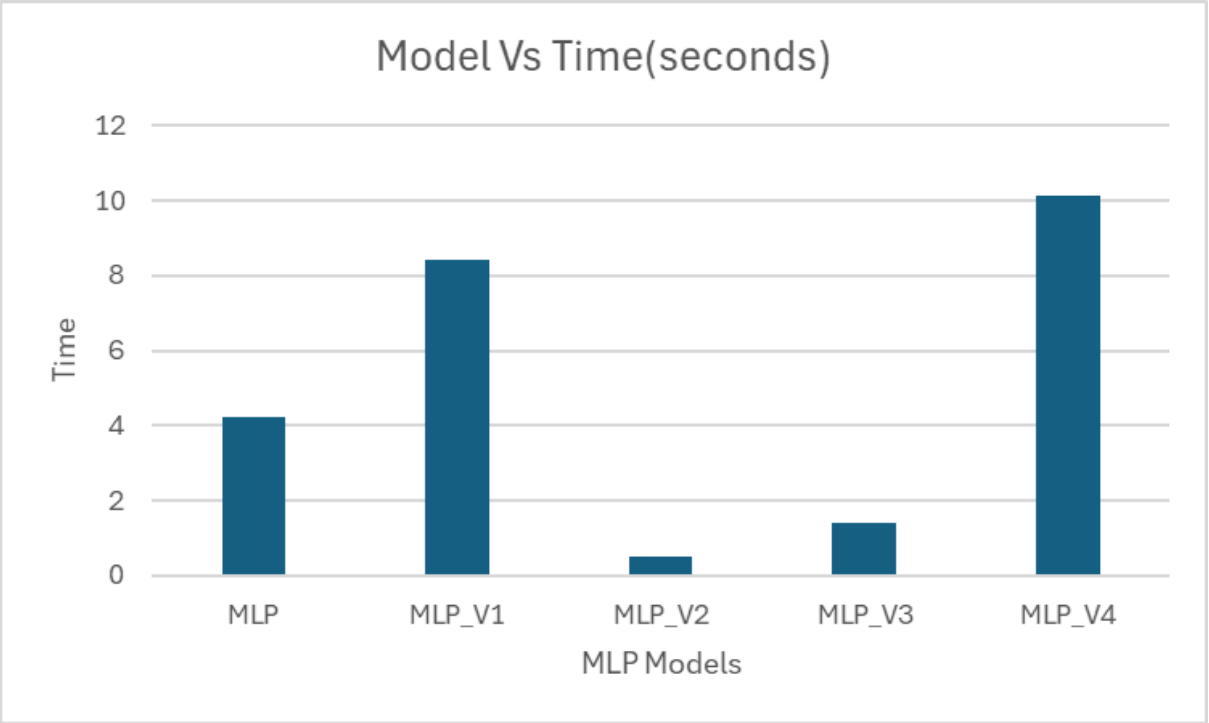


Figure 20: Models vs Training time for MLP

The MLP_V2 model took the lowest time to train the data. This makes sense as it only has the input and output layers. MLP_V4 took the most time because it has 3 fully connected layers, which transform 1024 neurons. The more the weights, the greater the time taken to train the data.

Model	Accuracy	Test Loss	Precision	Recall	F1-score
MLP	0.82	0.48	0.83	0.83	0.83
MLP_V1	0.83	0.50	0.83	0.83	0.83
MLP_V2	0.82	0.51	0.82	0.82	0.82
MLP_V3	0.84	0.52	0.83	0.83	0.83
MLP_V4	0.84	0.47	0.85	0.84	0.84

Table 2: MLP model evaluation metrics

Observations and Discussion

Base Model as the Benchmark

- The Basic MLP model is a benchmark for testing and achieving the best accuracy for our modified models. The model achieved an accuracy of 82%, precision, recall, and F1 Scores of 83%, and a minimal test loss (0.48).

Impact of Modifications

1. Adding/Removing Complexity:
    - Adding extra layers to the basic model improved our accuracy by 1%, a small value to consider having a positive impact. Moreover, the precision, recall, and F1 scores stayed the same. The test loss was also higher(0.50).
    - Removing the layers in MLP_V2 decreased the accuracy by 1%. Moreover, the precision, recall and F1 score decreased by 1%. The test loss was also higher(0.51)
    - Even though adding extra depth to the model increased its accuracy, it was not high enough to make an impact. On the other hand, decreasing the depth of the model degrades its performance. With fewer layers, the model loses the feature processing function, which makes it harder for it to generalize.
2. Adjusting Layers size (Neurons):
    - Decreasing the neurons in the layers from 512 to 256 resulted in an increase in accuracy(acc=84%). However, an increase in test loss is noticed as it goes to 0.52. This suggests that even though the model increases in accuracy, it might be underfitting and not able to capture all the details, making generalization worse.
    - Increasing the neurons, scaling up to 1024, increased the model's accuracy by 84%. Increasing the neuron count enhances the model's capacity to capture the features from the data. A decent test loss of 0.48 was noticed, highlighting that the model is increasing in accuracy while generalizing for unseen data.

In terms of accuracy, there is not much of a difference in the models. However, there is a difference in test loss. MLP_V4 is the best-performing model regarding precision, recall, F1-score, and test loss, making it the optimal choice among the evaluated models. While the performance differences are marginal, the higher precision and lower test loss make MLP_V4 more reliable for deployment. It is noted that the model performs well in classes like automobiles and trucks. The model also performs the worst in animal classes like cats, dogs and birds.

3.4 Convolutional Neural Network (CNN)

Overview

The CNN implementation in this project focuses on training a VGG11 network on the limited CIFAR-10 dataset to perform image classification. Convolutional Neural Networks(CNNs) represent deep-learning algorithms predominantly used to process structured array data, such as images. CNN adopted the idea of MLP while employing convolution on data.

Architecture

Visual Geometry Group(VGG) 11 model takes input as a 224 x 224 pixel image in RGB format (x3). VGG utilizes 3x3 convolutional filters throughout the architecture. This helps the network to capture features on different scales. After every two convolutional layers, max-pooling layers are used for spatial down-sampling. Max-pooling assists in reducing the spatial dimensions while retaining the core features. The convolutional and pooling layers are continued by a few fully connected layers, which act as the final classifier. The fully connected layer uses the learned features to form class predictions. Within the network, the Rectified linear units(RelU) function is used as an activation function to introduce non-linearity, which helps model relationships within the data.[2]

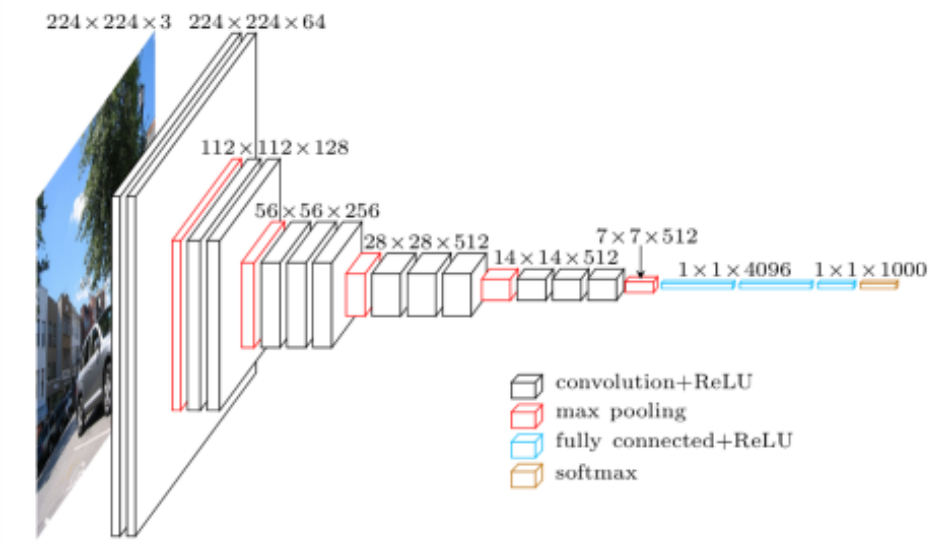


Figure 21: Example of a CNN VGG11 Model [3]

VGG11(Figure 21) is the simplest VGG deep learning model. It has 11 weight layers in total, hence the name VGG11. From the 11 layers, eight convolutional layers, Batch Normalizations, RelU activation functions, and MaxPool layers of kernel size 2 follow these layers. This is followed by three fully connected layers, the RelU activation function and Dropout layers [4]. The RelU function introduces non-linearity and makes the model converge better. On the other hand, the throughput layer prevents overfitting by reducing the complete dependency on neurons. The dropout layer drops 50% of neurons during the training process.

Training Methodology

The hyperparameters in training methodology affect the learning of the model. We decided to use 100 epochs for the CNN model(considering it takes longer to train a model from scratch). The optimization technique used for all the models is Stochastic Gradient Descent(SGD), with a learning rate of 0.01 and momentum of 0.9. SDG processes the data in mini-batches, which helps create some noise in the data to converge better. The aim of training the VGG model and its variations is to deduce how the depth of the model kernel size affects the predictions. It has been noted previously that VGG11 has a deep and complex architecture that allows for limited CIFAR-10 data. Therefore, in addition to the exploration, we will try to change the number of neurons in the fully connected layer [4].

Main Model:

```
- Conv(001, 064, 3, 1, 1) - BatchNorm(064) - ReLU - MaxPool(2, 2)
- Conv(064, 128, 3, 1, 1) - BatchNorm(128) - ReLU - MaxPool(2, 2)
- Conv(128, 256, 3, 1, 1) - BatchNorm(256) - ReLU
- Conv(256, 256, 3, 1, 1) - BatchNorm(256) - ReLU - MaxPool(2, 2)
- Conv(256, 512, 3, 1, 1) - BatchNorm(512) - ReLU
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
- Linear(0512, 4096) - ReLU - Dropout(0.5)
- Linear(4096, 4096) - ReLU - Dropout(0.5)
- Linear(4096, 10)
```

Figure 22: Basic VGG11 model provided by the project guidelines

Figure 22 shows the baseline model of VGG11 architecture used for our CNN classification of CIFAR-10.

VGG11_AddConv: We modify the base VGG11 model by removing two convolutional layers: Conv (64,64, 3, 1, 1) and Conv (128, 128, 3, 1, 1). These convolutions are also followed by batch normalization and RelU functions.

VGG11_LessConv: We modify the base VGG11 model by adding two additional convolutional layers. The first is Conv(512,512, 3, 1, 1) and the second is Conv(512, 512, 3, 1, 1).

VGG11_Kernel2: We modified the basic VGG11 model by making the kernel size two instead of 3. Kernel sizes are important as they determine how features are extracted from the data, which in turn affects the model's capacity to identify the details of the images. Small kernels capture fine details and local patterns. Stacking multiple convolutional layers allows the model to learn hierarchical and complex features. Small kernels also allow the network to retain spatial granularity across layers.

VGG11_Kernel5: We modified the VGG11 model using a kernel size of 5. It is possible that the kernel size of 3 is not enough to retain all the features in the dataset. Small kernel sizes have limited receptive fields, a reduced ability to detect patterns, and a loss of spatial context. Therefore, we try higher kernel sizes to fit the data.

Model Outputs and Evaluation

1. VGG11 Model:

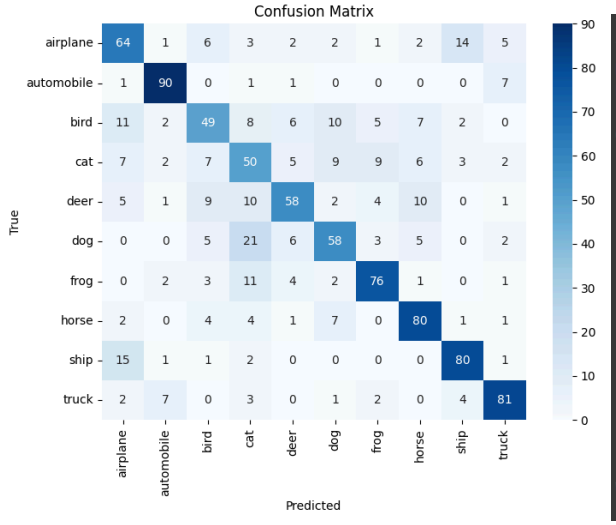


Figure 23:Confusion Matrix for VGG11 base

```
Test Accuracy: 68.60%
Test Loss: 66.6395
Precision: 0.6861
Recall: 0.6860
F1-Score: 0.6844
Classification Report:
              precision    recall  f1-score   support

    airplane       0.60      0.64      0.62       100
  automobile       0.85      0.90      0.87       100
        bird       0.58      0.49      0.53       100
         cat       0.44      0.50      0.47       100
        deer       0.70      0.58      0.63       100
         dog       0.64      0.58      0.61       100
        frog       0.76      0.76      0.76       100
       horse       0.72      0.80      0.76       100
        ship       0.77      0.80      0.78       100
       truck       0.80      0.81      0.81       100

    accuracy                           0.69      1000
   macro avg       0.69      0.69      0.68      1000
weighted avg       0.69      0.69      0.68      1000
```

Figure 24:Classification report for VGG11 base

The base model of our VGG11 obtained an accuracy of 69%. The confusion matrix indicates a diagonal dominance. However, misclassification occurs in classes like cat, dog, deer, and horse. This is particularly because of the similar-looking features. This is also supported by low F1 scores for the classes (cat = 0.47, dog = 0.61, deer = 0.63, bird = 0.53). The best-performing classes are automobiles and trucks backed by strong recall and precision (F1=0.87, 0.81). This is possible because of the distinct features of these classes. The high test loss of 66.64 also indicates that the model is overfitting.

2. VGG11_AddConv(2 extra Convolutional layers):

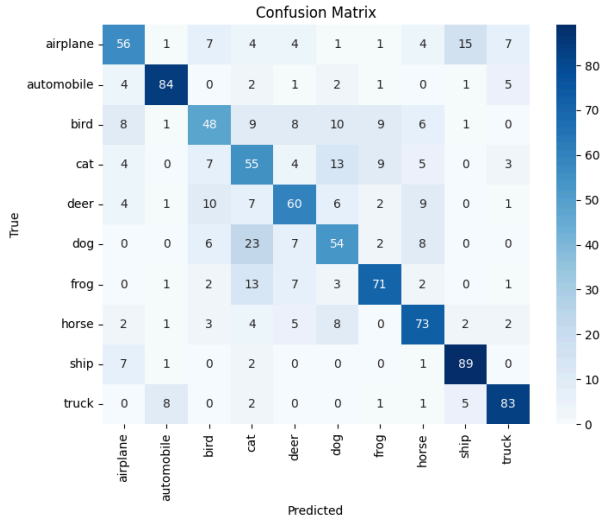


Figure 25:Confusion Matrix for VGG11 AddConv

```
Test Accuracy: 67.30%
Test Loss: 67.3295
Precision: 0.6741
Recall: 0.6730
F1-Score: 0.6717
Classification Report:
              precision    recall  f1-score   support

    airplane       0.66      0.56      0.61       100
  automobile       0.86      0.84      0.85       100
        bird       0.58      0.48      0.52       100
         cat       0.45      0.55      0.50       100
        deer       0.62      0.60      0.61       100
         dog       0.56      0.54      0.55       100
        frog       0.74      0.71      0.72       100
       horse       0.67      0.73      0.70       100
        ship       0.79      0.89      0.84       100
       truck       0.81      0.83      0.82       100

    accuracy                           0.67      1000
   macro avg       0.67      0.67      0.67      1000
weighted avg       0.67      0.67      0.67      1000
```

Figure 26:Classification report for VGG11 AddConv

The modified VGG11 model with extra convolutional layers obtained an accuracy of 67%, precision of 67%, recall of 67% and F1-score of 67%. The model performs well in classes like automobiles, ships and trucks, highlighted by the higher F1 scores. However, the model still struggles to identify animal classes such as cats, dogs, and birds(F1 score= 0.5, 0.55, 0.53, respectively). The confusion matrix underlines these challenges, showing the misclassification of dogs and birds. The test loss for this model is 67.33, which means that the model still struggles to identify the classes correctly. The reason is that it tends towards overfitting, as the VGG11 model contains complex layers.

3. VGG11 Less Convolutional layer:

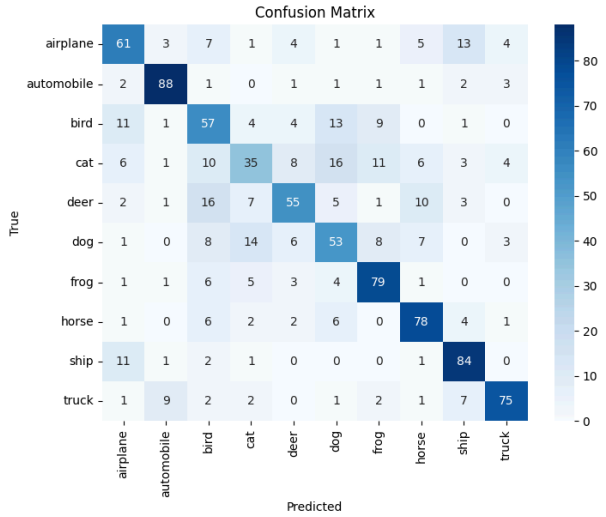


Figure 27:Confusion Matrix for VGG11 Less Conv

```
Test Accuracy: 66.50%
Test Loss: 73.9570
Precision: 0.6614
Recall: 0.6650
F1-Score: 0.6600
Classification Report:
              precision    recall  f1-score   support

    airplane       0.63      0.61      0.62       100
  automobile       0.84      0.88      0.86       100
        bird       0.50      0.57      0.53       100
         cat       0.49      0.35      0.41       100
        deer       0.66      0.55      0.60       100
         dog       0.53      0.53      0.53       100
        frog       0.71      0.79      0.75       100
       horse       0.71      0.78      0.74       100
        ship       0.72      0.84      0.77       100
       truck       0.83      0.75      0.79       100

    accuracy                           0.67      1000
   macro avg       0.66      0.67      0.66      1000
weighted avg       0.66      0.67      0.66      1000
```

Figure 28:Classification report for VGG11 Less Conv

The performance of the modified VGG11 model with fewer convolutional layers achieved an accuracy of 66.5%, precision of 66%, recall of 67% and F1_score of 66%. The best-performing classes are automobile and ship with F1 scores of 86% and 77,%

respectively. The worst-performing classes include cat, bird and dog, with F1 scores of 41%, 53% and 53%, respectively. The confusion matrix highlights notable misclassifications, such as a bird being confused with an airplane and a cat with a dog class. On the other hand, cats are misclassified as dogs, birds and frogs. Removing layers from VGG11 seems to have reduced the model's ability to extract complex features. These features mainly differentiate similar-looking classes.

4. VGG11 Kernel2:

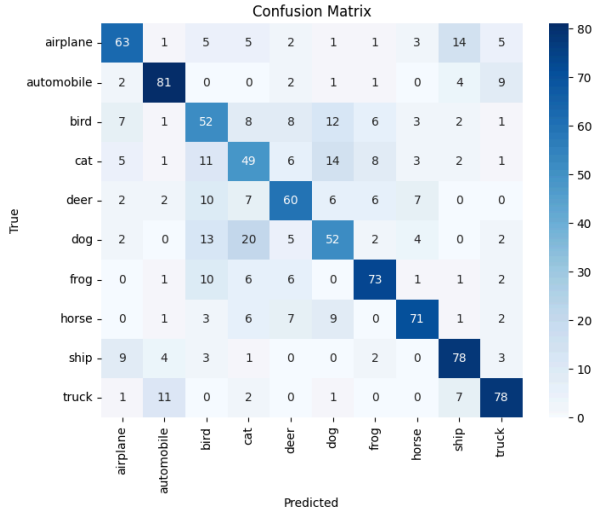


Figure 29:Confusion Matrix for VGG11 Kernel size=2

```
Test Accuracy: 65.70%
Test Loss: 79.7481
Precision: 0.6585
Recall: 0.6570
F1-Score: 0.6572
Classification Report:
              precision    recall  f1-score   support

    airplane       0.69      0.63      0.66       100
  automobile       0.79      0.81      0.80       100
        bird       0.49      0.52      0.50       100
         cat       0.47      0.49      0.48       100
        deer       0.62      0.60      0.61       100
         dog       0.54      0.52      0.53       100
        frog       0.74      0.73      0.73       100
       horse       0.77      0.71      0.74       100
        ship       0.72      0.78      0.75       100
       truck       0.76      0.78      0.77       100

    accuracy                           0.66      1000
   macro avg       0.66      0.66      0.66      1000
weighted avg       0.66      0.66      0.66      1000
```

Figure 30:Classification report for VGG11 Kernel size=2

The VGG11 model with kernel size 2 achieved an accuracy of 65.7%, precision of 66%, recall of 66% and F1-score of 66%. The model performs similarly to the previous models with slight variations in class-wise metrics. The trend follows as non-animal classes perform the best. The automobile(F1 score= 88%) and truck(F1-score= 77%) perform the best. The worst-performing classes are the cat and the dog classes, with F1-scores of 0.48 and 0.53, respectively. Compared to the previous results, the model with kernel size 2 performs worse in the challenging classes. This might be due to the smaller kernel size limiting the model's ability to capture complex spatial hierarchies.

5. VGG11 Kernel 5:

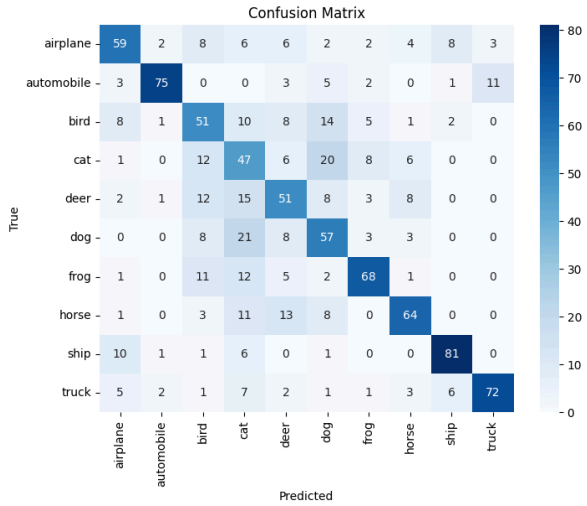


Figure 31:Confusion Matrix for VGG11 Kernel size=5

```
Test Accuracy: 62.50%
Test Loss: 77.9411
Precision: 0.6492
Recall: 0.6250
F1-Score: 0.6340
Classification Report:
              precision    recall  f1-score   support

    airplane       0.66      0.59      0.62       100
  automobile       0.91      0.75      0.82       100
        bird       0.48      0.51      0.49       100
         cat       0.35      0.47      0.40       100
        deer       0.50      0.51      0.50       100
         dog       0.48      0.57      0.52       100
        frog       0.74      0.68      0.71       100
       horse       0.71      0.64      0.67       100
        ship       0.83      0.81      0.82       100
       truck       0.84      0.72      0.77       100

    accuracy                           0.62      1000
   macro avg       0.65      0.63      0.63      1000
weighted avg       0.65      0.62      0.63      1000
```

Figure 32:Classification report for VGG11 Kernel size=5

The modified VGG11 model with a kernel size of 5 obtained an accuracy of 62.5%, precision of 65%, recall of 63% and F1-score of 63%. This model performs less than previous models with kernel sizes 2 and 3. The confusion matrix shows the same trend: while classes like automobiles(75% recall) and trucks(72% recall) perform well, the model struggles with classes like cats(47% recall) and birds(51% recall). Cats are heavily misclassified as dogs and birds. On the other hand, the birds show confusion with cats and dogs. The model's sharp decrease in accuracy shows that a larger kernel size led to a loss of spatial details. This, in turn, negatively affected the feature extraction for challenging classes. Automobile and truck classes continue to perform well as they display distinct features.

Overall Analysis

Computation time:

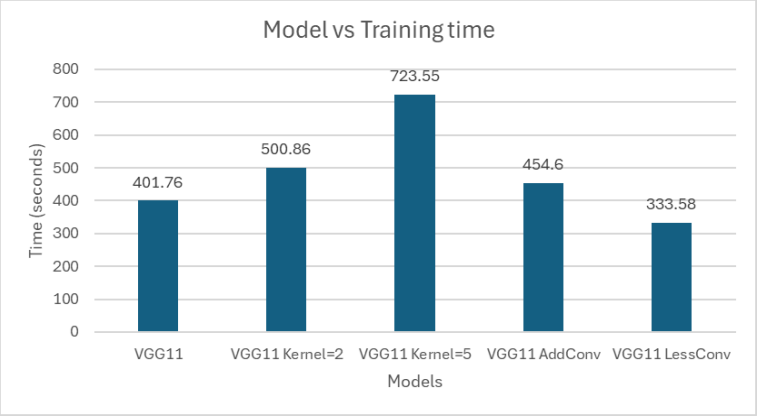


Figure 33: Bar graph comparing models and their training times

The bar graph in Figure 33 compares the training time of our VGG11 models. This highlights the impact of kernel size and depth of the model. The baseline VGG11 model takes 401.76 seconds to train. When the kernel size is increased to 5x5, it results in greater training time(723.55 seconds). This is due to the higher computational resources needed for filtering with the 5x5 kernel. Reducing the kernel to 2x2 is also seen as 500 seconds. This is probably because of more operations for the receptive field. Adding convolutional layers increases time (454.6 seconds), and removing results in faster training(333.58 seconds). These results in training time highlight the trade-off between computational resource efficiency and the specific model's ability to extract features with larger kernels and deeper architectures for a complex task.

Model	Accuracy	Test loss	Precision	Recall	F1-score
VGG11-base model	68.6	66.64	68.61	68.6	68.4
VGG11 AddConv	67.3	67.33	67.41	67.30	67.17
VGG11 LessConv	66.5	73.96	66.14	66.50	66
VGG11 Kernel size=2	65.70	79.75	65.85	65.70	65.72
VGG11 Kernel size=5	62.50	77.94	64.92	62.50	63.40

Table 3: VGG11 models and their metrics

Observations and Discussion

1. Base Model as the Benchmark:
    - The standard VGG11 performs the best across all the models. This is highlighted in Table 3, as it has the highest accuracy, lowest test loss, highest precision-recall, and highest F1 Score. The modifications generally degraded the performance.
2. Impact of Adding/Removing Convolutional Layers:
    - Adding or removing convolutional layers didn't seem to improve the standard VGG11 model's results. However, the modified VGG11 model with extra depth performed slightly better than the one with fewer layers, which indicates that our model's depth is balanced.
3. Kernel Size Impact:
    - The modified VGG11 model with kernel size 2 achieved an accuracy of 66%(65.70). This is less accurate than the base model, which stands at almost 69%. However, it does perform considerably better than the model with kernel size 5. The 2x2 kernel may lack the receptive field to capture enough spatial data details. On the other hand, the 5x5 kernel doesn't focus on the small spatial details in the images. Therefore, 3x3 is a balanced kernel size for our model.
4. Test Loss Trends:
    - Our base model achieves the lowest test loss, which is 66%. The highest increase in test loss is for models with kernel size two and less depth. This further affirms our speculation that the small kernel size does not capture all the spatial details. Additionally, a shallow VGG11 model cannot capture all the features. Hence, there is higher test loss and lousy generalization.

The standard VGG11 model performs the best, with an accuracy of almost 69%. This is followed by the lowest test loss and highest precision, recall and F1-scores. The base model provides a balance between the kernel size and the depth of the model. Additionally, the base model is computationally efficient, as highlighted by the time results in Figure 33. The precision, recall and F1 scores are stable across all the models. A notable observation in all the models was that the models were performing the best for automobiles and trucks.

Additionally, the models struggled with cats, dogs, and birds. A valid reason for this is that automobiles and trucks have distinct spatial features that can be easily captured by the model. However, the features of animals are much more complex. For example, cats and dogs have similar features(legs, whiskers, ears). Therefore, the models struggled to classify them correctly.

4. Conclusion

In conclusion, we implemented and analyzed four machine learning models—Naive Bayes, Decision Tree, Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN)—to classify images from the CIFAR-10 dataset. Each model provided a different perspective on machine learning techniques, ranging from advanced deep learning systems to simple probabilistic methods.

This process enabled us to observe how several models address the challenges posed by the CIFAR-10 dataset, such as overlapping characteristics and the challenge of distinguishing between visually identical classes. Despite providing a foundational understanding, the feature independence assumption limited the usefulness of Naive Bayes. Decision trees were flexible when simulating nonlinear interactions but overfitted in limited datasets. MLP and CNN models achieved higher accuracies and were better suited for complex tasks. It is important to note that all classes struggled to classify the dog, cat and bird classes. The best performance was achieved for automobiles, truck and ship classes. This indicates that the models learn the distinct features efficiently. However, struggle to achieve the spatial details for the animal classes.

MLP obtained the highest accuracy of all the models. There are multiple reasons for this; we extracted the features with PCA, which lets MLP focus on classification without performing feature extraction. We used transfer learning, where the ResNet 18 model was pre-trained on ImageNets data and weight. Therefore, our MLP could quickly adapt to it. On the other hand, our CNN model is learning the data from scratch.

Additionally, VGG11 is a deeper model requiring more data and computational resources to perform feature extraction and classification. Considering the dataset we were using, we lacked data. CIFAR-10 is a low-resolution dataset(32x32), so CNN may not be able to learn the spatial details in this tiny dataset.

One of the most essential lessons acquired was combining efficient libraries such as Scikit-learn with creating models from scratch. The library-based models demonstrated how much faster and more valuable algorithms can be for real-world applications, while hand implementation helped us comprehend how algorithms operate internally.

The challenge we faced was that we opted for a strategy where we preprocessed the data, trained, and evaluated the models in the same notebook. A better strategy would be to preprocess and save the datasets, save each notebook for models and their variations, and run evaluation of all the notebooks in one notebook.

Looking ahead, there's room for improvement. Using data augmentation to increase the variety of training samples could help address overfitting. Testing advanced architectures like ResNet or fine-tuning pre-trained models might also boost performance. Exploring

techniques like hyperparameter tuning and ensemble methods could be another way to push the models further.

Overall, this project provided valuable hands-on experience and strengthened our ability to approach machine-learning tasks methodically. It has prepared us to tackle more challenging problems and make meaningful contributions to the field of artificial intelligence.

5. References

[1] M. Sena, “A Step-by-Step Guide to Implementing Multi-Layer Perceptrons in Python,” Medium, Feb. 13, 2024. https://python.plainenglish.io/building-the-foundations-a-step-by-step-guide-to-implementing-multi-layer-perceptrons-in-python-51ebd9d7ecbe

[2] Atulanand, “VGGNet Complete Architecture,” CodeX, Nov. 20, 2022. https://medium.com/codex/vggnet-complete-architecture-5c6fa801502b

[3] J. Brownlee, “How to Develop a CNN From Scratch for CIFAR-10 Photo Classification,” Machine Learning Mastery, May 12, 2019. https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

[4] K. Simonyan and A. Zisserman, “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION,” Apr. 2015. Available: https://arxiv.org/pdf/1409.1556