COMP472 Section F
Artificial Intelligence

Project Report

Thi Mai Anh Nguyen

40208493

Yihuan Liu

26966462

2024/11/27

Performance of Different AI Models in Image Classification

Abstract

This project explores the performance of various machine learning models in the task of image classification using the CIFAR-10 dataset. Four models—Naïve Bayes, Decision Tree, Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN)—were implemented and evaluated based on their accuracy, precision, recall, and F1 scores. While simpler models like Naïve Bayes and Decision Tree exhibited moderate performance, they struggled with the complexities of high-dimensional image data. In contrast, neural network-based models, particularly MLP, demonstrated superior classification capabilities due to their ability to capture and process intricate patterns. Despite CNN's potential for optimal performance, its outcomes were constrained by limited computational resources and dataset size. These findings emphasize the importance of model selection and parameter optimization in achieving effective image classification, with future improvements focusing on enhancing CNN's capabilities for more complex tasks.

Introduction

Artificial intelligence (AI) is widely used in complex task processes these days. One of its most common applications is image classification [1,2,3]. This process involves assigning a label to an input image based on its visual content, and is crucial in technologies such as facial recognition, auto-driving vehicles, security surveillance etc. Therefore, improving AI's ability to correctly classify images has become the focus of contemporary studies.

In order to label an image, the AI needs to learn the distinctive features of various objects. There are abundant different algorithms that help AI to achieve this learning process, and these algorithms are called models. The general AI training process involves training dataset selection, data pre-processing, training the AI, and evaluation of the training outcomes.

The dataset we used in this project is Canadian Institute for Advanced Research (CIFAR)-10. This dataset is commonly used in machine learning and computer vision algorithms [4], since it consists of an enormous number of images. Before starting training the AI, the images need to be pre-processed. This is because the machine can only understand binary machine language, while an image contains rich spatial information of pixels. Therefore, pre-processing will convert input data into a format that AI understands.

After preparing the training set, the AI begins to study the knowledge using some mathematical algorithms. The choice of a specific algorithm depends on the expected outcomes modellers are looking for, and the nature of the dataset itself. Simple dataset and requests may only need simple algorithms such as linear regression, K-Nearest Neighbors etc., while more complex datasets require more complicated and powerful algorithms such as Neural Network.

Once the training is done, we need to evaluate the performance of the model. By convention, we usually split our dataset into two parts: training set and testing set. The ratio between the two sets can vary. A higher ratio such as 9:1 may provide AI more materials to learn, while a lower ratio such as 6:4 uses a balanced portion, so the testing results are more convincing. The specific choice of the ratio depends on the actual requirements of the task.

This project uses CIFAR-10 as the training set, comparing the performance of four different models, Naïve Bayes, Decision Tree, Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN), to find the optimal model in image classification. The following sections will first explain the methodology of the training, and then discuss the results of different model types.

Methods

Pre-processing

The image dataset used in this project is CIFAR-10, which is a classic dataset for machine learning. It consists of 60,000 32x32 color images in 10 classes [4]. In this project, we used the first 500 training images as our training set, and first 100 test images of each class as our testing set.

In this project, we used the first 500 training images as our training set, and first 100 test images of each class as our testing set.

After loading the dataset, we reduced the size of the dataset to 50 x 1, since Naïve Bayes, decision trees, and MLPs do not have a good performance in processing high-dimension color images. Meanwhile, CNN is suitable for feature extraction in RGB images, so CNN model uses the original image set with the size of 512 x 1.

Model Architectures

In order to get a better understanding of the performance of the model, we implemented customized Naïve Bayes, Decision Tree and MLP models using basic Python, Numpy and Pytorch libraries. Then we also ran the dataset using Scikit's implementation of Gaussian Naïve bayes classifier and DT to compare their effectiveness. Meanwhile, we also varied the depth and size of the tree or layers in MLP & CNN to explore how these parameters influence models' performance.

1. Naïve Bayes

The Naïve Bayes model is based on the Bayes' Theorem, which assumes feature independence, and uses Gaussian distribution to define the condition probability when features are continuous.

During the training phase, we need to prepare our feature matrix X, and class label y. Then, we extract all possible class labels from y. Then we need to calculate the important feature values of each class. First, calculate the mean and variance of each feature in a specific class c, denoting them by μ_{cj} and σ^2_{cj} respectively. Then, calculate the prior (which is an initial estimation of the probability of each class) for each class. (*Nc is the number of samples belonging to class c, and N is the total number of samples):

$$P(c) = \frac{Nc}{N}$$

Next, we need to compute the likelihood of the features values under Gaussian distribution for each class using Gaussian density equation:

$$P(xi\,|c) = -\frac{1}{\sqrt{2\pi\sigma^2_{cj}}} \exp\left(-\frac{(x_i - \mu_{cj})^2}{2\sigma^2_{cj}}\right)$$

Then we need to compute the posterior probability to determine the most probable class for a given input. In order to reduce the computing complexity, we need to do a log transformation to calculate the conditional probability:

$$\log\, P(c \mid x) = \log P(c) + \sum_{j=1}^{n} \log P(x_i\,|c) + constant$$

The model will calculate the posterior for each class, and eventually place the input to the class with the highest posterior probability.

2. Decision Tree

The core principle of our decision tree model is to recursively partition the dataset based on feature threshold, so that the Gini impurity index will be minimized while Gain is maximized.

The loss function used in our DT model is Gini impurity, which measures the probability of misclassifying a randomly selected element from a dataset. To calculate the Gini index at a particular node, we need to find the complement of the summation of the proportion of samples of class i at node t for all classes:

$$Gini(t) = 1 - \sum_{i=1}^{C} p_i^2$$

By default, we set the max depth of our DT model as 50, and then we started by building the tree recursively. If the samples belong to a single class, or the split reaches the max depth, we stop splitting, and label it as the predominant class. Otherwise, we find the best feature and threshold to split the data, divide the dataset into left and right subsets, and repeat this step to build the subtrees.

To be more specific, in order to find the best threshold, we used an exhaustive approach here, where it iterates all features and their unique values, and then calculates their corresponding Gain, and sets the one that maximizes Gain as threshold:

$$Gain = Gini_{parent} - (\frac{N_{left}}{N_{parent}} * Gini_{left} + \frac{N_{right}}{N_{parent}} * \ Gini_{right})$$

Then during the prediction phase, we just traverse the tree until a leaf node is reached, and the input data will be placed in the same class as this leaf node.

In order to investigate how the max depth affects the performance of the DT model, we also ran the DT model with a max depth of 5, 35, 75, and 100.

3. Multi-Layer Perceptron

Perceptron is the simplest structure of a neural network, and a Multi-Layer Perceptron model consists of many layers of perceptrons. It is used in both classification and regression. An MLP contains an input layer, which receives the input data, hidden layers, which perform most of the learning in the model by data transformation, and an output layer, which delivers the final classes or regression values.

Our default MLP model has an input layer of size 50, two hidden layers of size (50, 512) and (512, 512), and an output layer of size 10. This model uses ReLU (which avoid negative inputs) as its activation function, Stochastic Gradient Descent (SGD) with a momentum of 0.9 and learning rate of 0.01 as its optimization algorithm:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

and cross-entropy loss as loss function:

$$L = \sum_{i=1}^{N} \sum_{c=1}^{10} y_{i,c} \log \left(Softmax(y_{pred,i})_c \right)$$

Moreover, the hidden layer is normalized to produce a faster and more stable output. By default, the MLP will run 10 epochs, and the batch size is 32.

Then we also explored the factors that may affect MLP model performance by changing its hidden layer numbers to 1, 3, 5, and 7 (with hidden layer size of 512), and hidden layer sizes to 16, 128, and 1024 (with 2 hidden layers).

4. Convolution Neural Network

We implemented a convolutional neural network (CNN) model that takes an image of size 32 x 32 x 3 as input. The image is passed through a series of eight convolutional layers, each with a kernel size of 3 x 3. After each convolutional layer, we applied batch normalization and the ReLU activation function. In some layers, the output of the ReLU function is passed through the max-pooling layer with a kernel size of 2 x 2 and a stride of 2. We then flatten the output before passing it to a fully connected layer, which consists of three linear layers. Additionally, we apply the ReLU activation function and dropout layers after the first and second linear layers. The last linear layer outputs an array 1 x 10 which represents 10 object classes.

We implemented three variants of the CNN model by removing convolutional layers.

- The 4-layered CNN model (4-CNN) is created by keeping only the first convolutional layer and the fully connected layer from the original CNN model.

- The 6-layered CNN model (6-CNN) is implemented by keeping the first three convolutional layers and the fully connected layer from the original CNN model.

- The 9-layered CNN model (9-CNN) is derived by removing the last 2 convolutional layers from the original CNN model:
    - Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU
    - Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)

Moreover, we experiment with adding convolutional layers, resulting in two CNN models:

- The 12-layer CNN model (12-CNN) is developed by adding the convolutional layer "Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU" after the last convolutional layer of the original CNN model.
- The 13-layer CNN model (13-CNN) is derived by adding the convolutional layer "Conv(256, 256, 3, 1, 1) - BatchNorm(256) - ReLU" and "Conv(256, 256, 3, 1, 1) - BatchNorm(256) - ReLU - MaxPool(2, 2)" after the fourth layer of the original CNN model.

Furthermore, we experimented with different kernel sizes, such as 2 x 2, 5 x 5, and 7 x 7. We observed that using larger kernels could lead to errors when the input size is smaller than the kernel size. Therefore, we adjusted the padding to 2 and 3, respectively, when experimenting with kernel sizes of 5 x 5 and 7 x 7.

We finally trained the Convolutional Neural Network models for 30 epochs with a learning rate of 0.01 and the Cross Entropy loss function. Moreover, we used mini-batch gradient descent with a batch size of 64 and momentum set to 0.9.

Results & Discussion

In general, there are 4 important matrices we usually keep track in assessing the performance of a machine learning model: accuracy, precision, recall, and F1 score.

1. The **accuracy** measures the overall correctness of a model by calculating the ratio of correctly classified images to the total number of predictions. However, the accuracy matrix can be misleading sometimes, especially when the dataset is imbalanced.
2. **Precision**, on the other hand, focuses on how many of the predicted results are actually correct (both true positive and false positive). When the precision value is high, there are fewer positives.

3. The **recall** value implies how many of the actual positive results are correctly identified, and a high recall value is associated with fewer false negatives. In other words, it informs us about the sensitivity of the actual positive results.
4. Then the last important matrix that we are interested in is the **F1 score**, which gives us an insight of the trade-off between precision and recall. Generally, a good model should have a balanced F1 score (1), indicating that both false positives and false negatives are low.

With the clearly defined evaluating matrices, we will discuss the performance of each model in detail in the following sections.

1. Naive Bayes

Both our self-defined and Scikit's Naive Bayes models have the same performance, with an accuracy rate of 78.70%, precision rate of 79.23%, recall rate of 78.70%, and F1 score of 78.77% (Table 1). First of all, the same performance of the models indicates the correct implementation of our customized NB model. To be more specific, the high precision implies that the model is good at avoiding misclassification, while the high recall suggests that it is also doing well at minimizing the probability of not identifying images of a particular class. The F1 score demonstrates the trade-off between precision and recall, and it indicates that this model has a fair balance between these 2 matrices.

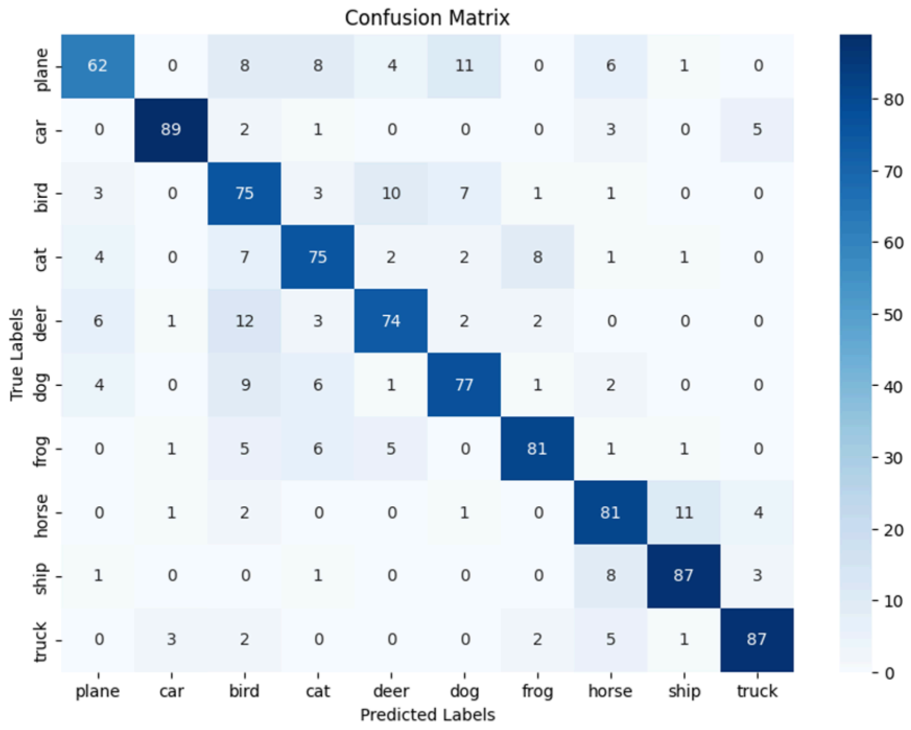
Table 1. Comparison of Performance of Different Naive Bayes Models

Model Name	Accuracy	Precision	Recall	F1 Score
Self-defined NB model	78.80%	79.30%	78.80%	78.87%
Scikit's NB model	78.80%	79.30%	78.80%	78.87%

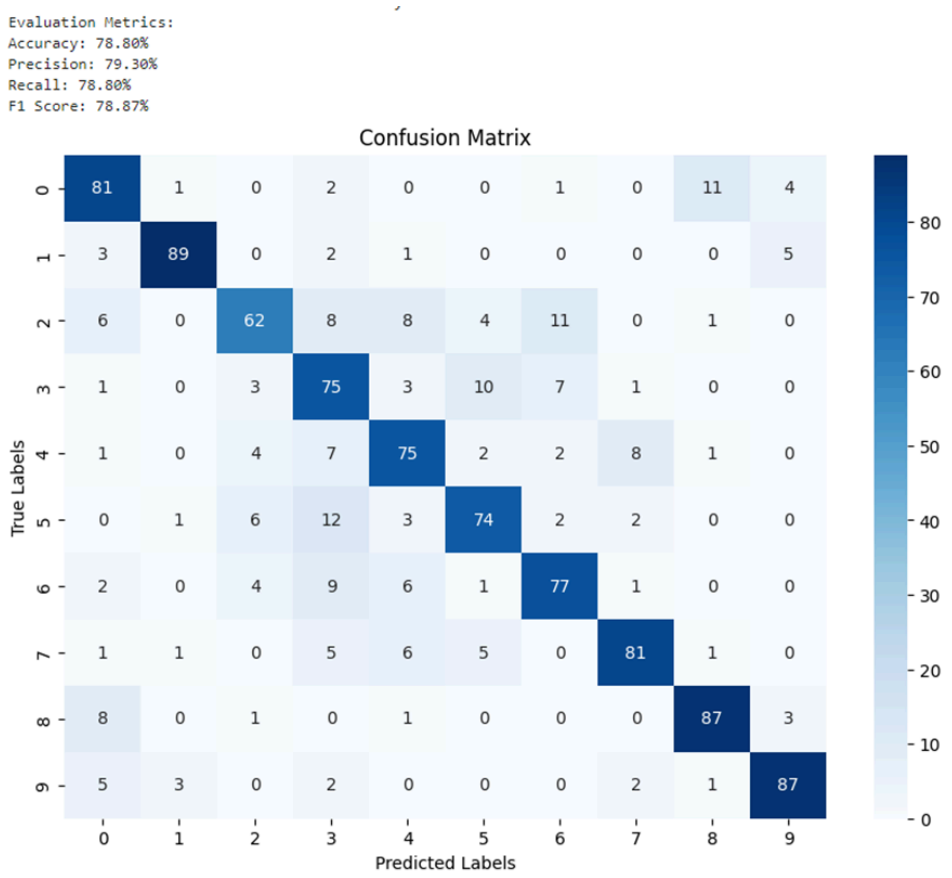
The confusion matrix indicates that the NB model can label most images correctly, with the highest number of identified images of 88 (class 1 and class 9), and lowest of 62 (class 2). However, this model misclassified 13 class0 samples as class 8, 12 class 2 samples as class 6, and 13 class 5 samples class 3.

Naive Bayes model is built on the assumption that the features are independent of each other, which might not be the case for CIFAR-10 images. For example, class0 Airplane and class8 Ship are both transportation tools, so they share many common features. In this case, if the model processes these features in an independent manner, it tends to mislabel a sample, since it fails to consider the correlation between features. Moreover, the Naïve Bayes model is a binary classifier, meaning that it might not be the most optimal choice to handle complex datasets such as RGB images, which contain an abundant number of non-linear patterns.

```
Evaluation Metrics:
Accuracy: 78.80%
Precision: 79.30%
Recall: 78.80%
F1 Score: 78.87%
```



a) self-defined NB model



```
Evaluation Metrics:
Accuracy: 78.80%
Precision: 79.30%
Recall: 78.80%
F1 Score: 78.87%
```

	0	1	2	3	4	5	6	7	8	9
0	81	1	0	2	0	0	1	0	11	4
1	3	89	0	2	1	0	0	0	0	5
2	6	0	62	8	8	4	11	0	1	0
3	1	0	3	75	3	10	7	1	0	0
4	1	0	4	7	75	2	2	8	1	0
5	0	1	6	12	3	74	2	2	0	0
6	2	0	4	9	6	1	77	1	0	0
7	1	1	0	5	6	5	0	81	1	0
8	8	0	1	0	1	0	0	0	87	3
9	5	3	0	2	0	0	0	2	1	87

b) Scikit's NB model

Figure 1. The comparison between self-defined NB and Scikit' NB models. a) The confusion matrix of the self-defined Naïve Bayes model. b) The confusion matrix of the Scikit's Naïve Bayes model.

2. Decision Tree

Comparison Between Default and Scikit' Decision Tree Models

Our self-implemented DT model has a similar performance compared to the Scikit's DT model, with an accuracy rate of (57.40% to 57.9%), precision value of (57.42% to 58.02%), recall value of (57.40% to 57.90%), and F1 measures of (57.28% to 57.92%) (Table 2 & Figure 2). Although the implementation of these two models are not exactly the same, they still share a lot of common parameter settings, and this is why they have a similar performance in image classification. Both models use Gini coefficient as criterion and limit the max depth as 50, and they try to use the best feature to split the tree.

Table 2. Comparison of Performance of Different Decision Tree Models

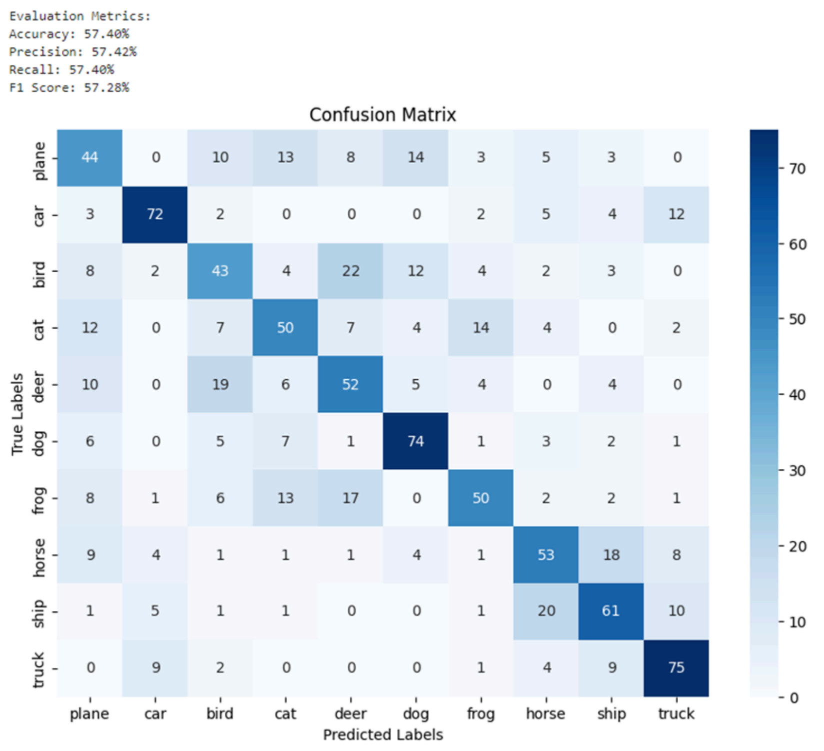
Model Name	Accuracy	Precision	Recall	F1 Score
Self-defined DT model (depth 5)	54.90%	56.26%	54.90%	54.15%
Self-defined DT model (depth 35)	57.70%	58.06%	57.70%	57.77%
***Self-defined DT model (depth 50)**	57.90%	58.02%	57.90%	57.92%
Scikit's DT model (depth 50)	57.40%	57.42%	57.40%	57.28%
Self-defined DT model (depth 75)	57.70%	58.06%	57.70%	57.77%
Self-defined DT model (depth 100)	57.70%	58.06%	57.70%	57.77%

*Default self-defined DT model (depth 50)

One main difference between the two is how to select the threshold. Threshold selection is a process of finding the optimal value of a feature to split the dataset. Our self-defined DT model iteratively checks each feature by using unique values, and for each potential threshold candidate, our model calculates the Gini gain, and eventually sets the one that maximizes Gini gain as the threshold. Meanwhile, Scikit's DT model used an optimizer to find an approximate optimal threshold. In this way, although it reduces computational complexity by avoiding exhaustive search, it may use suboptimal thresholds instead.

Although exhaustive search allows us to find the optimal threshold, it does not imply that this algorithm outperforms the Scikit's DT model in every way. For example, it may cause other problems such as overfitting, because it learned too much about this particular dataset. Overfitting is also a common problem of the DT algorithm, and it explains why both DT models have a less ideal outcome in image classification compared to other models, even a simpler model such as NB. In order to mitigate such potential concerns, the Scikit's DT model deployed other constraints such as *min_samples_leaf* to drop the node with few samples to avoid overfitting. Therefore, if we want to improve our DT model, this could be one aspect that we can work on.

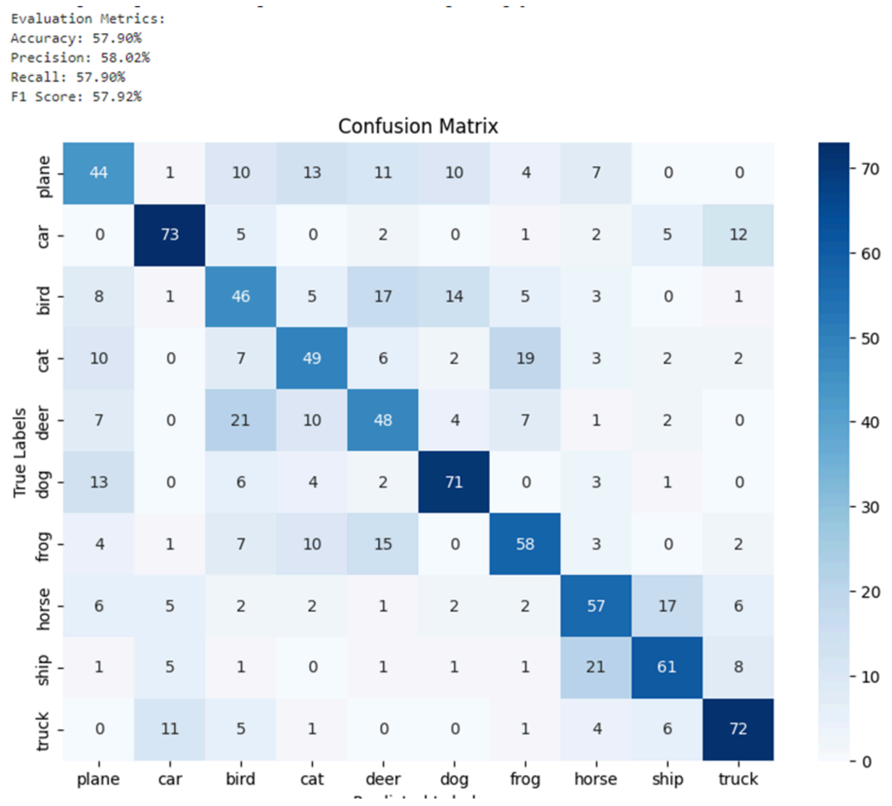
Aside from the potential concern of overfitting, there are other reasons why both DT models have a non-ideal performance in classifying CIFAR-10 images. CIFAR-10 is a complex dataset with diverse features, and it requires a good capacity of the model to catch and differentiate the spatial features. However, Decision Tree is a simple model, and it works better with structured data such as tables, where the features are treated independently. Therefore, DT is not the most suitable algorithm to handle high-dimension images.



```
Evaluation Metrics:
Accuracy: 57.40%
Precision: 57.42%
Recall: 57.40%
F1 Score: 57.28%
```

True \ Predicted	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	44	0	10	13	8	14	3	5	3	0
car	3	72	2	0	0	0	2	5	4	12
bird	8	2	43	4	22	12	4	2	3	0
cat	12	0	7	50	7	4	14	4	0	2
deer	10	0	19	6	52	5	4	0	4	0
dog	6	0	5	7	1	74	1	3	2	1
frog	8	1	6	13	17	0	50	2	2	1
horse	9	4	1	1	1	4	1	53	18	8
ship	1	5	1	1	0	0	1	20	61	10
truck	0	9	2	0	0	0	1	4	9	75

a) self-defined DT model



```
Evaluation Metrics:
Accuracy: 57.90%
Precision: 58.02%
Recall: 57.90%
F1 Score: 57.92%
```

True \ Predicted	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	44	1	10	13	11	10	4	7	0	0
car	0	73	5	0	2	0	1	2	5	12
bird	8	1	46	5	17	14	5	3	0	1
cat	10	0	7	49	6	2	19	3	2	2
deer	7	0	21	10	48	4	7	1	2	0
dog	13	0	6	4	2	71	0	3	1	0
frog	4	1	7	10	15	0	58	3	0	2
horse	6	5	2	2	1	2	2	57	17	6
ship	1	5	1	0	1	1	1	21	61	8
truck	0	11	5	1	0	0	1	4	6	72

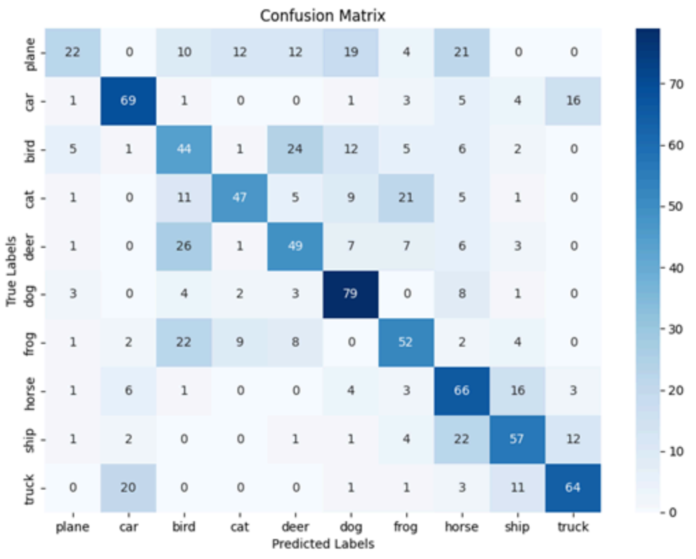
b) Scikit's DT model

Figure 2. The comparison between self-defined and Scikit' DT models. a) Evaluation of self-defined Decision Tree model. b) Evaluation of Scikit's Decision Tree model.

Comparison Between Decision Tree Models with Various Max Depths

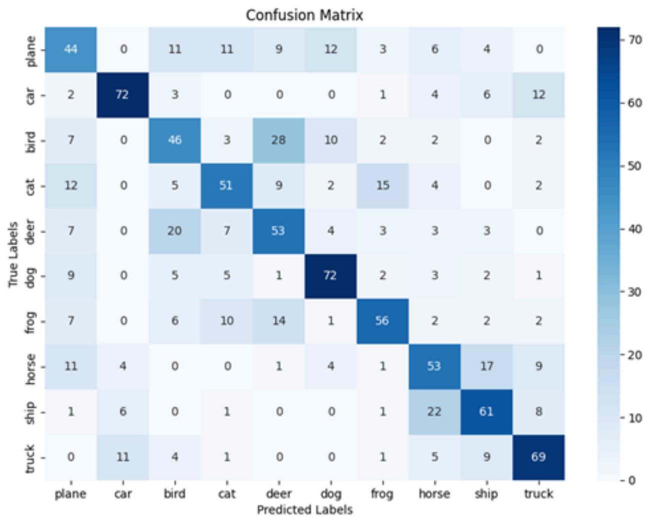
Moreover, we also explored how DT behaves with different depths (5,35, 75, and 100). Except a DT model with depth of 5, all other DT models with depth more than 35 demonstrated a similar effectiveness in image classification (Figure 3). Although a DT model with depth of 5 has a slightly inferior performance (~54% compared to ~57% in other models), the overall capacity still falls in the same range as others. This insignificant difference in the DT models with different depths indicates that the max depth is not the most important parameter in determining the performance of a DT model.

Another interesting point about the DT models we found is their ability to identify the objects in the image. According to the confusion matrix (Figure 2 & 3), both models succeeded and struggled in the same classes. For example, they both had a poor success rate of identifying birds, and they both confused between birds and deer. This indicates that these classes share overlapping features such as colors and shapes. These kinds of confusion may help us to better understand how object detection, or facial recognition, in image processing works. In other words, the key to obtain a model with better performance is to increase its ability to distinguish extreme edge cases, and this implies that simple models such as Naïve Bayes or Decision Tree are not well-suited for such complex tasks.



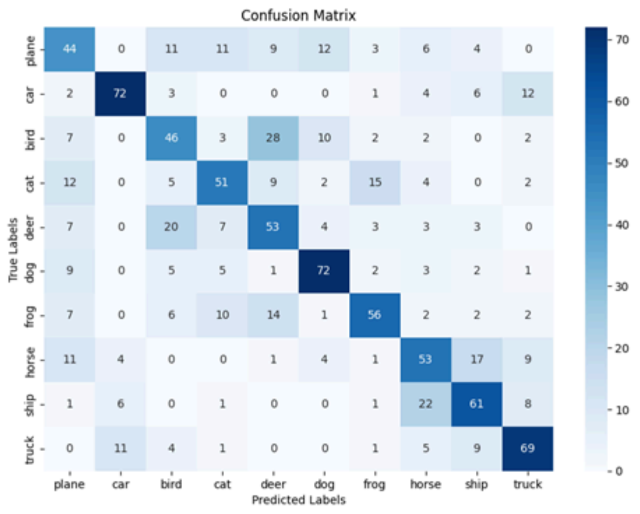
```
Evaluation Metrics
Accuracy: 54.90%
Precision: 56.26%
Recall: 54.90%
F1 Score: 54.15%
```

a) depth 5



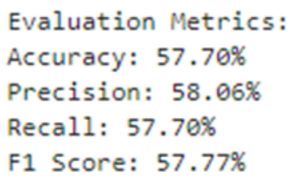
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	44	0	11	11	9	12	3	6	4	0
car	2	72	3	0	0	0	1	4	6	12
bird	7	0	46	3	28	10	2	2	0	2
cat	12	0	5	51	9	2	15	4	0	2
deer	7	0	20	7	53	4	3	3	3	0
dog	9	0	5	5	1	72	2	3	2	1
frog	7	0	6	10	14	1	56	2	2	2
horse	11	4	0	0	1	4	1	53	17	9
ship	1	6	0	1	0	0	1	22	61	8
truck	0	11	4	1	0	0	1	5	9	69

b)
```
Evaluation Metrics:
Accuracy: 57.70%
Precision: 58.06%
Recall: 57.70%
F1 Score: 57.77%
```
depth 35

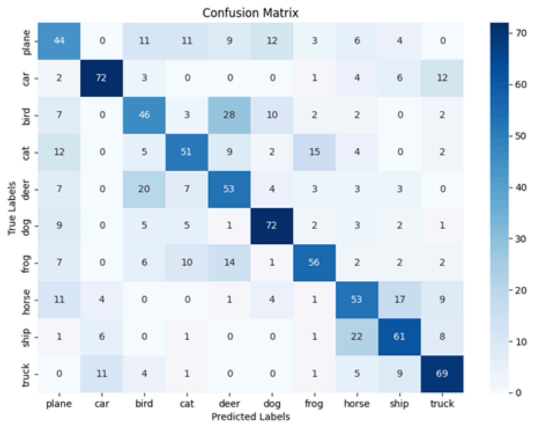


	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	44	0	11	11	9	12	3	6	4	0
car	2	72	3	0	0	0	1	4	6	12
bird	7	0	46	3	28	10	2	2	0	2
cat	12	0	5	51	9	2	15	4	0	2
deer	7	0	20	7	53	4	3	3	3	0
dog	9	0	5	5	1	72	2	3	2	1
frog	7	0	6	10	14	1	56	2	2	2
horse	11	4	0	0	1	4	1	53	17	9
ship	1	6	0	1	0	0	1	22	61	8
truck	0	11	4	1	0	0	1	5	9	69

c)
```
Evaluation Metrics:
Accuracy: 57.70%
Precision: 58.06%
Recall: 57.70%
F1 Score: 57.77%
```
depth 75

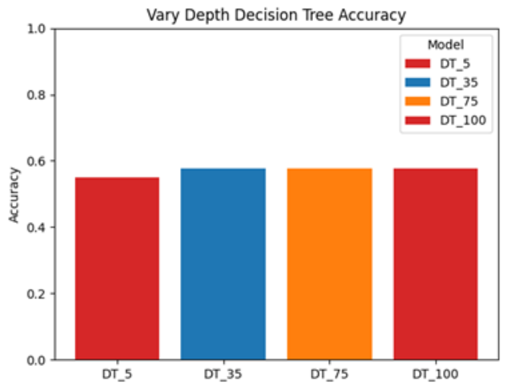
d) 

```
Evaluation Metrics:
Accuracy: 57.70%
Precision: 58.06%
Recall: 57.70%
F1 Score: 57.77%
```

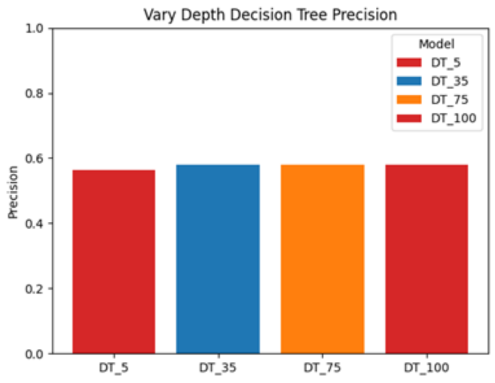


	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	44	0	11	11	9	12	3	6	4	0
car	2	72	3	0	0	0	1	4	6	12
bird	7	0	46	3	28	10	2	2	0	2
cat	12	0	5	51	9	2	15	4	0	2
deer	7	0	20	7	53	4	3	3	3	0
dog	9	0	5	5	1	72	2	3	2	1
frog	7	0	6	10	14	1	56	2	2	2
horse	11	4	0	0	1	4	1	53	17	9
ship	1	6	0	1	0	0	1	22	61	8
truck	0	11	4	1	0	0	1	5	9	69

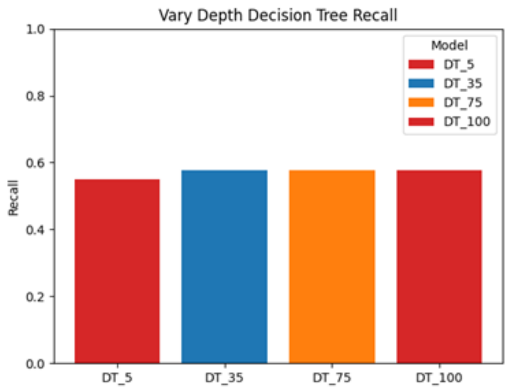
depth 100



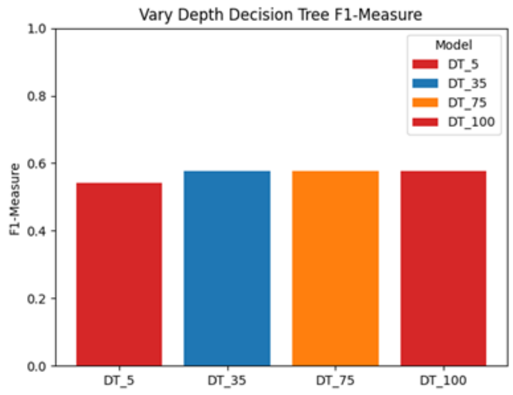
e)



f)



g)



h)

Figure 3. **The overall performance of DT models with different depths.** a) Evaluation of self-defined Decision Tree model of depth 5. b) Evaluation of self-defined Decision Tree model of depth 35. c) Evaluation of self-defined Decision Tree model of depth 75. d) Evaluation of self-defined Decision Tree model of depth 100. e) A comparison of accuracy rates between decision tree models with various depths. f) A comparison of precision values between decision tree models with various depths. g) A comparison of recall values between decision tree models with various depths. h) A comparison of F1-measures between decision tree models with various depths.

3. Multi-Layer Perceptron

All MLP models achieved a relatively high performance in this image classification task. Regardless of the hidden layer number and size, the MLP model successfully classifies at least 77.90% of images (Table 3). Moreover, they also all demonstrated a good score in precision, recall and F1 measures, indicating that the MLP model is generally better in complex image processing tasks, compared to simple models such as NB and DT.

Default MLP model

The default MLP that we used contains 2 hidden layers with the size of 512, and 10 epochs. According to Figure 4, the training loss gradually drops as epoch number increments, implying that the model is gradually learning. The increasing accuracy rate also proved that the model is gradually learning, and it almost reached its max value after epoch 7 (Figure 4).

Aside from using required settings, there are several parameters that need to be determined by us. The first is the learning rate of SGD optimizer. The overall goal of the SGD algorithm is to minimize the loss during training:

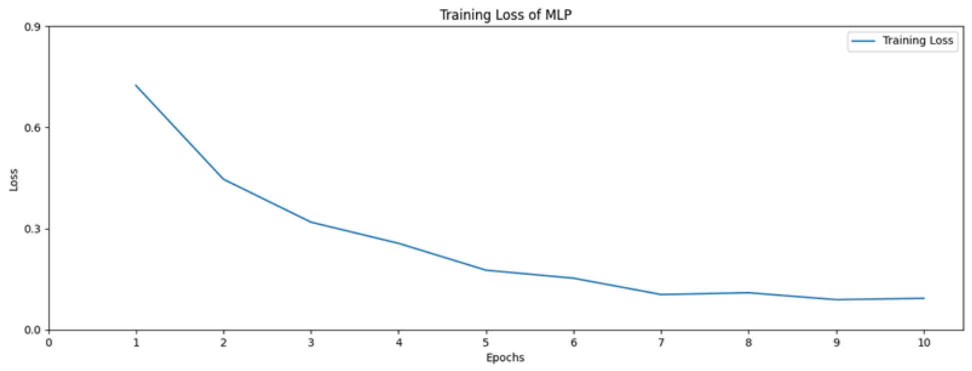
$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

The learning rate controls how fast the optimizer moves at each step. We used a default value (0.01) for our MLP model, and the overall performance did not change much when we gently increased/decreased the learning rate. However, regardless of our model, in general, a greater learning rate will result in quick computation, but may miss the minimum point, and a lower learning rate may have a relatively slower training, it has a higher chance to identify the minima. Moreover, the momentum in the SGD optimizer will accelerate training by considering past gradients. A momentum of 0.9 was used for all MLP models, indicating that 90% of the previous information is retained by the optimizer.

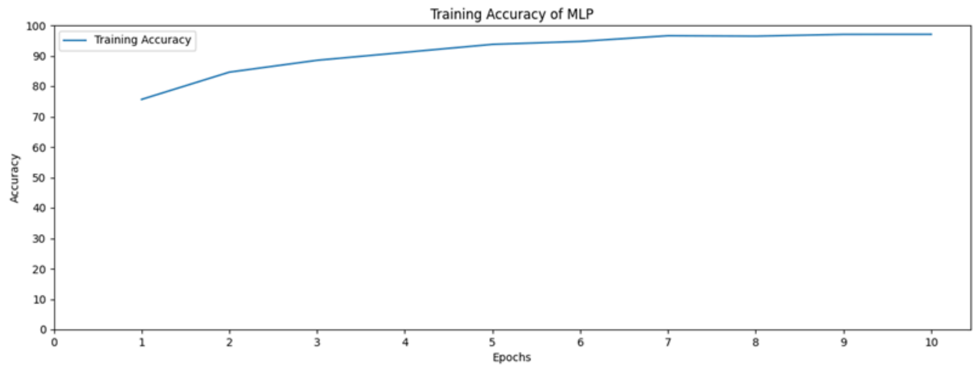
Aside from the learning rate, we also got a chance to determine the proper batch size used to train the model. In general, a larger batch size will accelerate the training process, but it may also lead to a risk of degraded performance. On the other hand, a smaller batch size indicates that the model will focus on a smaller set of images at a step, which slows down the training convergence, but has a higher chance of getting a better result. Since MLP models usually do not require a large amount of processing time, we prioritized the quality over quantity here, and chose a relatively small batch size (32).

The confusion matrix of the default MLP model also indicates neural network models' superior ability in handling sophisticated jobs compared to the simple models. Based on the confusion matrix, we can see that MLP works best for 'truck', 'dog', and 'ship' classes, since it

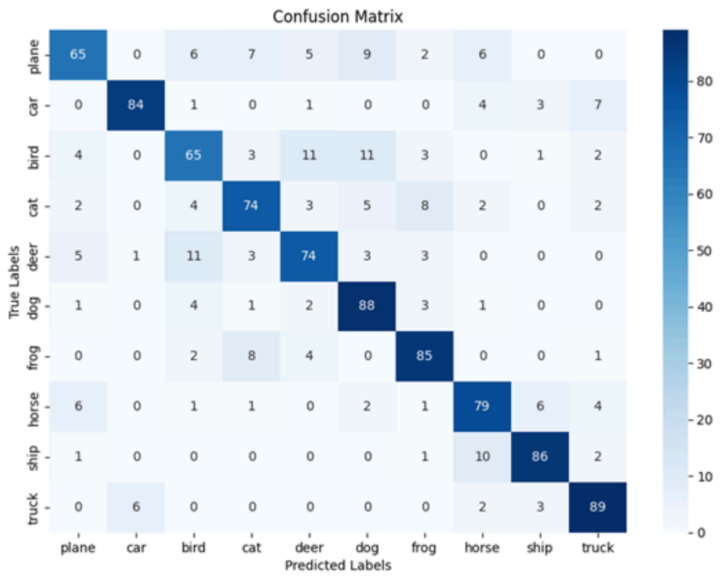
Default MLP model



a)



b)



c)

Figure 4. The performance of the default MLP model. a) The training loss of default MLP. b) Training accuracy of default MLP. c) Confusion matrix of default MLP.

successfully labelled 89, 88 and 86 images in the testing set. However, it is also weak in identifying objects such as 'plane' and 'birds', since it only classified 65 of them successfully (Figure 4c).

One possible explanation of the large number of misclassified images in certain classes is that MLP is not an ideal model in processing 2D images, since it still treats inputs as independent features. In order to extract features, especially facial features from images, the model needs to learn the spatial relationships between pixels. However, our MLP models took a flattened 1D vector as input, ignoring the spatial information. Moreover, unlike CNN that uses filters, MLP does not have filters to help it process local features. Therefore, MLP will miss a lot of fine edges, resulting in a bad feature detection in some cases, where details are crucial.

MLP Models with Various Numbers of Hidden Layers

After evaluating a default MLP model, we wanted to explore how different numbers of layers affect the performance of MLP. The default model has 2 hidden layers, and we also ran models with 1, 3, 5, 7 layers. All models showed a steadily reducing training loss and increasing training accuracy as the number of epochs increments (Figure 5), similar to the learning curve as seen in the default model.

Table 3. Comparison of Performance of Different Multi-Layer Perceptron Models

Model Name	Accuracy	Precision	Recall	F1 Score
MLP (hidden layer=1, layer size=512)	81.90%	82.09%	81.90%	81.88%
MLP (hidden layer=2, layer size=16)	77.90%	78.40%	77.90%	77.87%
MLP (hidden layer=2, layer size=128)	77.90%	78.40%	77.90%	77.87%
***MLP (hidden layer=2, layer size=512)**	78.90%	78.99%	78.90%	78.79%
MLP (hidden layer=2, layer size=1024)	77.90%	78.40%	77.90%	77.87%
MLP (hidden layer=3, layer size=512)	80.40%	80.88%	80.40%	80.36%
MLP (hidden layer=5, layer size=1024)	80.70%	81.01%	80.70%	80.63%
MLP (hidden layer=7, layer size=512)	77.90%	78.40%	77.90%	77.87%

*Default MLP (hidden layer=2, layer size=512)

Then, we compared 4 important measuring matrices of MLP models with varying numbers of layers. Surprisingly, the accuracy rate did not increase as the number of layers increased (Figure 6). On the contrary, the MLP model with only 1 hidden layer displayed the best performance, with an accuracy rate of 81.90%, and the model with most hidden layers had the worst performance, with an accuracy of 77.90% (Table 3). Although the overall trending for our MLP models when we increased the number of layers is a decreasing accuracy rate, the process is fluctuating. In other words, the model with 2 hidden layers showed a less optimal performance compared to models with 3 and 5 layers. Although this seems a bit unexpected, it still falls within an accepting range. The reason why we believed that is because of the random nature of our training process. All weights were initialized randomly at the beginning of training, so each training may lead to different results, and this particular combination of our model using 2 hidden layers may be more sensitive to this than other training. As long as the variance still falls in a consistent small range, we would believe that this is a normal acceptable result.

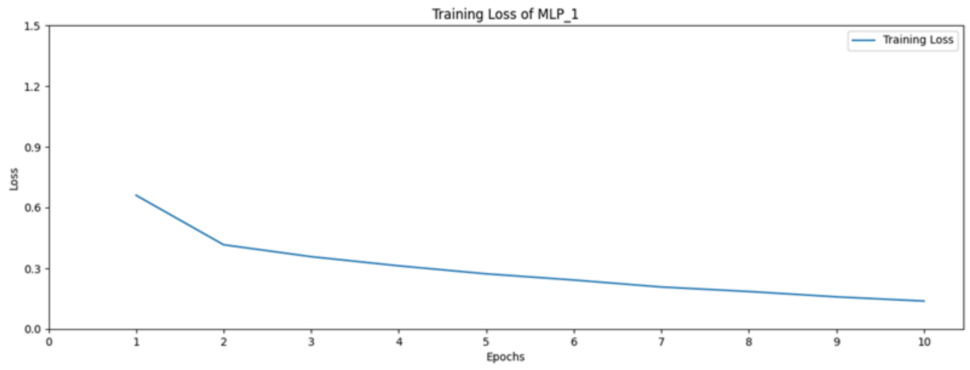
MLP Models with Various Size of Hidden Layers

Since the impact of varying the number of hidden layers in MLP, we also explored how the size of hidden layers affects model performance. According to Figure 7, all MLP models demonstrated an expected decreasing learning loss and increasing accuracy rate during training. However, the first model with a layer size of 16 still draws our attention, since its training loss started from around 130%, instead of 100%, which seemed odd at first sight. Although it looked a bit counterintuitive at first, based on our judgement, this is still acceptable. Since the weights are randomly initialized, with an abnormally large or small value, things like this will happen. Moreover, it is more important that the model shows an ability to reduce training loss as time goes by, and eventually reaches a low loss and stays stable.

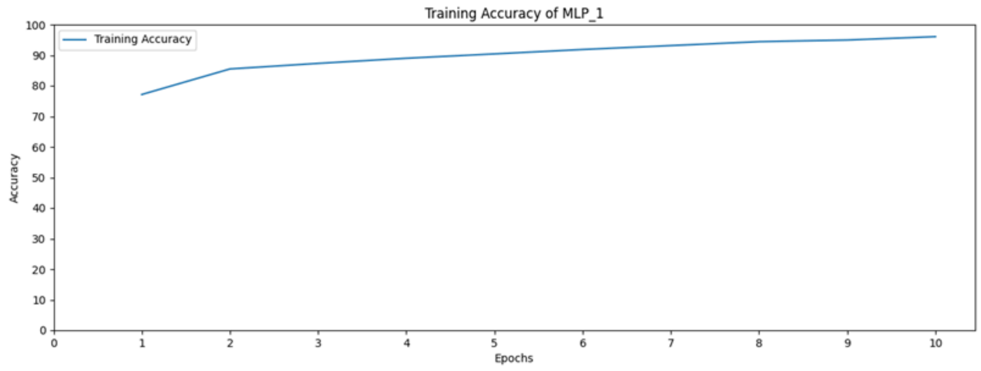
Apart from this training loss graph, other statistics of models are as what we expected. They demonstrated the same performance with an accuracy of 77.90%, precision of 78.40%, recall of 77.90%, and F1-score of 77.87% (Figure 8). However, the model with the best performance, even though the difference is minimal, is the default model with a layer size of 512.

To conclude, according to our test runs on various models, the best combination of a MLP model for this particular CIFAR-10 subset seems to be: 1 hidden layer with size of 512. However, all MLP models still demonstrated similar performance, which is more optimal than simple models such as NB and DT, implying that the choice of model itself might be more important than fine tuning the model sometimes.

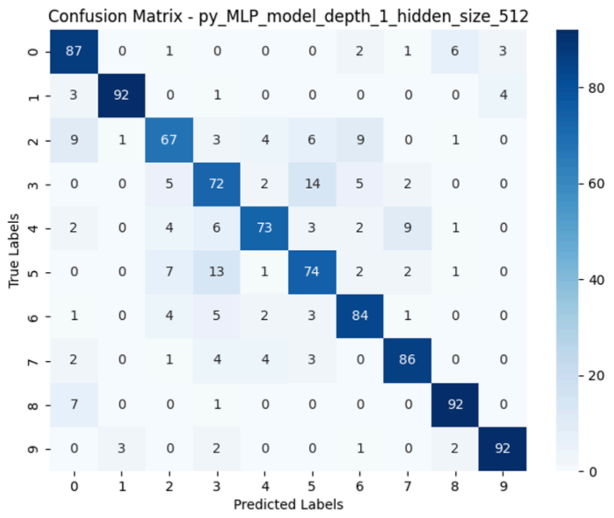
MLP with 1 hidden layer



a)



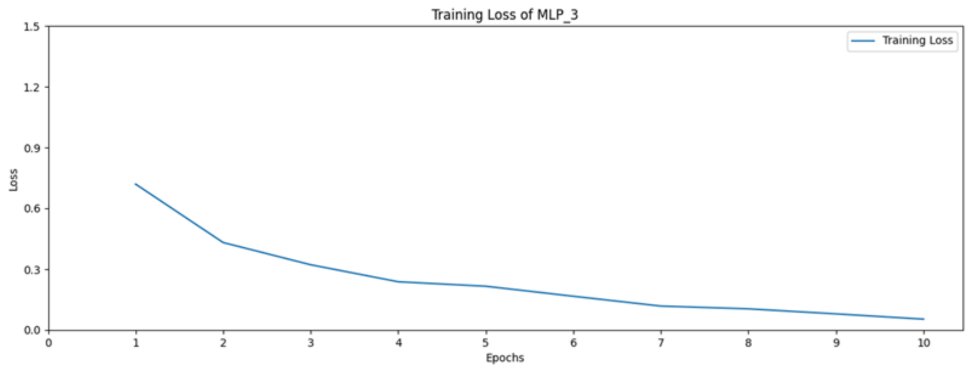
b)



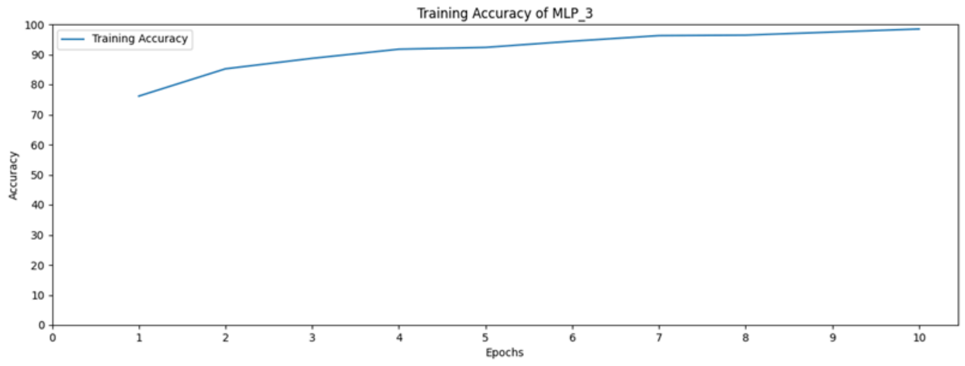
	0	1	2	3	4	5	6	7	8	9
0	87	0	1	0	0	0	2	1	6	3
1	3	92	0	1	0	0	0	0	0	4
2	9	1	67	3	4	6	9	0	1	0
3	0	0	5	72	2	14	5	2	0	0
4	2	0	4	6	73	3	2	9	1	0
5	0	0	7	13	1	74	2	2	1	0
6	1	0	4	5	2	3	84	1	0	0
7	2	0	1	4	4	3	0	86	0	0
8	7	0	0	1	0	0	0	0	92	0
9	0	3	0	2	0	0	1	0	2	92

c)

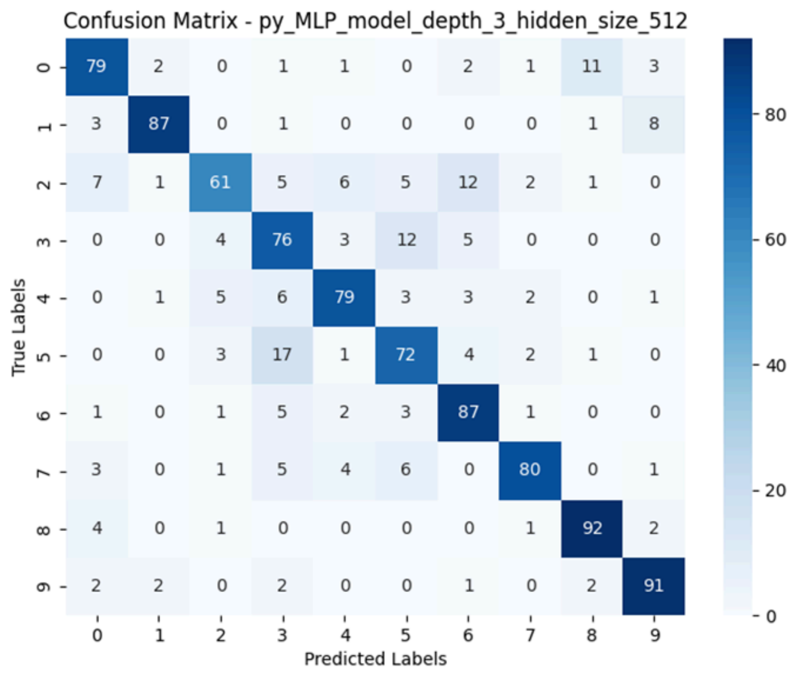
MLP with 3 hidden layers



d)



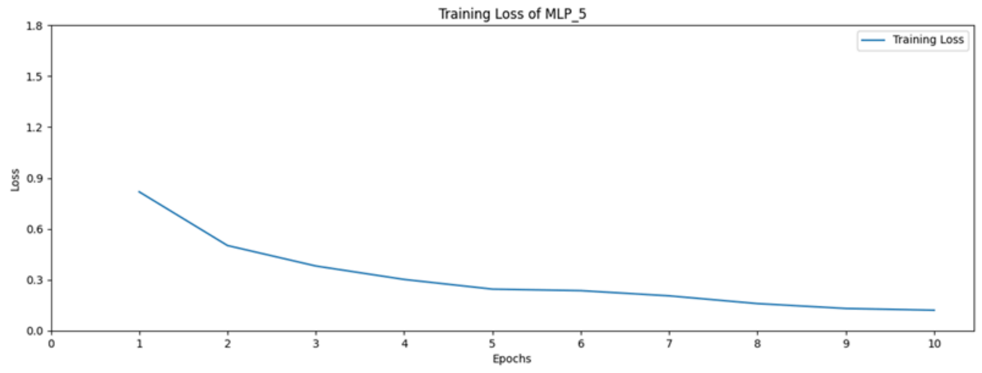
e)



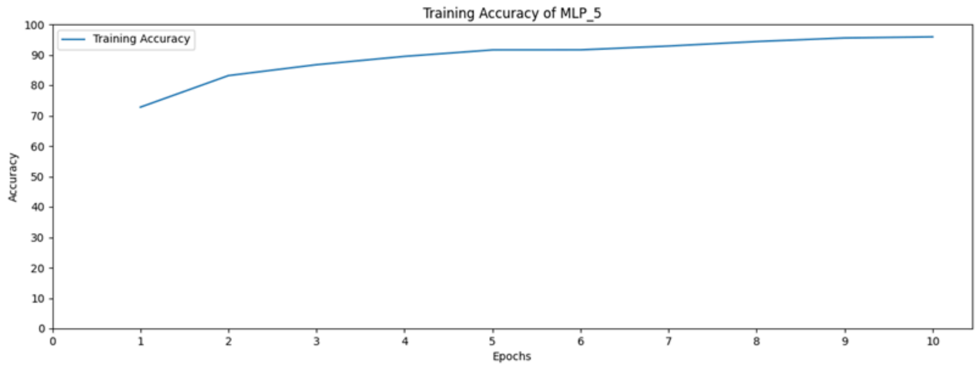
	0	1	2	3	4	5	6	7	8	9
0	79	2	0	1	1	0	2	1	11	3
1	3	87	0	1	0	0	0	0	1	8
2	7	1	61	5	6	5	12	2	1	0
3	0	0	4	76	3	12	5	0	0	0
4	0	1	5	6	79	3	3	2	0	1
5	0	0	3	17	1	72	4	2	1	0
6	1	0	1	5	2	3	87	1	0	0
7	3	0	1	5	4	6	0	80	0	1
8	4	0	1	0	0	0	0	1	92	2
9	2	2	0	2	0	0	1	0	2	91

f)

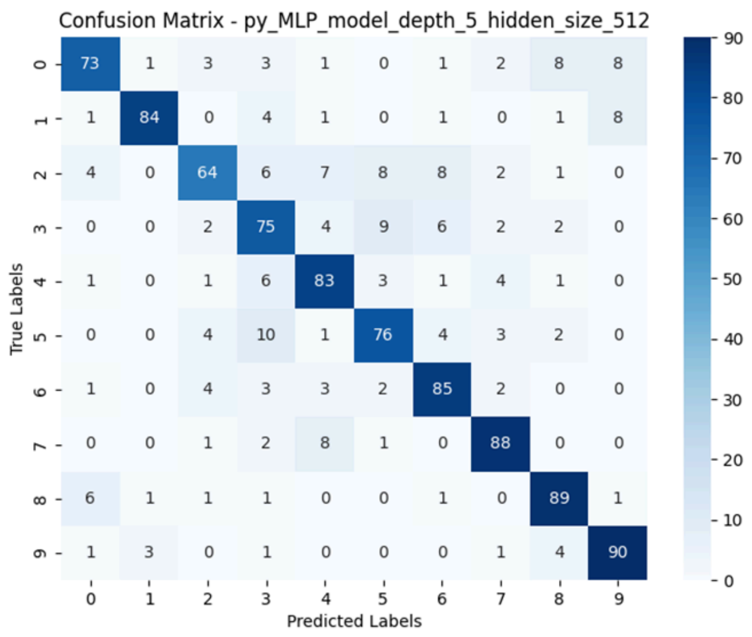
MLP with 5 hidden layers



g)



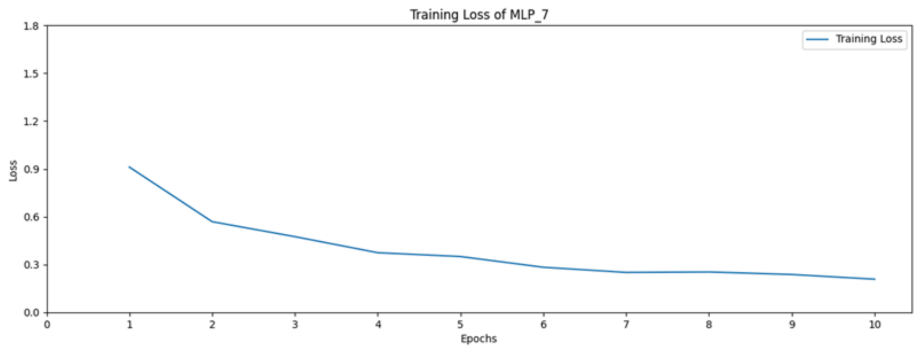
h)



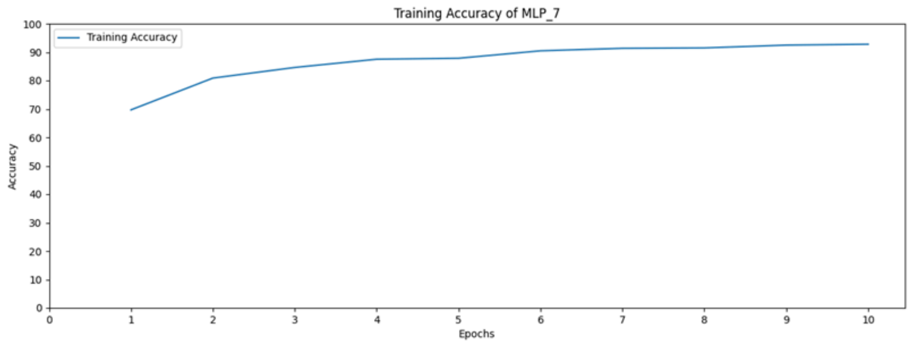
	0	1	2	3	4	5	6	7	8	9
0	73	1	3	3	1	0	1	2	8	8
1	1	84	0	4	1	0	1	0	1	8
2	4	0	64	6	7	8	8	2	1	0
3	0	0	2	75	4	9	6	2	2	0
4	1	0	1	6	83	3	1	4	1	0
5	0	0	4	10	1	76	4	3	2	0
6	1	0	4	3	3	2	85	2	0	0
7	0	0	1	2	8	1	0	88	0	0
8	6	1	1	1	0	0	1	0	89	1
9	1	3	0	1	0	0	0	1	4	90

i)

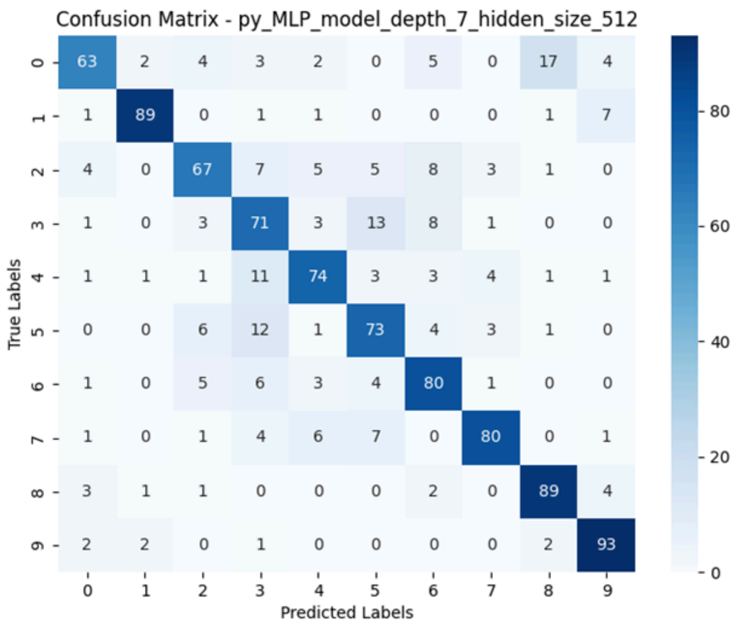
MLP with 7 hidden layers



j)

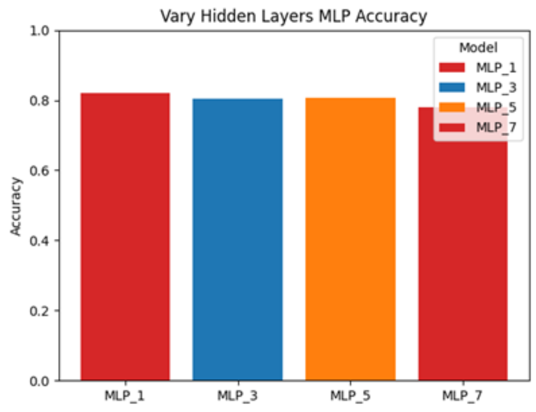


k)

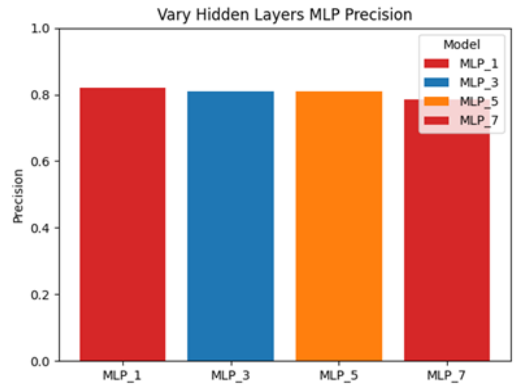


i)

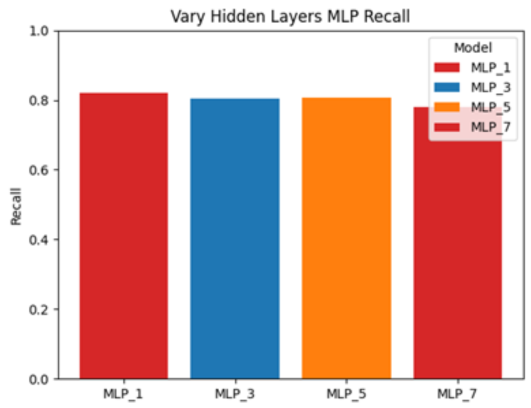
Figure 5. The performance of the MLP models with various numbers of hidden layers (1, 3, 5, 7). a-c) The training loss, accuracy rate, and confusion matrix of MLP model with 1 hidden layer. d-f) The training loss, accuracy rate, and confusion matrix of MLP model with 3 hidden layers. g-i) The training loss, accuracy rate, and confusion matrix of MLP model with 5 hidden layers. j-l) The training loss, accuracy rate, and confusion matrix of MLP model with 7 hidden layers.



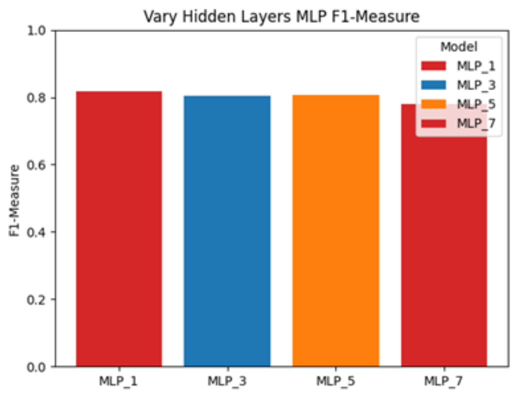
a)



b)



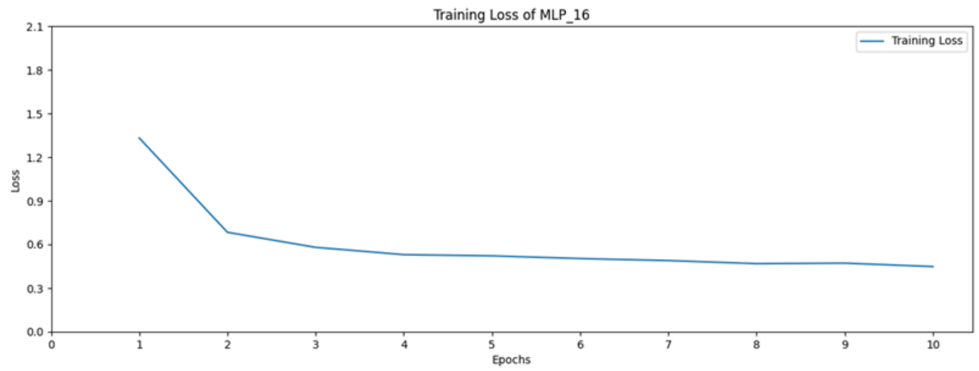
c)



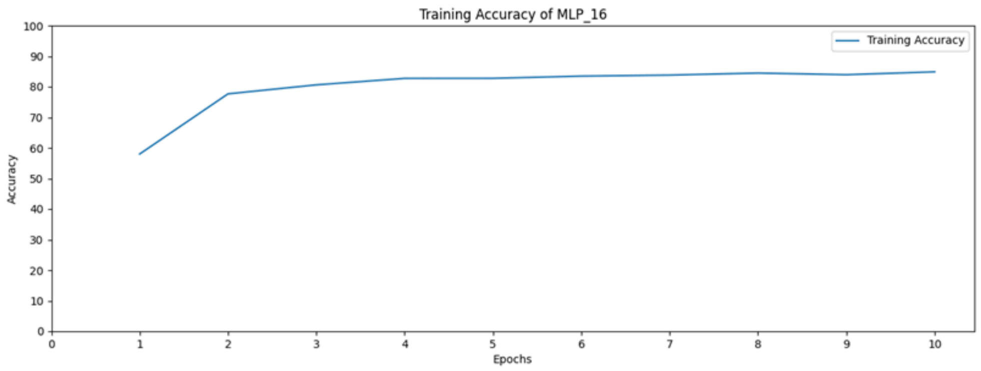
d)

Figure 6. The performance of MLP models with various number of layers (1,3,5,7). a) The accuracy rate of MLP models with different numbers of layers. b) The precision value of MLP models with different numbers of layers. c) The recall values of MLP models with different numbers of layers. d) F1-measure of MLP models with different numbers of layers.

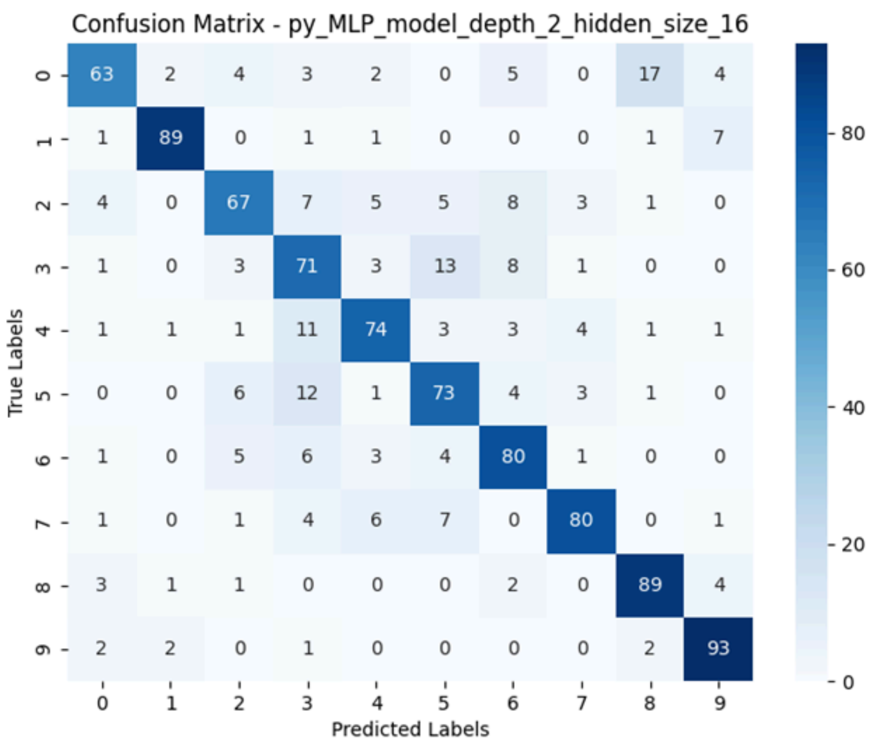
MLP with hidden layer size of 16



a)

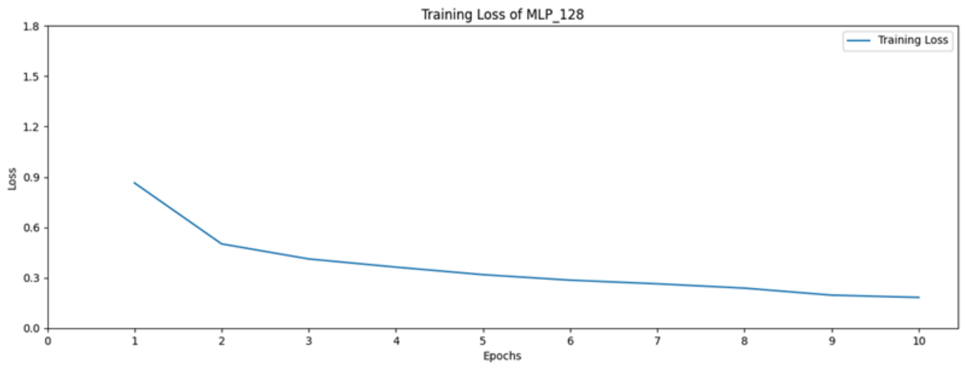


b)

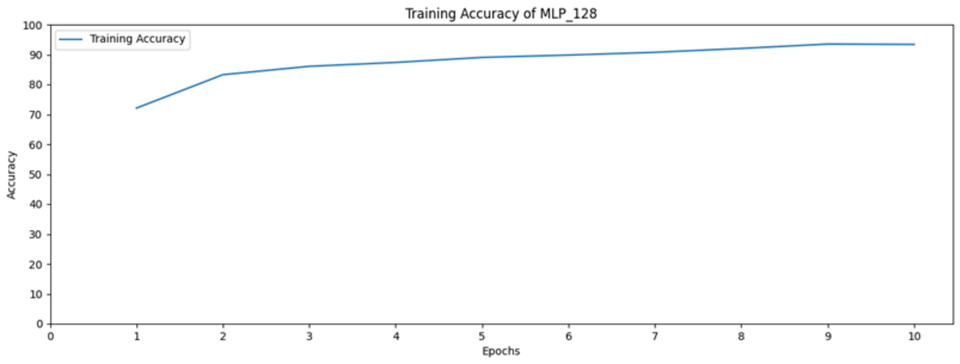


c)

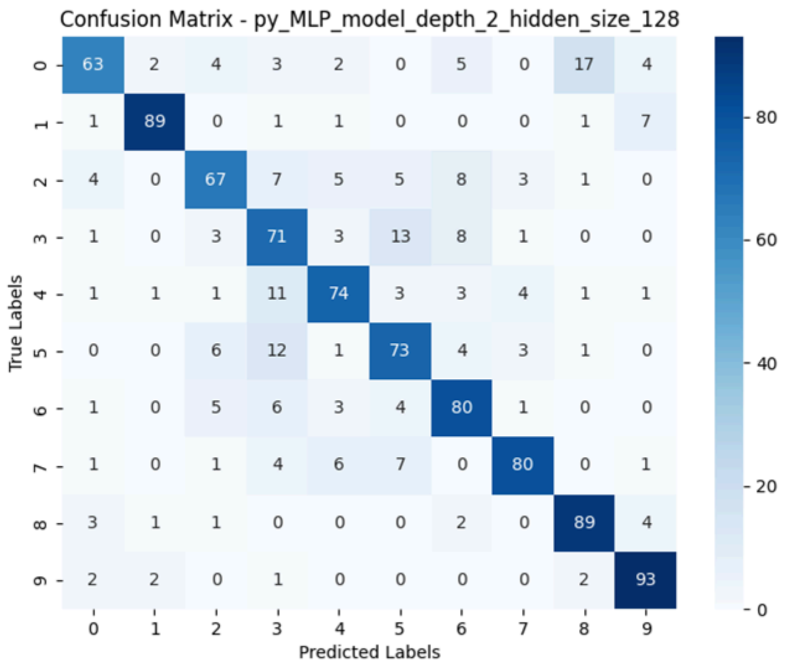
MLP with hidden layer size of 128



d)

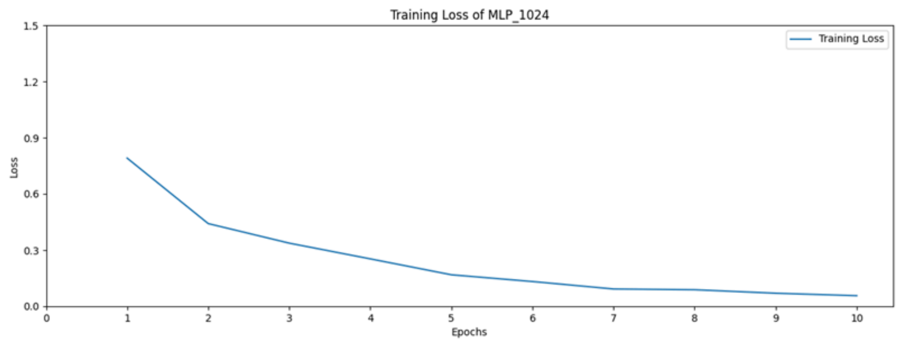


e)

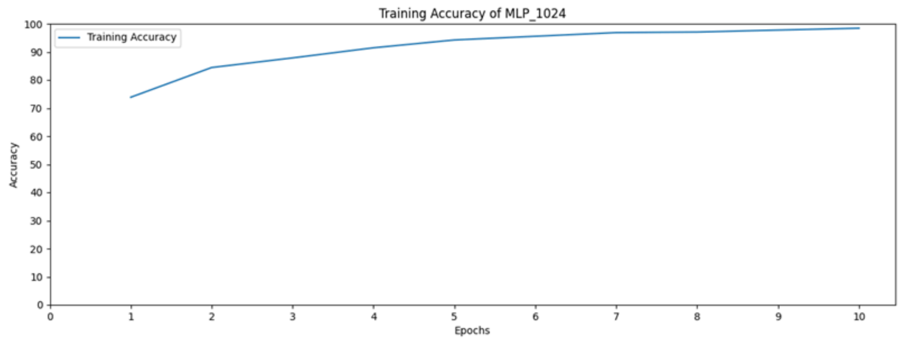


f)

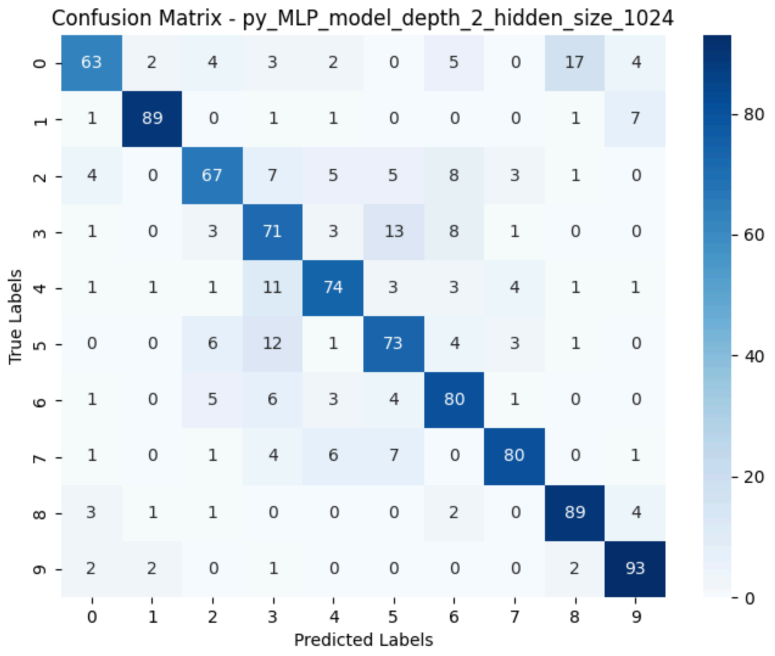
MLP with hidden layer size of 1024



g)



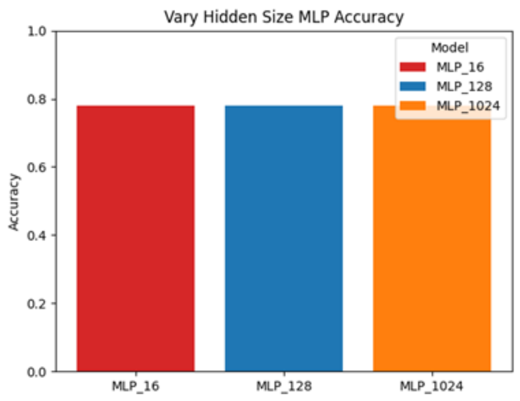
h)



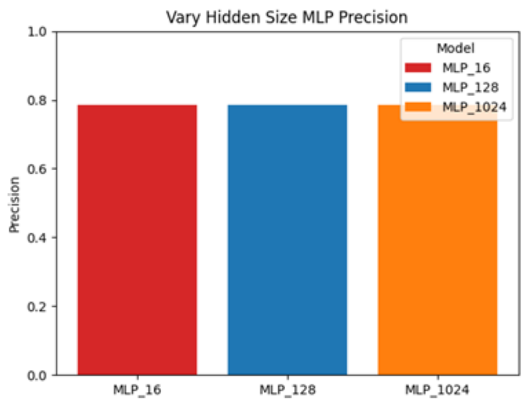
	0	1	2	3	4	5	6	7	8	9
0	63	2	4	3	2	0	5	0	17	4
1	1	89	0	1	1	0	0	0	1	7
2	4	0	67	7	5	5	8	3	1	0
3	1	0	3	71	3	13	8	1	0	0
4	1	1	1	11	74	3	3	4	1	1
5	0	0	6	12	1	73	4	3	1	0
6	1	0	5	6	3	4	80	1	0	0
7	1	0	1	4	6	7	0	80	0	1
8	3	1	1	0	0	0	2	0	89	4
9	2	2	0	1	0	0	0	0	2	93

i)

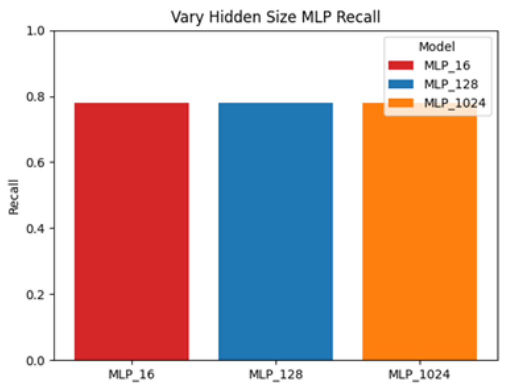
Figure 7. The performance of the MLP models with various sizes of hidden layers (16, 128, 1024). a-c) The training loss, accuracy rate, and confusion matrix of MLP model with hidden layer size of 16. d-f) The training loss, accuracy rate, and confusion matrix of MLP model with hidden layer size of 128. g-i) The training loss, accuracy rate, and confusion matrix of MLP model with hidden layer size of 1024.



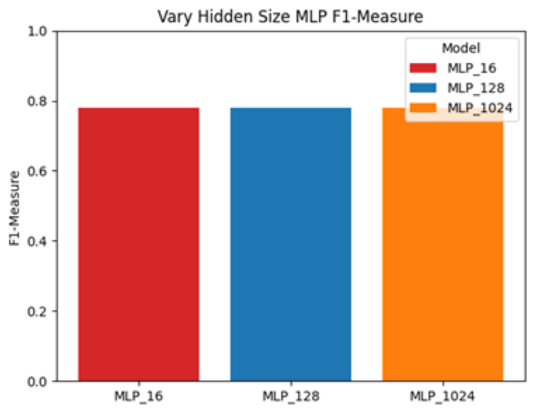
a)



b)



c)



d)

Figure 8. The performance of MLP models with various sizes of hidden layers (16, 128, 1024). a) The accuracy rate of MLP models with different hidden layer sizes. b) The precision value of MLP models with different hidden layer sizes. c) The recall values of MLP models with different hidden layer sizes. d) F1-measure of MLP models with different hidden layer sizes.

4. Convolutional Neural Network

All CNN models perform at an average level when trained on a subset of the CIFAR-10 dataset. Furthermore, although they achieve training accuracies above 90%, they struggle to generalize to new samples, implying that these models are overfitted.

The default model achieves an accuracy of 64.00%, precision of 63.55%, recall of 64.00%, and an F1 measure of 63.50%.

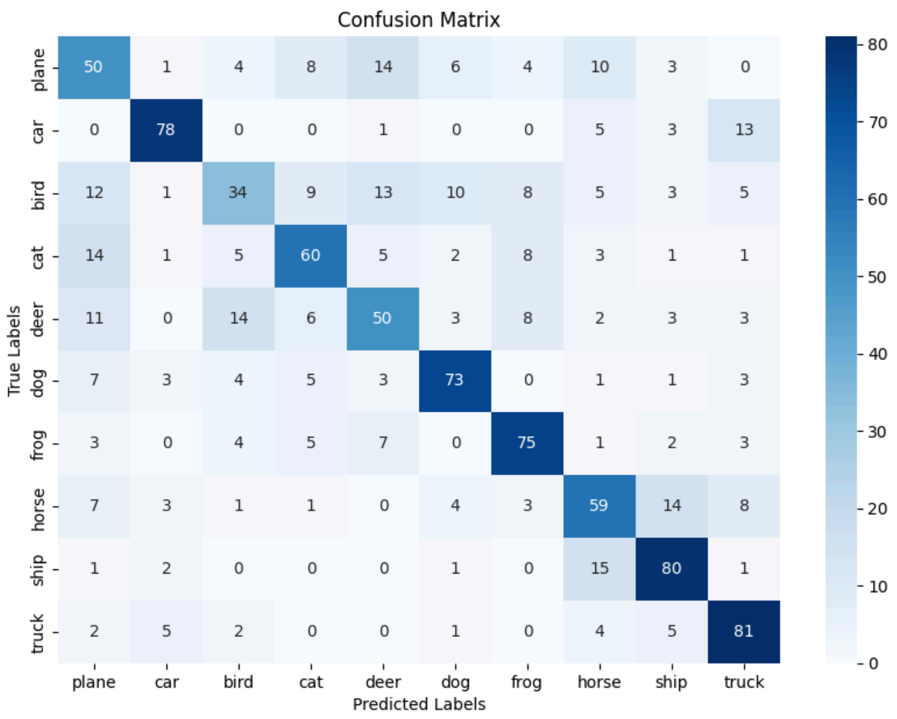


Figure 9. The confusion matrix of default CNN trained on 5000 CIFAR-10 images.

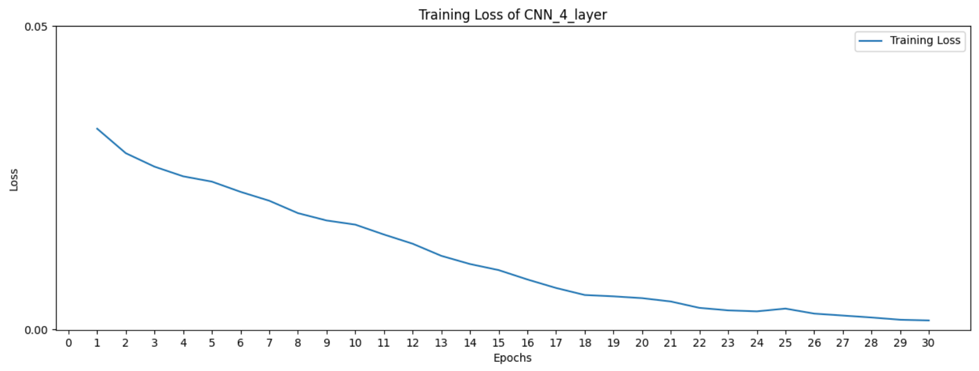
The confusion matrix shows that the CNN model performs best at classifying ship and truck images. However, the model performs poorly at labeling bird images, correctly predicting only 34 out of 100 bird images. Moreover, the model tends to confuse birds with deer, planes, and dogs, and ships with horses, and trucks with cars. This could be because these objects share similar visual features. For example, both birds and planes have wings, while birds and deer both have eyes on the sides of their heads. Ships and horses may also be confused due to their similar size. These shared features may be factors that the model uses to differentiate between classes. Moreover, the objects might share similar natural backgrounds, which could also be details the model pays attention to when classifying objects. For instance, birds and planes often appear in the sky, while trucks and cars are typically found in traffic settings. This also explains why ship and truck images are classified well, as these objects do not share as many distinctive features, such as specific shapes or background contexts (e.g., the sea for ships).

Variants of Convolutional Neural Network Model

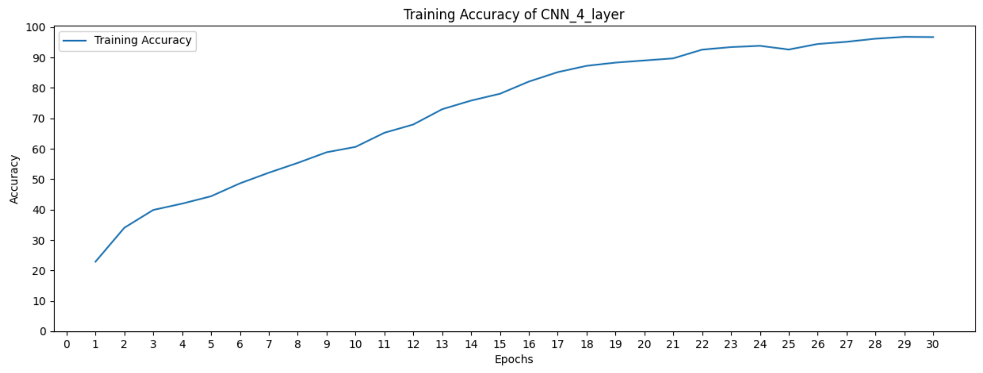
a. Different-layered CNN models

We experimented with the default CNN model with various numbers of layers (i.e 4, 6, 9, 12, 13). Their performance of those models are reported in Figure 9, Figure 10, and Table 4.

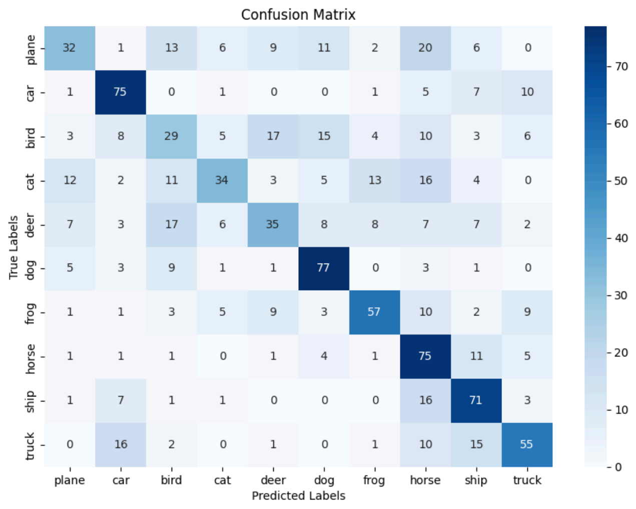
4-layered CNN model (4-CNN)



a)



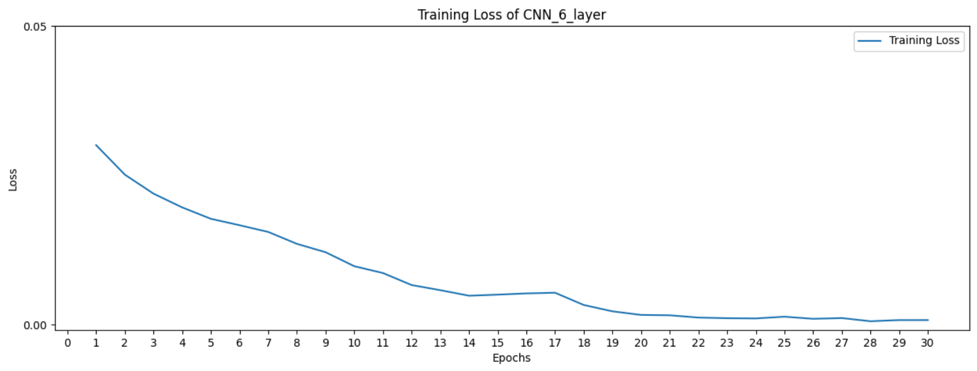
b)



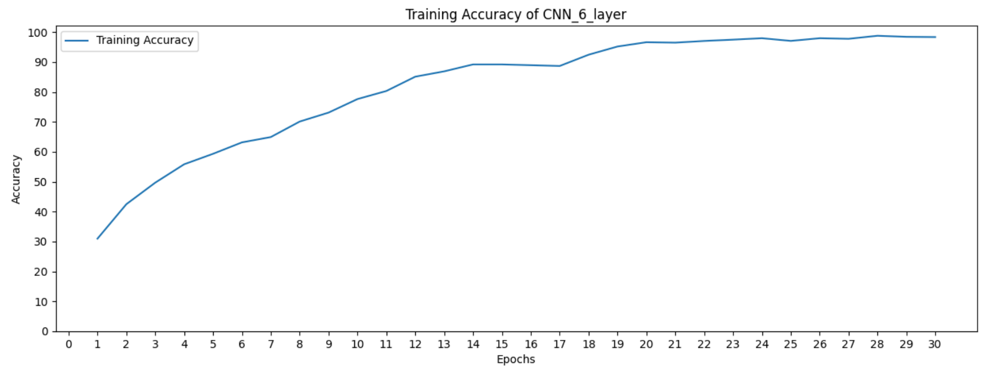
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	32	1	13	6	9	11	2	20	6	0
car	1	75	0	1	0	0	1	5	7	10
bird	3	8	29	5	17	15	4	10	3	6
cat	12	2	11	34	3	5	13	16	4	0
deer	7	3	17	6	35	8	8	7	7	2
dog	5	3	9	1	1	77	0	3	1	0
frog	1	1	3	5	9	3	57	10	2	9
horse	1	1	1	0	1	4	1	75	11	5
ship	1	7	1	1	0	0	0	16	71	3
truck	0	16	2	0	1	0	1	10	15	55

c)

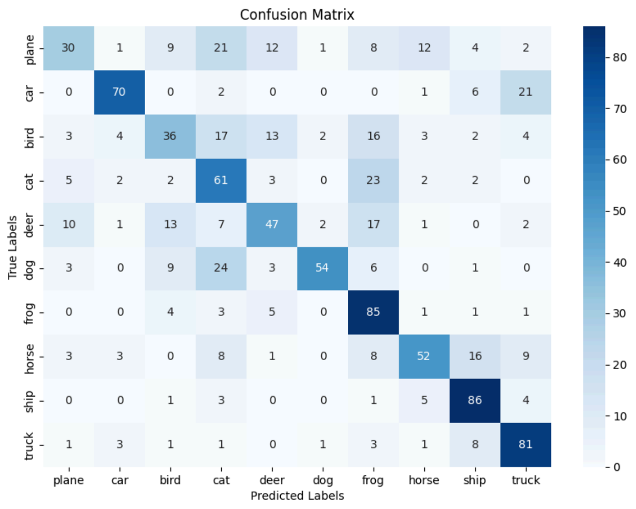
6-layered CNN model (6-CNN)



d)



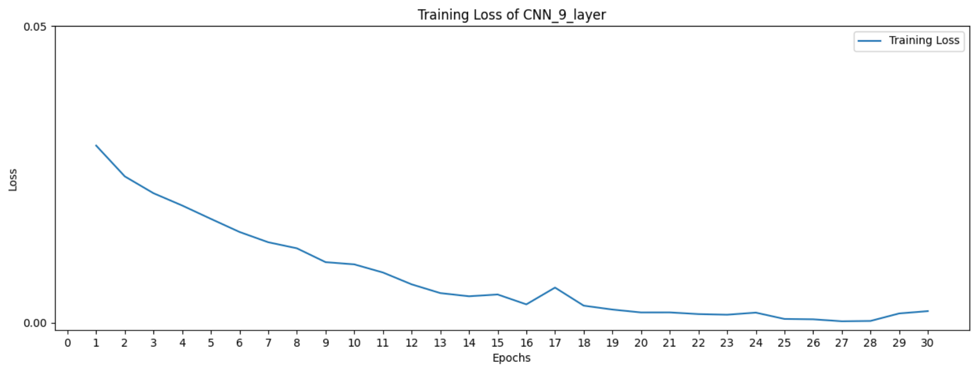
e)



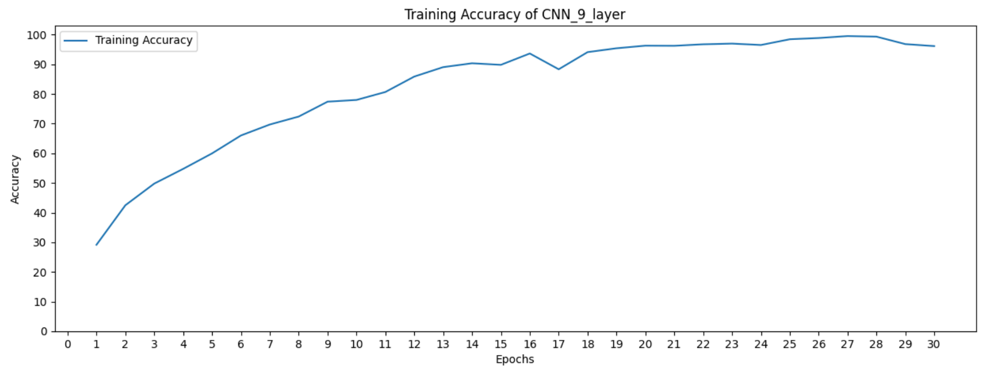
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	30	1	9	21	12	1	8	12	4	2
car	0	70	0	2	0	0	0	1	6	21
bird	3	4	36	17	13	2	16	3	2	4
cat	5	2	2	61	3	0	23	2	2	0
deer	10	1	13	7	47	2	17	1	0	2
dog	3	0	9	24	3	54	6	0	1	0
frog	0	0	4	3	5	0	85	1	1	1
horse	3	3	0	8	1	0	8	52	16	9
ship	0	0	1	3	0	0	1	5	86	4
truck	1	3	1	1	0	1	3	1	8	81

f)

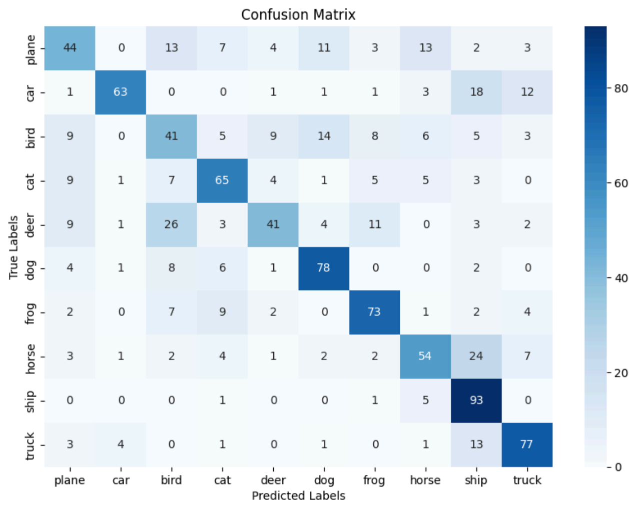
9-layered CNN model (9-CNN)



g)



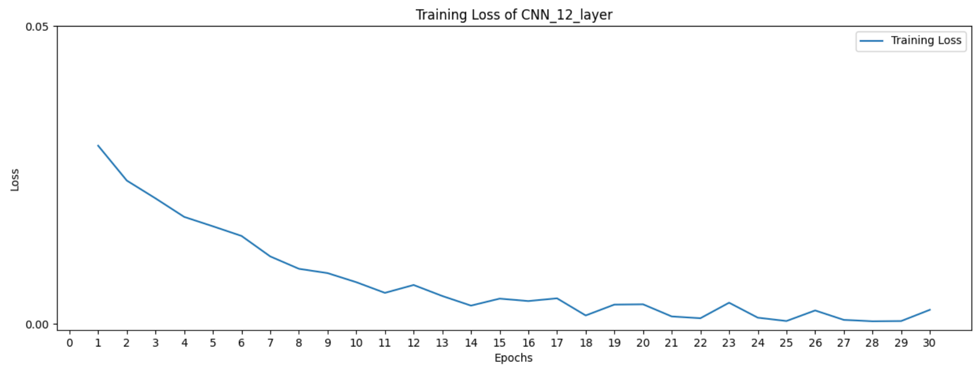
h)



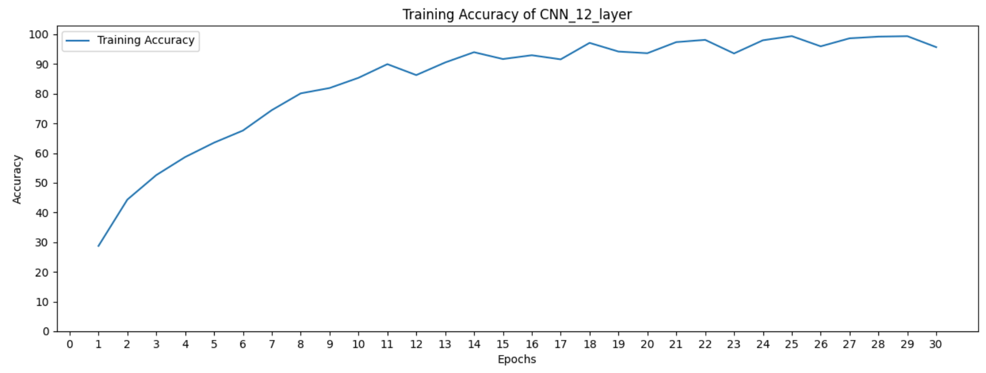
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	44	0	13	7	4	11	3	13	2	3
car	1	63	0	0	1	1	1	3	18	12
bird	9	0	41	5	9	14	8	6	5	3
cat	9	1	7	65	4	1	5	5	3	0
deer	9	1	26	3	41	4	11	0	3	2
dog	4	1	8	6	1	78	0	0	2	0
frog	2	0	7	9	2	0	73	1	2	4
horse	3	1	2	4	1	2	2	54	24	7
ship	0	0	0	1	0	0	1	5	93	0
truck	3	4	0	1	0	1	0	1	13	77

i)

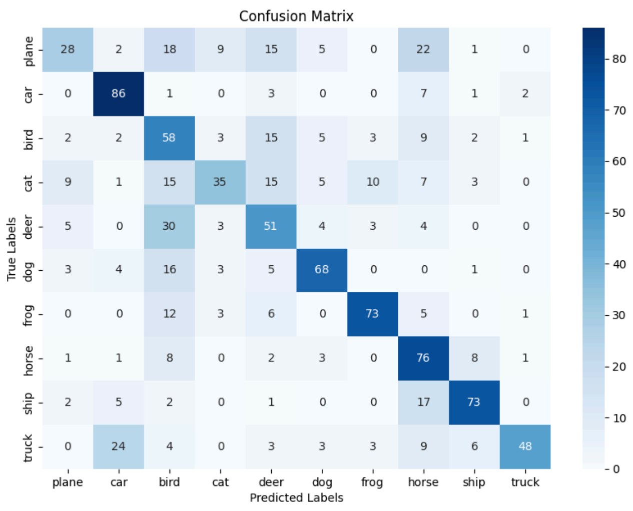
12-layered CNN model (12-CNN)



j)



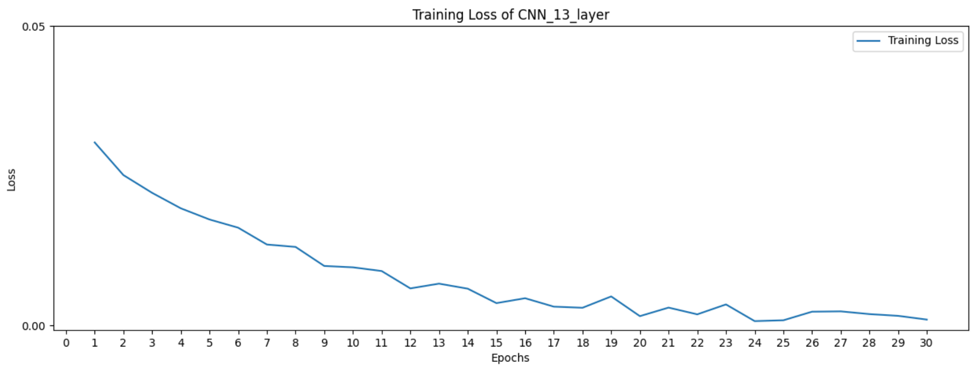
k)



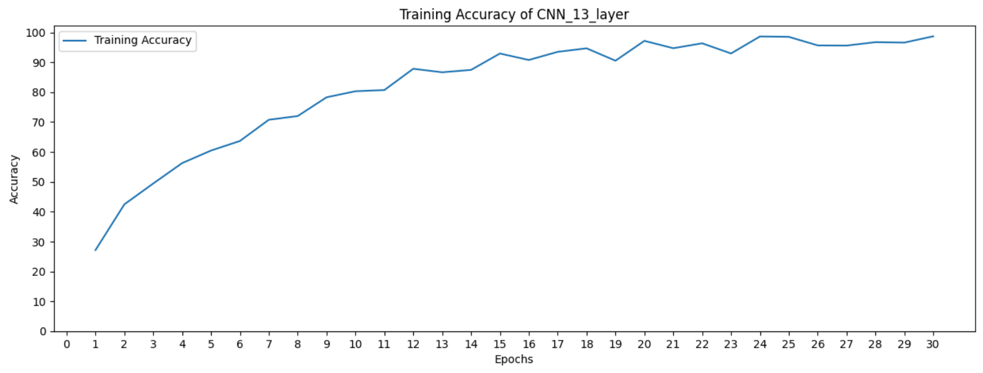
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	28	2	18	9	15	5	0	22	1	0
car	0	86	1	0	3	0	0	7	1	2
bird	2	2	58	3	15	5	3	9	2	1
cat	9	1	15	35	15	5	10	7	3	0
deer	5	0	30	3	51	4	3	4	0	0
dog	3	4	16	3	5	68	0	0	1	0
frog	0	0	12	3	6	0	73	5	0	1
horse	1	1	8	0	2	3	0	76	8	1
ship	2	5	2	0	1	0	0	17	73	0
truck	0	24	4	0	3	3	3	9	6	48

i)

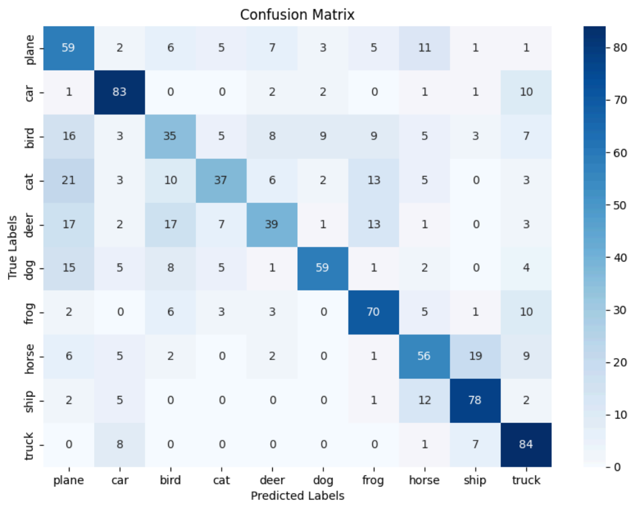
13-layered CNN model (13-CNN)



m)



n)



	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	59	2	6	5	7	3	5	11	1	1
car	1	83	0	0	2	2	0	1	1	10
bird	16	3	35	5	8	9	9	5	3	7
cat	21	3	10	37	6	2	13	5	0	3
deer	17	2	17	7	39	1	13	1	0	3
dog	15	5	8	5	1	59	1	2	0	4
frog	2	0	6	3	3	0	70	5	1	10
horse	6	5	2	0	2	0	1	56	19	9
ship	2	5	0	0	0	0	1	12	78	2
truck	0	8	0	0	0	0	0	1	7	84

o)

Figure 10. The performance of CNN models with various numbers of layers (4, 6, 9, 12, 13). a-c) The training loss, accuracy rate, and confusion matrix of the 4-layered CNN model. d-f) The training loss, accuracy rate, and confusion matrix of the 6-layered CNN model. g-i) The training loss, accuracy rate, and confusion matrix of the 9-layered CNN model. j-l) The training loss, accuracy rate, and confusion matrix of the 12-layered CNN model. m-o) The training loss, accuracy rate, and confusion matrix of the 13-layered CNN model.

Table 4: Evaluation metrics for CNN models with different layers

Model Name	Accuracy	Precision	Recall	F1 Score
Default CNN model	64.00%	63.55%	64.00%	63.50%
4-layered CNN model	54.00%	54.10%	54.00%	52.77%
6-layered CNN model	60.20%	62.45%	60.20%	59.44%
9-layered CNN model	62.90%	63.88%	62.90%	62.35%
12-layered CNN model	59.60%	63.52%	59.60%	59.30%
13-layered CNN model	60.00%	60.29%	60.00%	59.18%

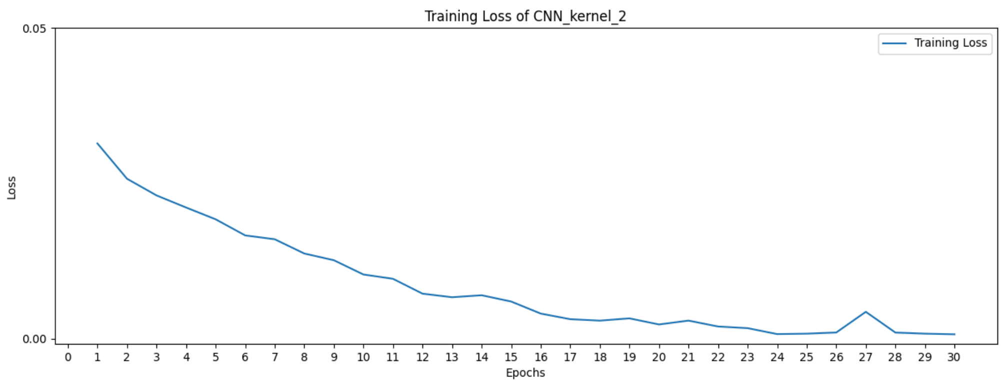
According to Figure 10, CNN models with more layers tend to converge faster, achieving higher training accuracy in fewer epochs. Suppose our target training accuracy is set to 90%. The 4-layer CNN takes 22 epochs to achieve 90% accuracy, the 6-layer CNN takes 18 epochs, the 9-layer CNN takes 17 epochs, while the 12-layer and 13-layer CNNs take 14 epochs.

Furthermore, the confusion matrices of the CNN models with different layers show that models with fewer convolutional layers are more likely to misclassify objects, as they do not learn enough details to differentiate between them. On the other hand, models with too many layers may focus too much on specific details of the objects, which can lead to overfitting.

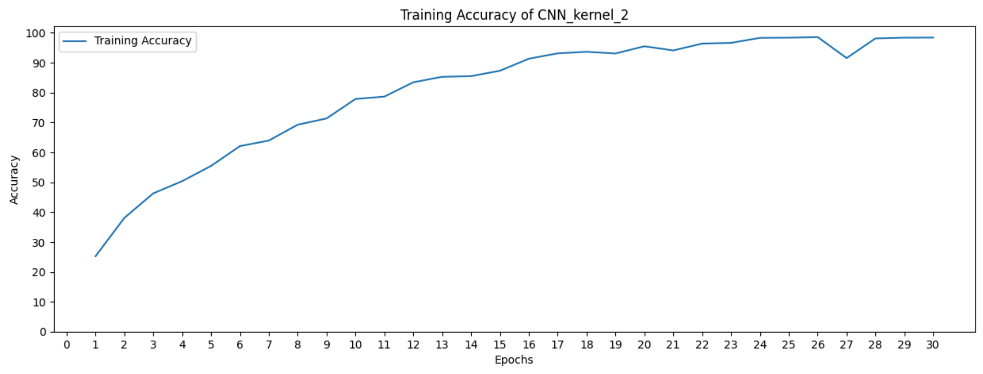
Moreover, Table 4 shows that the number of layers can affect model performance. The model with the fewest layers has the worst performance. However, adding more layers does not always improve the model's performance. For example, the CNN models with 12 and 13 layers perform worse than the model with 9 layers. Additionally, adding more layers can be costly and time-consuming to train.

To investigate how kernel size affects a CNN model, we experimented with the default CNN model using three different kernel sizes: 2x2, 5x5, and 7x7. The performance of these models is illustrated in Figure 11 and Table 5.

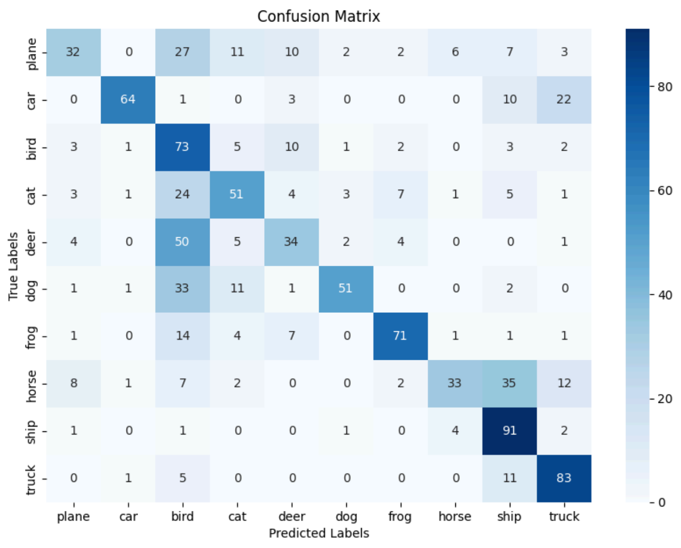
2x2 kernel size CNN model



a)



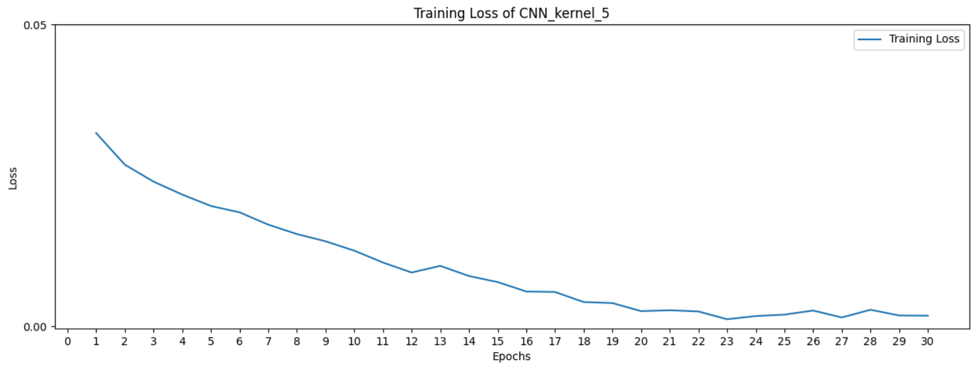
b)



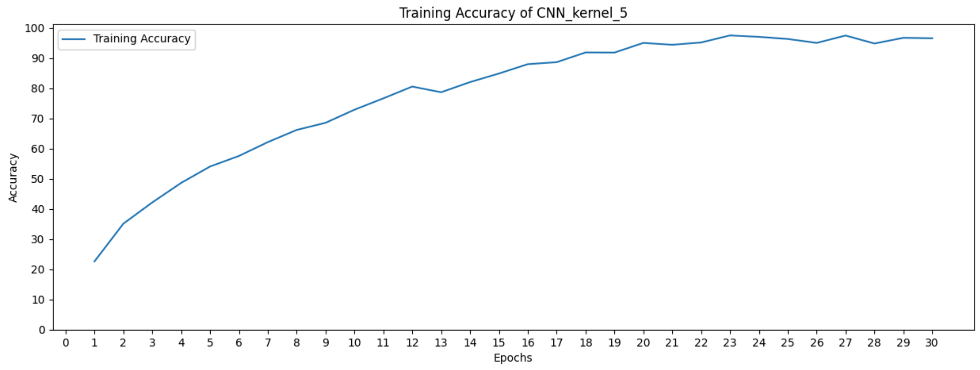
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	32	0	27	11	10	2	2	6	7	3
car	0	64	1	0	3	0	0	0	10	22
bird	3	1	73	5	10	1	2	0	3	2
cat	3	1	24	51	4	3	7	1	5	1
deer	4	0	50	5	34	2	4	0	0	1
dog	1	1	33	11	1	51	0	0	2	0
frog	1	0	14	4	7	0	71	1	1	1
horse	8	1	7	2	0	0	2	33	35	12
ship	1	0	1	0	0	1	0	4	91	2
truck	0	1	5	0	0	0	0	0	11	83

c)

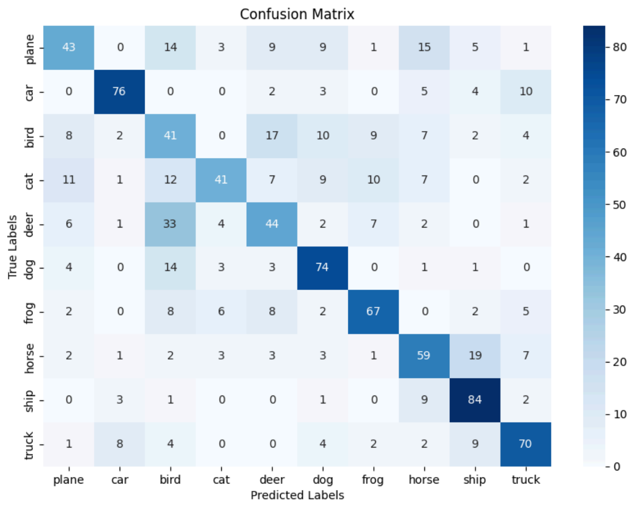
5x5 kernel size CNN model



d)

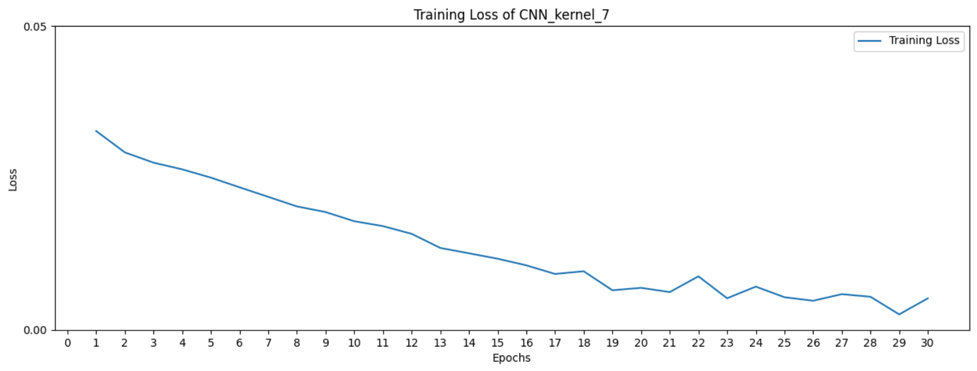


e)

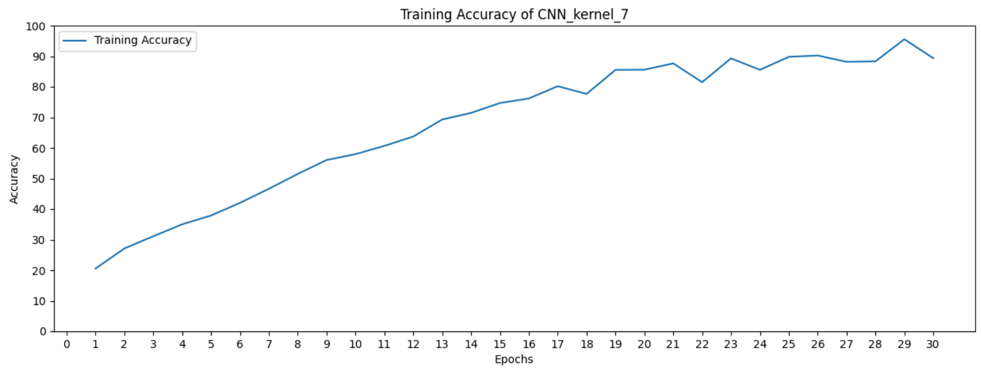


f)

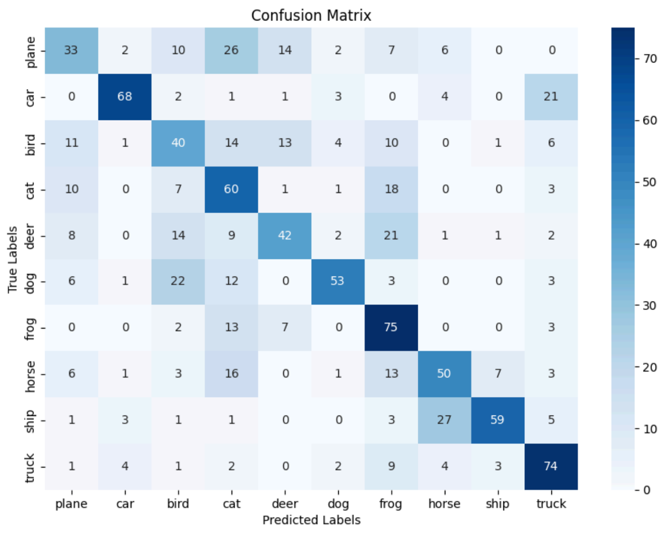
7x7 kernel size CNN model



g)



h)



i)

Figure 11. The performance of the CNN models with various kernel sizes (2x2, 5x5, 7x7). a-c) The training loss, accuracy rate, and confusion matrix of the 2x2 kernel size CNN model. d-f) The training loss, accuracy rate, and confusion matrix of the 5x5 kernel size CNN model. g-i) The training loss, accuracy rate, and confusion matrix of the 7x7 kernel size CNN model.

Table 5: Evaluation metrics for CNN models with different kernel sizes

Model Name	Accuracy	Precision	Recall	F1 Score
Default CNN model	64%	63.55%	64.00%	63.50%
2x2 kernel size CNN model	58.30%	65.03%	58.30%	58.20%
5x5 kernel size CNN model	59.90%	60.86%	59.90%	59.73%
7x7 kernel size CNN model	55.40%	58.47%	55.40%	55.65%

Figure 11 shows that a model converges faster with a smaller kernel size than with a larger kernel size. For example, when the training accuracy threshold is set to 90%, the CNN model with a 2x2 kernel takes 16 epochs to achieve 90% accuracy, while the CNN models with 5x5 and 7x7 kernels take 18 and 29 epochs, respectively. Therefore, increasing the kernel size can be more time-consuming and expensive when training the model.

From the confusion matrices, we observed that the model with the 2x2 kernel performs better at classifying bird images compared to the default CNN model and the CNN models with larger kernel sizes. This could be because the smaller kernel size allows the model to capture small details, such as edges and textures. In particular, the 2x2 kernel enables the model to learn details like birds' beaks and feathers. However, models with larger kernels perform better at classifying objects that differ in shape and size, as they focus on larger areas.

According to Table 5, the 2x2 kernel size CNN model has lower accuracy than the 5x5 kernel size CNN model. However, it achieves a better precision rate, as it tends to focus on smaller areas and tiny details of images. The 7x7 kernel size CNN model has the worst performance. This could be due to the added padding to the input, which prevents the kernel from being larger than the input size. However, this padding may introduce noise into the data and negatively affect the model's performance.

After experimenting with various models that use different numbers of layers and kernel sizes, we concluded that the default CNN model with 11 layers (8 convolutional layers and 3 fully connected layers) and a kernel size of 3 achieves the best performance among its variants, with an accuracy rate of 64%. Although we expected the CNN model to outperform the other models (e.g., Naive Bayes, Decision Tree, MLP), its performance was only better than that of the Decision Tree classifier. As a result, we trained the model on the entire CIFAR-10 dataset, which led to an accuracy rate of approximately 83.00% (Figure 12). Therefore, we can conclude that the CNN model has the potential to perform well with larger datasets.

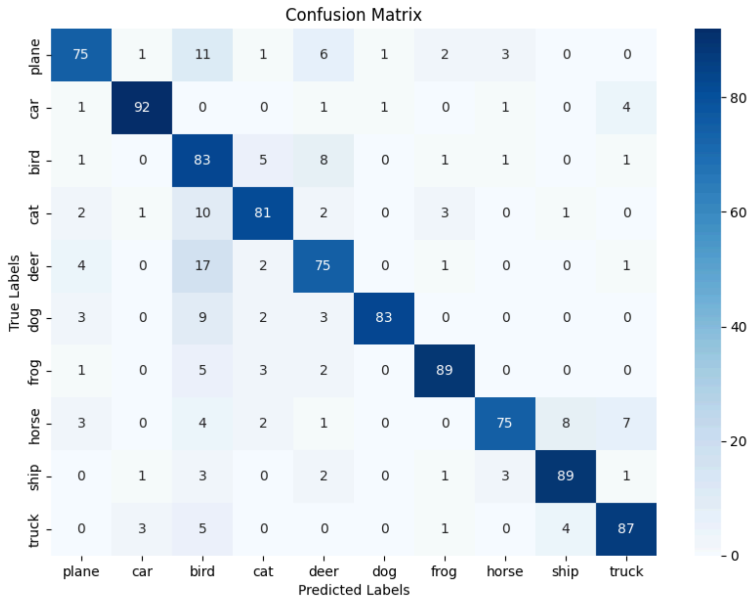


Figure 12. The performance of the default CNN model trained on the entire CIFAR-10 dataset.

5. Overall Model Performance

Table 6. Comparison of Performance of All Models

Model Type	Model Name	Accuracy	Precision	Recall	F1 Score
Naïve Bayes	**Self-defined NB model**	78.80%	79.30%	78.80%	78.87%
	Self-defined NB model	78.80%	79.30%	78.80%	78.87%
Decision Tree	Self-defined DT model (depth 5)	54.90%	56.26%	54.90%	54.15%
	Self-defined DT model (depth 35)	57.70%	58.06%	57.70%	57.77%
	***Self-defined DT model (depth 50)**	57.90%	58.02%	57.90%	57.92%
	Scikit's DT model (depth 50)	57.40%	57.42%	57.40%	57.28%
	Self-defined DT model (depth 75)	57.70%	58.06%	57.70%	57.77%
	Self-defined DT model (depth 100)	57.70%	58.06%	57.70%	57.77%

Multi-Layer Perceptron	**MLP (hidden layer=1, layer size=512)**	81.90%	82.09%	81.90%	81.88%
	MLP (hidden layer=2, layer size=16)	77.90%	78.40%	77.90%	77.87%
	MLP (hidden layer=2, layer size=128)	77.90%	78.40%	77.90%	77.87%
	*MLP (hidden layer=2, layer size=512)	78.90%	78.99%	78.90%	78.79%
	MLP (hidden layer=2, layer size=1024)	77.90%	78.40%	77.90%	77.87%
	MLP (hidden layer=3, layer size=512)	80.40%	80.88%	80.40%	80.36%
	MLP (hidden layer=5, layer size=1024)	80.70%	81.01%	80.70%	80.63%
	MLP (hidden layer=7, layer size=512)	77.90%	78.40%	77.90%	77.87%
Convolution Neural Network	**Default CNN (layer=11, kernel size=3)**	64.00%	63.55%	64.00%	63.50%
	4-layered CNN (layer=4, kernel size=3)	54.00%	54.10%	54.00%	52.77%
	6-layered CNN (layer=6, kernel size=3)	60.20%	62.45%	60.20%	59.44%
	9-layered CNN (layer=9, kernel size=3)	62.90%	63.88%	62.90%	62.35%
	12-layered CNN (layer=12, kernel size=3)	59.60%	63.52%	59.60%	59.30%
	13-layered CNN (layer=13, kernel size=3)	60.00%	60.29%	60.00%	59.18%
	CNN (layer=11, kernel size=2)	58.30%	65.03%	58.30%	58.20%
	CNN (layer=11, kernel size=5)	59.90%	60.84%	59.90%	59.73%
	CNN (layer=11, kernel size=7)	55.40%	58.47%	55.40%	55.65%

*Best model within the category is in bold.

After examining all models, we found that MLP models have an overall consistent best performance for this specific task. Among all MLP models, the MLP model with 1 hidden layer (size=512) has the best performance. Moreover, both NB models demonstrated the same level of performance, both default DT & CNN models had the most optimal performance within its class. Moreover, the default CNN model is the only one in the list that has a higher recall rate than precision rate. This means that the model is good at detecting the presence of classes, but it also uses patterns and features from other classes, which leads to misclassification.

Conclusion

In this project, we used 5000 CIFAR-10 images to train various AI models to classify input images. We first compared the performance between self-implemented simple algorithms including Naïve Bayes and Decision Tree with models from Scikit library. Although the implementation of the algorithms was not exactly the same, both models demonstrated a similar performance in classifying images. Moreover, we also compared the performance between simple models, such as Naïve Bayes and Decision Tree, and complex models, such as Multi-Layer Perceptron and Convolution Neural Network. The neural network models in general have a better performance in this type of complex tasks, because unlike simple models, which ignore the relationship between features, neural network models are supposed to better extract object features from images. Among all four types of models, MLP has the best performance, and DT has the worst. The CNN model is supposed to have the most optimal performance due to its complex design nature, but due to the limited training data size and pre-processing procedures, it did not achieve the performance as expected. Therefore, if we want to obtain a model with better performance than MLP in the future, we should keep working on improving the CNN model.

Reference

[1] Accessed: Nov. 27, 2024. [Online]. Available: https://www.design-reuse.com/articles/53213/artificial-intelligence-and-machine-learning-based-image-processing.html

[2] "AI in Image Processing," GeeksforGeeks. Accessed: Nov. 27, 2024. [Online]. Available: https://www.geeksforgeeks.org/ai-in-image-processing/

[3] V. Akstinaite, "Understanding hubris and heuristics in CEO decision-making: Implications for management," *Organ. Dyn.*, vol. 52, no. 2, p. 100978, Apr. 2023, doi: 10.1016/j.orgdyn.2023.100978.

[4] "CIFAR-10 and CIFAR-100 datasets." Accessed: Nov. 24, 2024. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html