Artificial Intelligence

Image Classification

Hadi Wehbi

Abdullah Itani

GitHub: https://github.com/Abdullah1tani/Image-Classification

Introduction

This report outlines the process of training and evaluating several machine-learning models to classify a subset of 5000 images in the CIFAR-10 dataset into ten categories.

We implemented, trained, and tested the following models:

1. **Naive Bayes**: A Manual implementation and Scikit-Learn version.
2. **Decision Tree**: A manual implementation with the Gini index alongside Scikit-Learn's version.
3. **Multi-Layer Perceptron (MLP)**: A three-layer neural network with experiments on varying depths and hidden layer sizes.
4. **Convolutional Neural Network (CNN)**: Based on the VGG11 architecture, with modifications to experiment with kernel sizes and network depth.

Each model was evaluated using accuracy, precision, recall, F1-score, and confusion matrices. The report also explores how feature extraction (using ResNet-18) and PCA influenced performance, and how changes to model architecture affected learning and generalization.

Model Architectures and Training

1. **Naive Bayes**

   We implemented a manual Gaussian Naive Bayes structure where we calculated priors and likelihoods explicitly, and provided a clear and customizable implementation of the Naive Bayes algorithm.

   The Sklearn Implementation used ‘GaussianNB’ from Scikit-learn, which is an efficient, pre-built implementation. This approach streamlined model training and enabled easier integration into machine learning pipelines.

   Naive Bayes does not require explicit training in the conventional sense. Instead, the probabilities are directly computed from the data.

2. **Decision Trees**
   We implemented a manual decision tree step-by-step, and then we split the data based on Gini impurity. The manual approach provided transparency but required computational effort for the dataset.

   The Sklearn Implementation used the ‘DecisionTreeClassifier’ from Scikit-learn. This variant included Scikit-learn optimizations and additional features like max depth and minimum samples per split, making it more efficient and versatile.

   While training the models, the manual decision tree was constructed recursively, splitting nodes based on Gini impurity until reaching the maximum depth or stopping conditions. The Sklearn version automated these steps, providing control over hyperparameters like maximum depth and minimum samples per split.

3. **Multi-Layer Perceptron (MLP)**

   The **three-layer MLP** model contains 3 hidden (linear) layers, 2 ReLU, and 1 BatchNorm layer. The first layer maps the input to a hidden layer, followed by ReLU activation. The second layer maps to another hidden layer, and applies BatchNorm to stabilize training followed by ReLU activation. Finally, the output layer maps to the output size.

The **shallow MLP** model contains 2 hidden (linear) layers, 1 ReLU activation layer, and no BatchNorm. The first layer maps the input to a hidden layer, followed by ReLU activation, and then mapped directly to the output size.

The **intermediate MLP** model contains 4 hidden (linear) layers, 3 ReLU activation layers, and 1 BatchNorm layer. The first layer maps the input to a hidden layer, followed by ReLU activation. The second layer maps the input to another hidden layer, followed by ReLU activation. The third layer maps to another hidden layer and applies BatchNorm to stabilize training followed by ReLU activation. Finally, the output layer maps to the output size.

The **deep MLP** model contains 5 hidden (linear) layers, 3 ReLU activation layers, and 1 BatchNorm layer. The first layer maps the input to a hidden layer, followed by ReLU activation. The second layer maps the input to another hidden layer, followed by ReLU activation. The third layer maps the input to a hidden layer, followed by ReLU activation. The fourth layer maps to another hidden layer and applies BatchNorm to stabilize training followed by ReLU activation. Finally, the output layer maps to the output size.

The **small-layer**, **moderate-layer**, and **large-layer** MLPs use the same architecture as the three-layer MLP but have different hidden layer sizes. The small layer MLP has 128 neurons in the first hidden layer and 64 neurons in the second hidden layer. The moderate layer MLP has 256 neurons in the first hidden layer and 128 neurons in the second hidden layer. The large layer MLP has 512 neurons in the first hidden layer and 256 neurons in the second hidden layer.

To train the MLP models, we used stochastic gradient descent (SGD) with a learning rate of 0.01 and a momentum of 0.9 to optimize the model's parameters. The loss function used was cross-entropy loss, which is well-suited for classification tasks. The models were trained for 20 epochs using a data loader with a batch size of 64, which fed mini-batches of features and labels during each epoch. For each batch, the optimizer’s gradients were reset, predictions were computed by passing the input features through the model, and the loss was calculated by comparing the predictions to the true labels. The gradients of the loss were then back-propagated through the network, and the optimizer updated the model’s weights accordingly. The training dataset consisted of feature vectors derived from 500 samples per class from the CIFAR-10 dataset, reduced to 50 dimensions using PCA. At the end of each epoch, the average loss across all batches was printed to monitor training progress. Upon completion of training, the model weights were saved to files named after each model for reuse during the evaluation process.

4. Convolutional Neural Networks (CNNs)

The **VGG11** model (**default CNN**) consists of 8 blocks: 8 convolutional layers, each followed by BatchNorm and ReLU activations, and 5 MaxPooling layers to reduce spatial dimensions from 32x32 to 1x1. The convolutional layers gradually increase the number of channels from 3 to 512. The classifier includes 3 linear layers (1 with 512 neurons and 2 with 4096 neurons) with 2 ReLU activations and 2 dropouts. The classifier outputs the final class predictions. The VGG11 model has a kernel size of 3x3.

The **Shallow CNN** model consists of 4 blocks: 4 convolutional layers, each followed by BatchNorm and ReLU activations, and 4 MaxPooling layers to reduce spatial dimensions. The convolutional layers gradually increase the number of channels from 3 to 512. The classifier includes 3 linear layers (1 with 512 neurons and 2 with 4096 neurons) with 2 ReLU activations and 2 Dropout layers. The classifier outputs the final class predictions. The Shallow CNN model has a kernel size of 3x3.

The **Deep CNN** model consists of 12 blocks: 12 convolutional layers, each followed by BatchNorm and ReLU activations, and 5 MaxPooling layers to reduce spatial dimensions. The convolutional layers gradually increase the number of channels from 3 to 512. The classifier includes 3 linear layers (1 with 512 neurons and 2 with 4096 neurons) with 2 ReLU activations and 2 Dropout layers. The classifier outputs the final class predictions. The Deep CNN model has a kernel size of 3x3.

The **Large CNN** and **Very Large CNN** models are created from the same structure (LargeCNN) which consists of 8 blocks: 8 convolutional layers, each followed by BatchNorm and ReLU activations, and 5 MaxPooling layers to reduce spatial dimensions. The convolutional layers gradually increase the number of channels from 3 to 512. The classifier includes 3 linear layers (512 neurons, 2048 neurons, and 1024 neurons) with 2 ReLU activations and 2 dropouts. The classifier outputs the final class predictions. The Large CNN model has a kernel size of 5x5 and the Very Large CNN model has a kernel size of 7x7.

To train the CNN models, we used stochastic gradient descent (SGD) with a learning rate of 0.01 and a momentum of 0.9 to optimize the model's parameters. The loss function used was cross-entropy loss, which is well-suited for classification tasks. The models were trained for 20 epochs using a data loader with a batch size of 64 to feed mini-batches of features and labels during each epoch. For each batch, the optimizer's gradients were reset, predictions were computed by passing the input features through the model, and the loss was calculated by comparing the predictions to the true labels. The gradients of the loss were then back-propagated through the network, and the optimizer updated the model's weights accordingly. At the end of each epoch, the average loss across all batches was printed to monitor training progress. The training dataset consisted of 500 samples per class from the CIFAR-10 dataset, augmented with random horizontal flipping and cropping, followed by normalization. At the end of each epoch, the average loss across all batches was printed to monitor training progress. Upon completion of training, the model weights were saved to a file named after the respective model for reuse during the evaluation process.

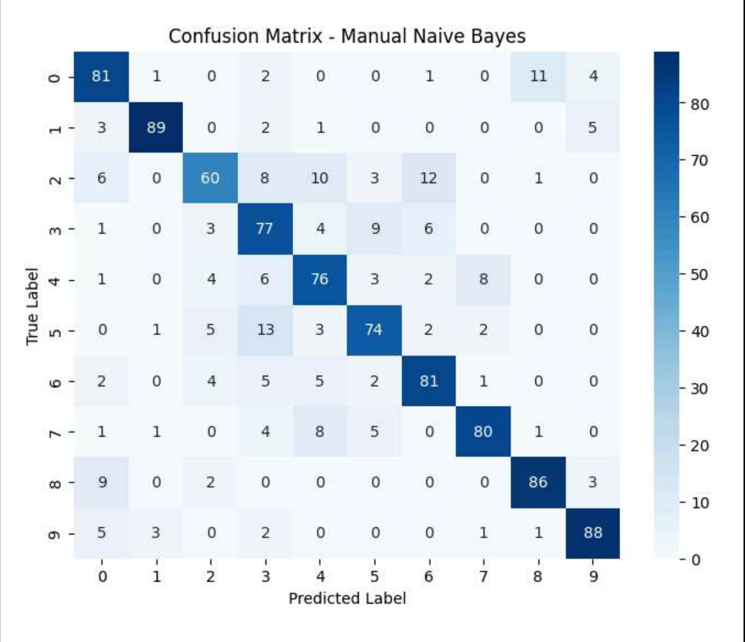
Evaluation Metrics For All Models

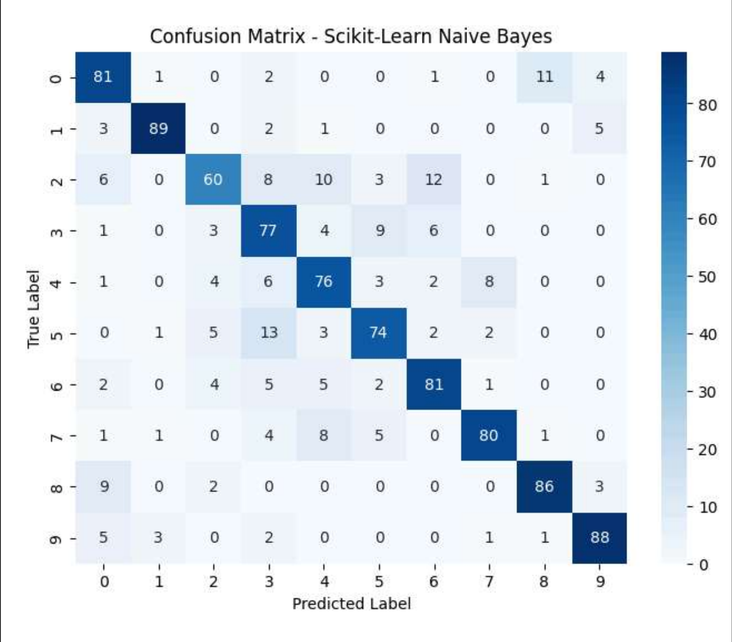
MODEL	ACCURACY	PRECISION	RECALL	F1 SCORE
Naive Bayes (Manual)	79.2%	79.6578%	79.2%	79.2215%
Naive Bayes (Sklearn)	79.2%	79.6578%	79.2%	79.2215%
Decision Tree (Manual)	58.9%	59.2029%	58.9%	58.9655%
Decision Tree(Sklearn)	58.4%	58.6774%	58.4%	58.4303%
Three-Layer MLP	83.8%	84.2453%	83.8%	83.8649%
Shallow MLP(Depth)	84%	84.26842%	84%	84.0032%
Intermediate MLP(Depth)	83.1%	83.2557%	83.1%	83.0918%
Deep MLP (Depth)	83.1%	83.3365%	83.1%	83.1129%
Small Hidden Layers (MLP size)	80.7%	80.7835%	80.7%	80.6650%
Moderate Hidden Layers (MLP size)	80.5%	80.7627%	80.5%	80.4514%
Large Hidden Layers (MLP size)	80.7%	81.1447%	80.7%	80.7325%
CNN (VG11) entire dataset	82.63%	83.13%	82.63%	82.58%
CNN (VGG11) subset	68.2%	67.9513%	68.2%	67.4675%
Shallow CNN (4 blocks)	64.5%	67.0613%	64.5%	64.2881%
Deep CNN (12 blocks)	68.1%	68.9494%	68.1%	67.0847%
Large Kernels (7x7)	57.6%	59.8610%	57.6%	56.1826%
Very Large Kernels (9x9)	54.4%	53.1708%	54.4%	52.1911%

Model Performance Analysis and Insights

Naive Bayes

The manual and Scikit-Learn implementations of Naive Bayes performed consistently, achieving **79.2%** accuracy with comparable precision, recall, and F1 scores. This highlights the simplicity of Naive Bayes, which relies on the assumption that features are independent. While computationally efficient, this assumption limits its ability to model complex relationships between features.

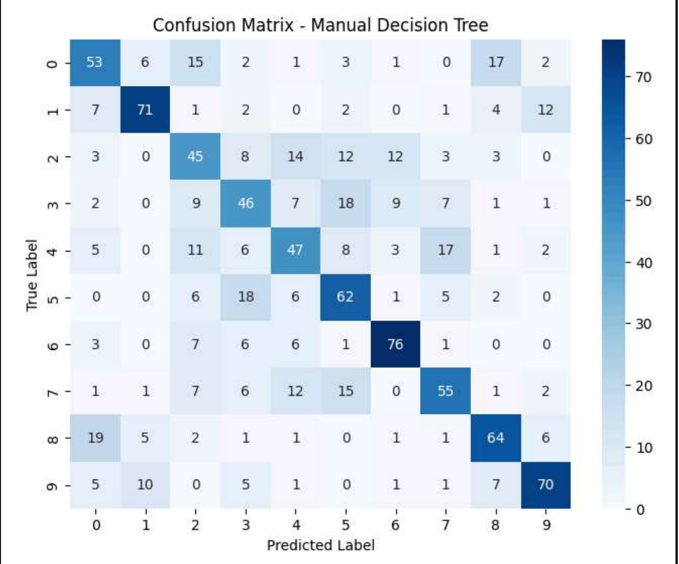


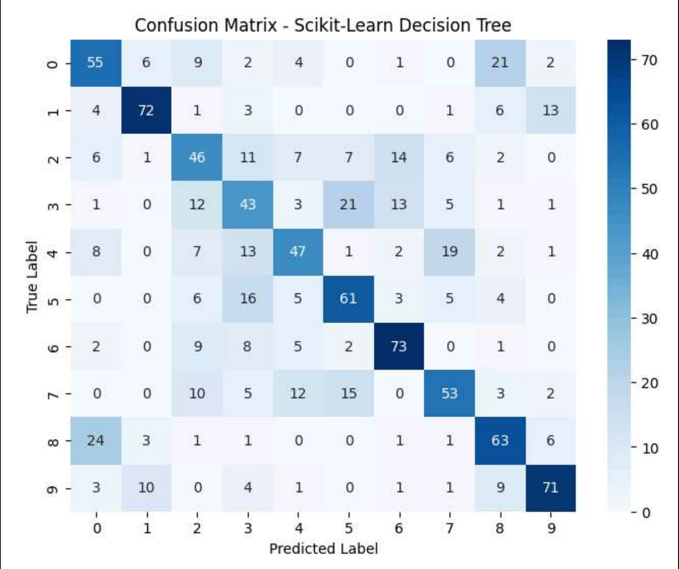


The confusion matrices show that both Gaussian Naive Bayes models performed well, as most predictions lie along the diagonal. In addition, we can see that the model shows the most misclassification in classes 2 (birds), 3 (cats), 4 (deer), and 5 (dogs) showing an accuracy of around 60-75% while performing very well in classifying the remaining classes by showing an accuracy around 80-95%.

Decision Trees

Decision Trees fell behind other models, with manual and Scikit-Learn implementations achieving accuracies of **58.9%** and **58.4%**, respectively. Their tendency to overfit, especially when data is limited, reduces their ability to generalize effectively. The low recall shows frequent misclassifications, making them less reliable for tasks like facial image recognition, where differences between classes play a crucial role.





The confusion matrices show that both Decision tree models performed poorly, as most predictions lie outside of the diagonal. In addition, we can see that the model shows the most misclassification in classes 0 (airplane), 2 (birds), 3 (cats), 4 (deer), 5 (dogs), and 7 (horse) showing an accuracy of around 45-55% while performing better in classifying the remaining classes by showing an accuracy around 60-70%.

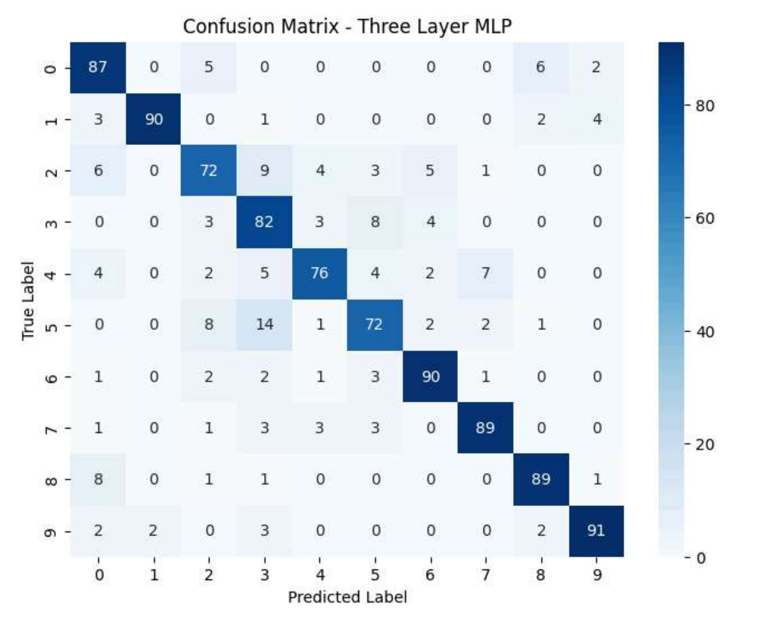
MLPs

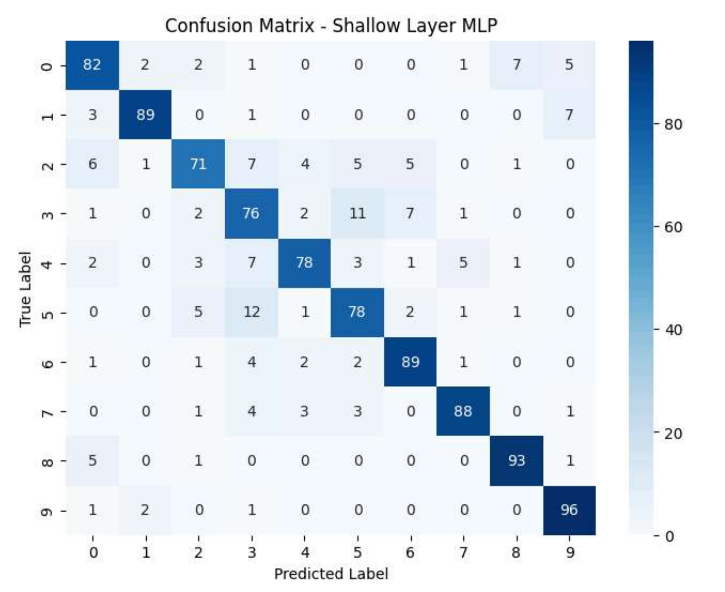
MLPs outperformed Naive Bayes and Decision Trees and demonstrated strong capabilities in handling non-linear relationships in the data:

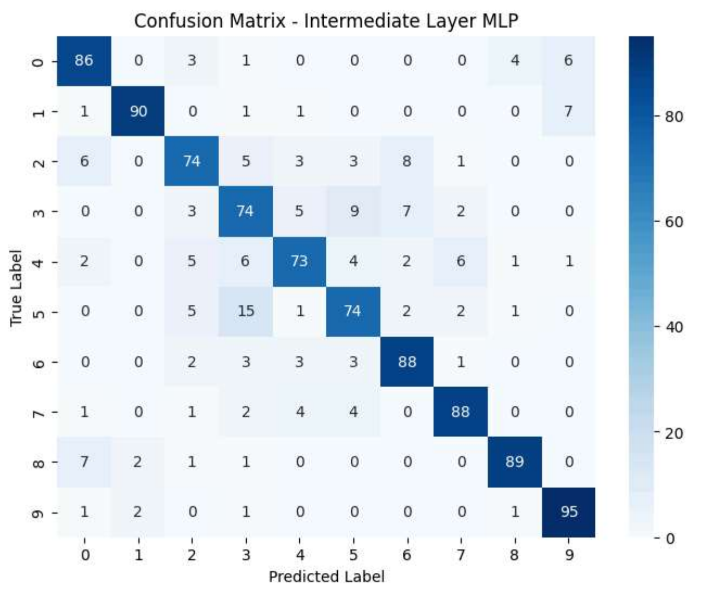
- **Three-Layer MLP (83.8%)**: Achieved impressive accuracy, effectively balancing precision, recall, and F1 scores. This performance highlights its ability to manage the complexity of the dataset.

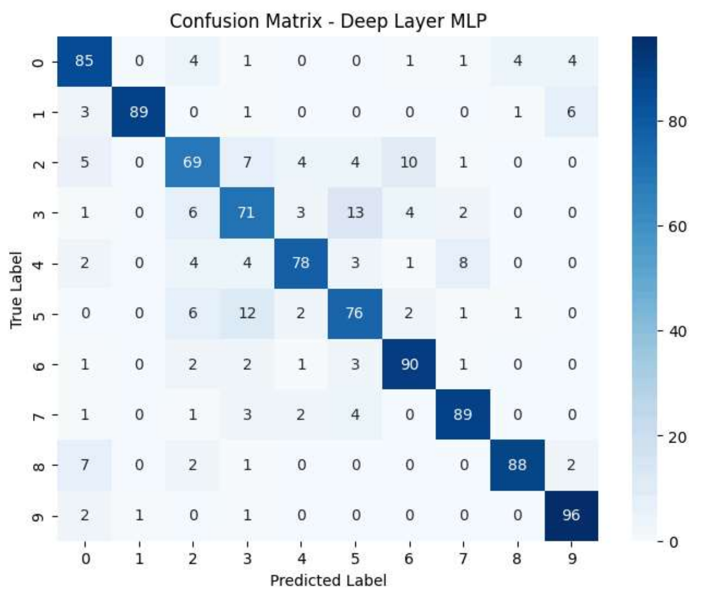
- **Shallow MLP (84%)**: The Shallow MLP has almost the same accuracy as the Three-Layer MLP, meaning that fewer layers can still yield good results by reducing the risk of overfitting.
- **Intermediate MLP (83.1%)** and **Deep MLP (83.1%)**: These models showed slight declines in performance compared to the Three-Layer and Shallow configurations, redundant feature learning might have impacted these deeper networks. In addition, both showed the same accuracy and recall even though deep MLP contains one more layer than intermediate MLP but deep MLP has a slightly higher precision and F1 score than intermediate MLP.
- **Small Hidden Layers MLP (80.7%)**: Delivered good performance, showing great ability to handle the dataset even with its small hidden layer size (64 and 128).
- **Moderate Hidden Layers MLP (80.5%)**: Provided a good accuracy, showing no significant advantage over smaller or larger configurations.
- **Large Hidden Layers MLP (80.7%)**: Matched the performance of the small hidden layers MLP, but slightly higher F1 score and precision.

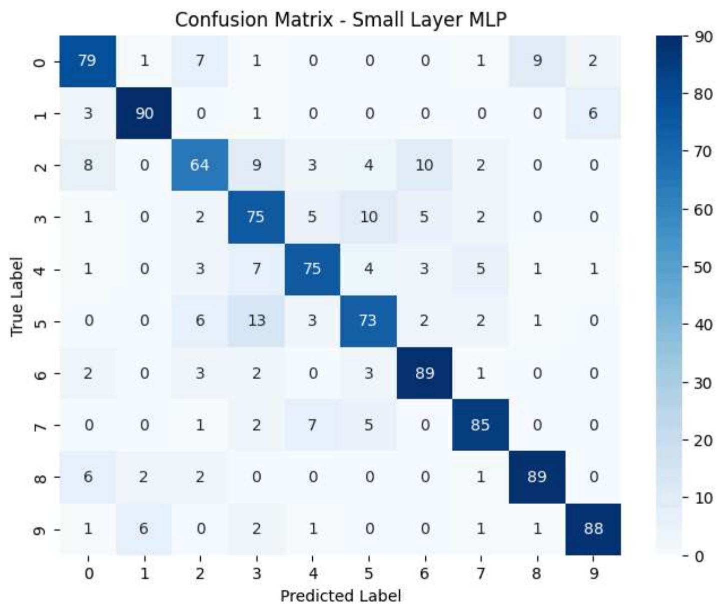
Since all three hidden layer variations of MLPs delivered almost the same metrics, this indicates that increasing size did not lead to noticeable gains or overfitting. In addition, we have noticed after retraining the size MLP models several times we get very different results, for example, for some training, the small hidden layer MLP showed higher accuracy than the moderate and large hidden layer MLP, and another training showed that moderate hidden layer MLP had a higher accuracy than small and large hidden layer MLP, and other trainings showed that large hidden layer MLP had a higher accuracy than the small and moderate hidden layer MLP. This big difference in results is because the subset we are training the MLPs on is small, thus they don't have enough data to be trained, resulting in inconsistent results among the hidden layer size variation MLPs.

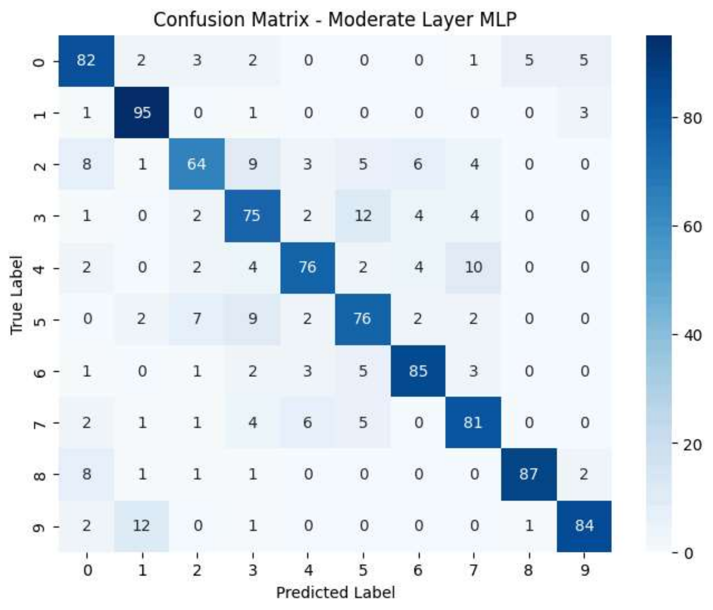


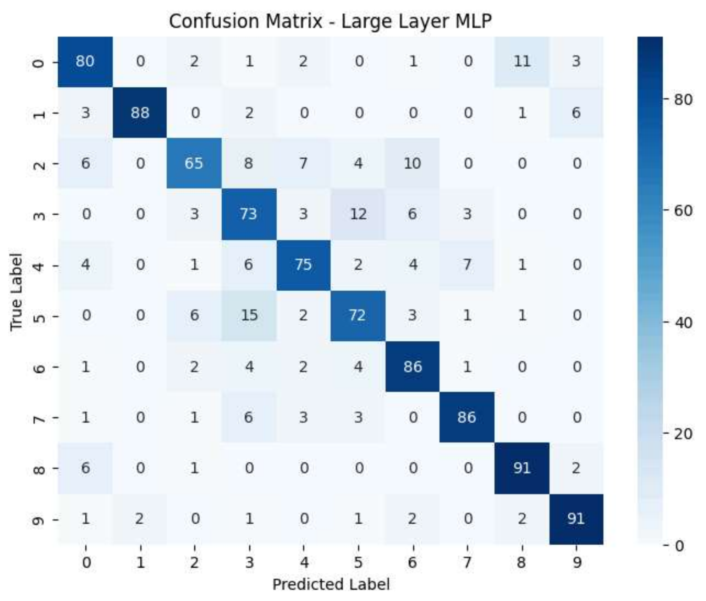



Confusion Matrix - Intermediate Layer MLP
True Label
Predicted Label
0: 86 0 3 1 0 0 0 0 4 6
1: 1 90 0 1 1 0 0 0 0 7
2: 6 0 74 5 3 3 8 1 0 0
3: 0 0 3 74 5 9 7 2 0 0
4: 2 0 5 6 73 4 2 6 1 1
5: 0 0 5 15 1 74 2 2 1 0
6: 0 0 2 3 3 3 88 1 0 0
7: 1 0 1 2 4 4 0 88 0 0
8: 7 2 1 1 0 0 0 0 89 0
9: 1 2 0 1 0 0 0 0 1 95
0 1 2 3 4 5 6 7 8 9
0 20 40 60 80


Confusion Matrix - Deep Layer MLP
True Label
Predicted Label
0: 85 0 4 1 0 0 1 1 4 4
1: 3 89 0 1 0 0 0 0 1 6
2: 5 0 69 7 4 4 10 1 0 0
3: 1 0 6 71 3 13 4 2 0 0
4: 2 0 4 4 78 3 1 8 0 0
5: 0 0 6 12 2 76 2 1 1 0
6: 1 0 2 2 1 3 90 1 0 0
7: 1 0 1 3 2 4 0 89 0 0
8: 7 0 2 1 0 0 0 0 88 2
9: 2 1 0 1 0 0 0 0 0 96
0 1 2 3 4 5 6 7 8 9
0 20 40 60 80


Confusion Matrix - Small Layer MLP
True Label
Predicted Label
0: 79 1 7 1 0 0 0 1 9 2
1: 3 90 0 1 0 0 0 0 0 6
2: 8 0 64 9 3 4 10 2 0 0
3: 1 0 2 75 5 10 5 2 0 0
4: 1 0 3 7 75 4 3 5 1 1
5: 0 0 6 13 3 73 2 2 1 0
6: 2 0 3 2 0 3 89 1 0 0
7: 0 0 1 2 7 5 0 85 0 0
8: 6 2 2 0 0 0 0 1 89 0
9: 1 6 0 2 1 0 0 1 1 88
0 1 2 3 4 5 6 7 8 9
0 10 20 30 40 50 60 70 80 90


Confusion Matrix - Moderate Layer MLP
True Label
Predicted Label
0: 82 2 3 2 0 0 0 1 5 5
1: 1 95 0 1 0 0 0 0 0 3
2: 8 1 64 9 3 5 6 4 0 0
3: 1 0 2 75 2 12 4 4 0 0
4: 2 0 2 4 76 2 4 10 0 0
5: 0 2 7 9 2 76 2 2 0 0
6: 1 0 1 2 3 5 85 3 0 0
7: 2 1 1 4 6 5 0 81 0 0
8: 8 1 1 1 0 0 0 0 87 2
9: 2 12 0 1 0 0 0 0 1 84
0 1 2 3 4 5 6 7 8 9
0 20 40 60 80

The confusion matrices show that all MLPs performed well, as most predictions lie along the diagonal. In addition, we can see that all models show misclassification in classes 2 (birds), 3 (cats), 4 (deer), and 5 (dogs) showing an accuracy of around 70-75% while performing very well in classifying the remaining classes by showing an accuracy around 85-95%. However, the MLPs with smaller hidden sizes (less than 512) showed slightly lower accuracy (65-70%) in these misclassified classes.

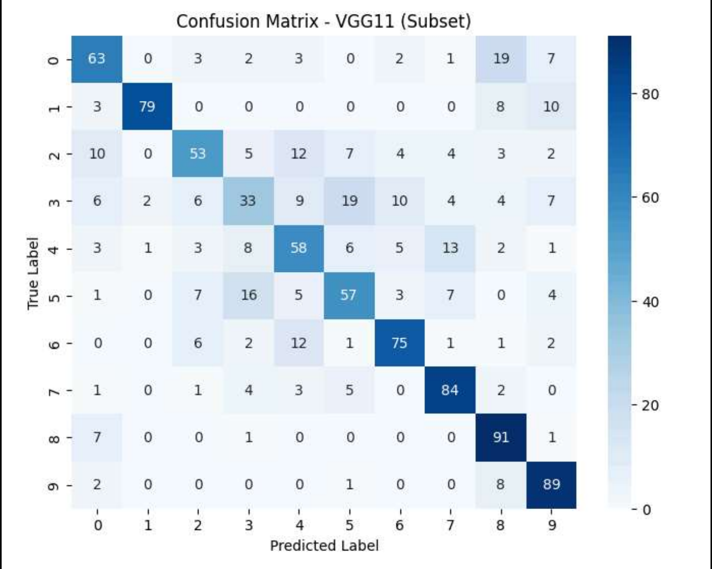
CNNs

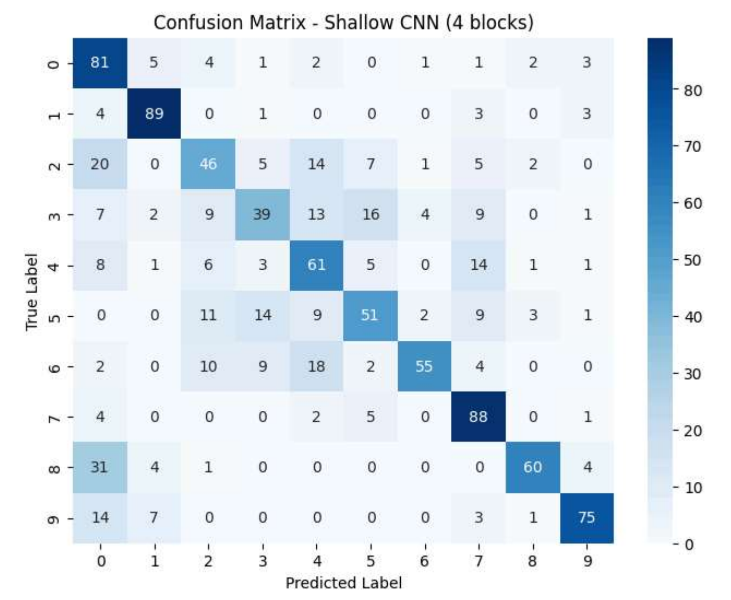
The CNNs have been trained by directly receiving the training images without feature extractions or dimensions reduction, The CNNs outperformed decision trees but still performed relatively poorly and lower than MLPs and Naive Bayes:

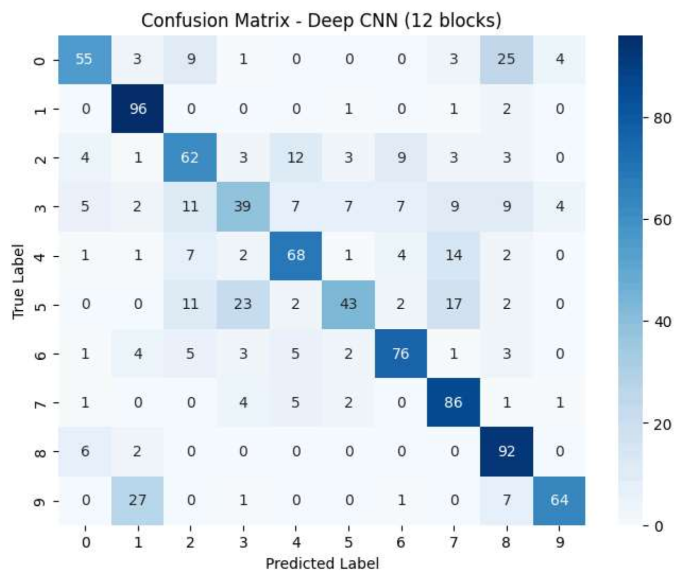
- **VGG11 model (Subset)(68.2%)**: Didn't perform very well but has the highest metrics between all other depth and size variations CNNs.
- **Shallow CNN (64.5%)**: Performed worse than the original VGG11 model because it has half the number of blocks (4) of the VGG11 (8) and one-third of the Deep CNN (12 blocks). Although it has a lower accuracy than the other 2 models, its accuracy is still very close to them. In addition, training the model took the least time out of all the CNNs due to its simplicity.
- **Deep CNN (68.1%)**: Didn't perform well and has almost the same accuracy as VGG11 even though it has 4 additional blocks than the VGG11 model. Took the most

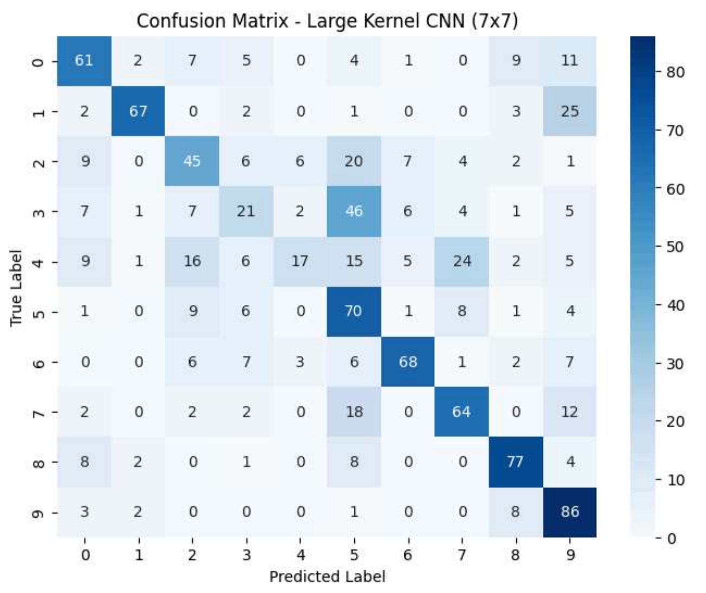
time to train among the 3 models and yet still provided 2nd best performance out of these 3.

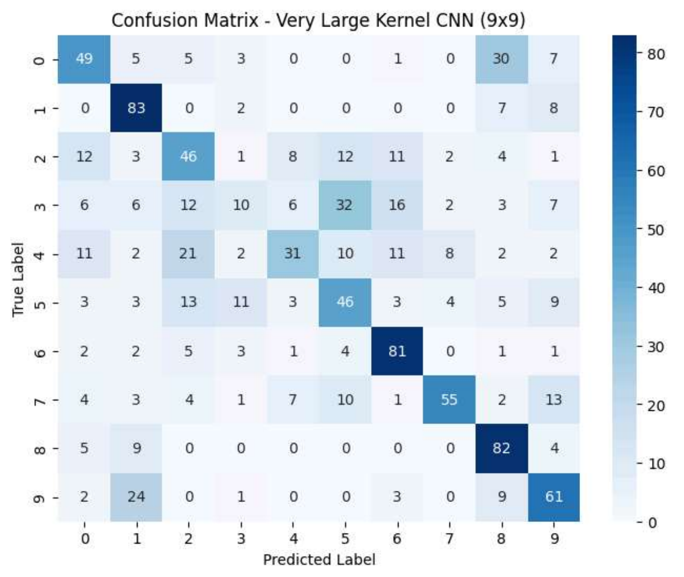
- **Large Kernels (7x7)(57.6%)**: Performed worse than the 3x3 kernel size VGG11 model. Despite a higher kernel size than the VGG11, this CNN did not only fail to show improvement but also showed lower metrics.
- **Very Large Kernels (9x9)**: Performed the worst out of all CNN models. Despite having a very large kernel size that is bigger than all CNN depth and kernel size variations, this CNN showed lower metrics and didn't show any improvement with the additional kernel size.











The confusion matrices show that all CNNs performed poorly, as most predictions lie outside the diagonal. In addition, we can see that all models show huge misclassification in classes 2 (birds), 3 (cats), 4 (deer), and 5 (dogs) showing an accuracy of around 40-60% while performing better in classifying the remaining classes by showing an accuracy around 65-85%. However, the CNNs with bigger kernel sizes (7x7 and 9x9) showed lower accuracy (10-30%) in classes 3 (cats) and 4 (deer).

Classes classifications

Upon reviewing all confusion matrices, we found out that all models poorly classified the classes 2 (birds), 3 (cats), 4 (deer), and 5 (dogs), and were able to well recognize the other classes. The models poorly classified these classes because they might have a lot of visual similarities, which makes it harder for the models to differentiate between them. For example, the model might not be able to properly differentiate between a cat and a dog due to them being very similar to each other since they share similarities like physical shape (both have four legs) and fur. In addition, since the dataset is small, each animal class has more variety, for example, birds have several species and each bird can have a different shape and color (eagles and penguins are the same species but they look very different), which might confuse the models and since we are training the models on a small dataset, training them on different species is a hard task. Training these models on a bigger dataset can have better results. The other classes that were well recognized by the models are 0 (airplane), 1 (automobile), 6 (frog), 7 (horse), 8 (ship), and 9 (truck). Since these classes have unique visual features, the models were able to recognize them and have higher classifying accuracy, for instance, each of the animal classes (horses and frogs) has their unique physical shape. In addition, even with a small dataset, these classes don't have a lot of variety among them (ships mostly look similar across images) which makes training the models to classify these classes easier.

Depth's Role in Performance

- **Decision Trees**: Increasing depth led to overfitting, reducing their generalization ability. Adjusting the max depth in our Decision Tree had a noticeable impact on the model's performance. With a higher max depth (50), the tree could capture more details in the training data, which improved training accuracy but led to overfitting. While the model performed well on the training set, it struggled to generalize to the test set. On the other hand, reducing the max depth to a lower value (10 or 20) simplified the model. This prevented overfitting, as the tree focused only on the most important patterns, resulting in slightly lower training accuracy but more stable test performance. Overall, a balanced max depth helped our model generalize better, showing that tuning this parameter is key to achieving reliable results.
- **MLPs**: The results show that shallow MLP performed the best with a leading accuray of 84% and the Three-Layer MLP close behind at 83.8%. The Intermediate and Deep MLPs (both at 83.1%), were still strong but didn't outperform the simpler configurations, suggesting that going too deep didn't add much value and could risk overfitting. These results emphasize the importance of not overcomplicating the architecture—sometimes, simpler models can be just as effective, if not better, especially when they're well-tuned.

- **CNNs**: CNNs demonstrated varying performance based on depth. The VGG11 CNN (subset) achieved the highest accuracy among the depth variant CNNs variants at 68.2%, closely followed by VGG11 at 68.1%. However, the Shallow CNN (4 blocks) lagged behind at 64.5%. Interestingly, while deeper architectures slightly outperformed shallower ones, all CNNs struggled compared to simpler models like MLPs and Naive Bayes. In addition, the VGG11 model trained on the entire CIFAR 10 dataset achieved an impressive accuracy of 82.63%. This shows that with limited data, CNNs cannot fully utilize their ability to capture spatial features, and their reliance on larger datasets leaves them prone to overfitting when training samples are insufficient.

Effect of Layer and Kernel Size

MLPs:

- The hidden layer configurations (small, moderate, and large) showed comparable performance, with accuracy values ranging narrowly around 80.5-80.7%. This indicates that for this dataset, the choice of hidden layer size had minimal impact on the MLP's ability to generalize or perform well. These results suggest that the dataset’s complexity could be effectively captured by MLPs without requiring extensive adjustments to hidden layer sizes, reinforcing their robustness across different configurations.

CNNs:

- Despite the larger kernel size, both models (Large CNN, Very Large CNN) failed to capture additional meaningful patterns in the dataset, resulting in lower metrics. The reduced accuracy shows that the larger kernels struggled with localized feature extraction, which is critical for distinguishing patterns in this dataset. Instead of improving performance, the increase in kernel size led to a significant drop in metrics, emphasizing that larger kernels are not always better.

Summary of Findings

This project revealed the importance of tailoring model architecture and training methodology to the dataset's characteristics:

- CNNs demonstrated exceptional performance on larger datasets, with VGG11 achieving an accuracy of 82.63% when trained on the entire dataset. This highlights CNNs' ability to effectively capture spatial relationships and complex patterns in data, provided they have access to a large dataset with sufficient training samples. However, their performance dropped significantly on smaller subsets, reinforcing the importance of dataset size in leveraging CNNs' full potential.

- Naive Bayes and Decision Trees performed well, achieving 79.2% and 58.9% accuracy respectively. Their simplicity and computational efficiency made them reliable options for training and testing on the subset with limited training and testing samples.
- MLPs delivered robust results, balancing accuracy, precision, recall, and F1 scores effectively. The Three-Layer MLP achieved 83.8% accuracy, showcasing its ability to handle non-linear relationships in the data. Variants with moderate hidden layers or depth further refined performance, highlighting MLPs as versatile models that can excel with proper hyperparameter tuning.

In conclusion, while CNNs excel on larger datasets by effectively capturing spatial features and patterns, their performance drops when trained on smaller datasets due to their reliance on larger amounts of data to generalize effectively. Naive Bayes and Decision Trees, despite their simplicity, delivered solid and reliable results on smaller datasets, showcasing their straightforward design as an advantage for limited data scenarios. MLPs stood out as a flexible and robust option, consistently delivering strong results across different configurations when carefully tuned, even on smaller datasets.