Final Project Report

COMP 472

Image Classification

Presented to Dr. Kefaya Qaddoum

Michel Kandalaft, 40227791

Yassin Al Kafri, 40214482

We certify that this submission is our original work and meets the Faculty’s Expectations of Originality

Submitted on Monday November 25th, 2024.

Link to GitHub Repository: https://github.com/Mikeyy6/Project_COMP472.git

Introduction:

In this project, we were asked to apply different machine learning techniques such as the Gaussian Naïve Bayes, the Decision Tree, the Multi-Layer Perceptron (MLP) and the Convolutional Neural Network (CNN) to perform image classification on the CIFAR-10 dataset. The main goal of the project was to build, train and evaluate those models to accurately classify the given images in the dataset. The implementation of those models was made using Python and PyTorch libraries. Also, for the Gaussian Naïve Bayes as well as the Decision Tree, Scikit's Learn libraries were integrated to be compared with the custom model that we have implemented.

There were different key stages throughout the implementation of this image classification tool. First, we started by extracting the data, reducing its dimensions and preprocessing it into training data and test data. Once that was done, each model was implemented in its own directory to avoid confusion between the models. Finally, the main function, englobing all the implemented models, was designed to call all the models as well as compute the classification reports for each of the cases, print the evaluation metrics (Precision, Recall, Accuracy and F1-Score) and generate a confusion matrix with the results obtained for an easier comparison between the models.

Table of Contents

List of Figures

List of Tables

Model Architecture and Training:

The implementation of this project required designing four distinct models: 1) Gaussian Naïve Bayes, 2) Decision Tree, 3) Multi-Layer Perceptron (MLP) and 4) Convolutional Neural Network (CNN). In this section, a summary of each model will be provided as well as the training methodologies that were used to successfully implement the models.

First, the Gaussian Naïve Bayes model starts by assuming that the image features are independent from the class label. The way it is implemented is by performing probability calculation on the training data that is predefined as NumPy arrays. Then, the program will compute both mean and variance for each feature of a given class to then be able to calculate the probability density function and predict the testing data based on the training data. Moreover, the training methodology doesn't really apply to the GNB model since it is not an iterative model and doesn't require optimization. Therefore, we don't need to implement the number of epochs and the learning rate for this model. For the loss function, the GNB model follows a negative log of likelihood function to determine the real value of the loss function.

For the second model, the decision tree is implemented in a way that each internal node represents a feature. The changes that are made compared to the main model are mainly concerning the depth and the criterion of the tree. Therefore, the depth of the tree can be varied and can influence the test accuracy percentage of a given dataset and can also tend towards overfitting the data. The training methodology, same as the GNB model, doesn't really apply for the decision tree since it isn't an iterative model and no optimization is required. The only training methodology that is required for the decision tree model is concerning the loss function which minimizes the Gini Impurity index and deciding on the best split of the tree.

Next, the Multi-Layer Perceptron (MLP) is a fully connected neural network that is implemented using a given number of hidden layers. It uses an activation function that we have used in our implementation (ReLU). The changes that can be implemented compared to the main model are the variation of the depth of each layer and the number of hidden layers. We can also use different activation functions to run the implementation of the MLP model. For the training methodology, it is slightly different compared to the previous two models explained above since different hyperparameters are needed for the full implementation of the model. Those methodologies can be summarized by using 20 epochs based on the size of the dataset in our case, a learning rate of 0.001, the Cross-Entropy Loss function and the SGD optimizer with a momentum that is equal to 0.9.

Finally, the last model that was designed is the Convolutional Neural Network (CNN). The main CNN model consists of three types of layers: 1) Convolutional Layers, 2) Pooling Layers and 3) Fully Connected Layers. The changes that can be implemented to the main model are almost the same as the MLP Model where we can vary the number of convolution layers, the kernel size and the depth of the fully connected layers. Again, for the training methodologies, they are almost the same as the MLP model described previously. Since the CNN model is an iterative model, the use of epochs is used which we defined to be 10 in our implementation. The learning rate is also used and was set to 0.001. Finally, the loss function that was implemented used the Cross-Entropy Loss function and the optimizer that was used is the Adam optimizer.

Results & Discussion:

The following section highlights the results obtained after running each of the ML models that were designed. The evaluation metrics have been computed and provided in the tables below. For GNB and Decision Tree models, the classification report is provided in Appendix A of this report. Also, for the Multi-Layer Perceptron (MLP) and the Convolutional Neural Network (CNN), a detailed table showing the result of the loss function after each epoch as well as its accuracy can be found in Appendix A of this report. Moreover, the confusion matrices for each of the models that were implemented are attached as figures in Appendix B of this report.

1) Gaussian Naive Bayes:

	Accuracy	**Precision**	**Recall**	**F1-Score**
Custom GNB	0.7950	0.7991	0.7950	0.7952
Scikit's Learn GNB	0.7950	0.7991	0.7950	0.7952

Table 1: Evaluation Metrics for GNB

Analyzing the results obtained for the Gaussian Naïve Bayes Model, both the Custom GNB and the Scikit's Learn GNB yielded into identical results. The accuracy that has been achieved implementing this model is approximately 79.5% alongside almost 80% for the rest of the evaluation metrics indicating that both models are performing as expected on the CIFAR-10 dataset. The results of those two models are identical since the GNB algorithm is independent. Therefore, since both implementations are using similar probability calculations, both models will yield approximately the same results when performing the evaluation metrics.

2) **Decision Tree:**

	Accuracy	**Precision**	**Recall**	**F1-Score**
DT Depth 1	0.1970	0.0430	0.1970	0.0697
DT Depth 5	0.5460	0.5615	0.5460	0.5385
DT Depth 10	0.6160	0.6252	0.6160	0.6172
DT Depth 50	0.5870	0.5905	0.5870	0.5870
Scikit's Learn DT	0.6060	0.6094	0.6060	0.6064

Table 2: Evaluation Metrics for Decision Tree

First, when analyzing the results of the Decision Tree, we can observe that by varying the depth of the tree, the results slightly improve. As the table above summarizes, when the depth of the decision tree is set to 1, the evaluation metrics result in a poor performance of the model since it is too simple to capture the different complexities of the dataset.

Second, when rising the depth of the decision tree to 5 or 10, we can observe significant improvements on the model that is being trained. Both the accuracy and precision are increasing proportionally to the depth of the tree. We can also observe an improvement in the F1-score that ensures a better balance between precision and recall.

Moreover, when varying the depth of the tree to 50, we can observe a slight decrease in performance. This indicates that the model may have overfitted the training dataset, thus decreasing the accuracy of the model.

Finally, analyzing the results of the Scikit's Learn Decision Tree we can conclude that the model performs similarly to a Depth 10 tree since it has almost the same results and avoids overfitting. All in all, the Decision Tree model with depth 10 is the best-balanced model out of the ones that were tested since it provides a solid performance on training and testing the data as well as not risking overfitting as seen in deep trees (Depth 50).

3) Multi-Layer Perceptron (MLP):

Depth	Hidden Units	Test Accuracy	Training Time (s)
1	128	0.8210	2.25434
1	256	0.8280	2.22385
1	512	0.8390	2.53622
2	128	0.7990	3.1164
2	256	0.8110	3.03951
2	512	0.7940	3.5444
3	128	0.7990	3.39777
3	256	0.8110	3.89097
3	512	0.8030	3.92762

Table 3: Test Accuracy for MLP

Analyzing the results that were obtained when running the Multi-Layer Perceptron (MLP), the goal was to understand how different hyperparameters influence the accuracy of the model. When comparing the test accuracy to the number of hidden units in each depth, we can observe a clear improvement in the accuracy of the model when increasing the number of hidden units. This improvement ensures that the model can make more precise predictions for complex data such as the CIFAR-10 dataset. At depth 2 and 3, we an see similar results were the model increases the performance from 128 to 256 hidden units then drops back from 256 to 512 hidden units. This drop may lead into overfitting were the model fails to generalize the test data provided by the dataset.

Moreover, taking the training time into account, we can observe a slight increase in the training time of the model when the number of hidden units increases. This reflects on the additional complexity that the model must go through when training on a larger network. Also, increasing the depth of the MLP model has slightly dropped the accuracy results and increases the training time of the model itself. If we increase the depth of the model even higher, we will observe a slight drop in test accuracy and will observe a higher training time for the model since it will have a larger network to go through.

All in all, designing the model at depth 1 with 512 hidden units is the model that provided the highest accuracy results of approximately 83.90%.

4) Convolutional Neural Network:

Kernel Size	Test Accuracy
3	0.6360
5	0.5740
7	0.5560

Table 4: Test Accuracy for CNN

The last model that was implemented is the Convolutional Neural Network (CNN). The results obtained are based on varying the kernel size of the designed model. Based on our results, we can observe that the smaller kernel (3x3) will result in a higher test accuracy. We can also observe that the test accuracy drops when the kernel size increases to (5x5) and (7x7).

The smaller kernel size is more effective in the case of the Convolutional Neural Network (CNN) since it is more effective focusing on the local features for recognizing images in the CIFAR-10 dataset. The smaller the kernel size the better the model is at generalizing the training data. Using a larger kernel size, it is less effective in the case of the CNN model since it increases the chances of overfitting the data.

Moreover, for the depth variation, it is the same as the decision tree model. The smaller the depth of the model, the less the model can correctly recognize the data since the model is less effective for complex data. When implementing the CNN model with moderate depth, it will improve the training and testing of the model resulting in better test accuracy. Finally, having a large depth, it can recognize more complex images which makes the CNN model having the best test accuracy. Compared to the decision tree, the higher the depth of the model results in a higher test accuracy.

In summary, the results obtained for each of the four implemented models average a test accuracy between 60 % and 80 %. 1) The GNB model results in a test accuracy of 79.5%, 2) The Decision Tree Model with Depth 10 resulted with a 61.6 % accuracy, 3) The Multi-Layer Perceptron with Depth 1 and 512 hidden layers resulted with 83.9% and finally 4) The CNN model with kernel size 3x3 resulted with a test accuracy of 63.6 %.

Implemented Model	**Test Accuracy (%)**
Custom GNB	79.5 %
Scikit's Learn GNB	79.5 %
Custom Decision Tree with depth 10	61.6 %
Scikit's Learn Decision Tree	60.6 %
Multi-Layer Perceptron with depth 1 and 512 hidden units	83.9 %
Convolutional Neural Network with Kernel size 3x3	63.6 %

Table 5: Summary of the Obtained Results

References:

- [1] PyTorch, "CIFAR10", [Online]. Available: https://pytorch.org/vision/0.19/generated/torchvision.datasets.CIFAR10.html, [Accessed on November 14th, 2024].

- [2] PyTorch, "Transfer Learning for Computer Vision Tutorial", [Online]. Available: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html, [Accessed on November 14th, 2024].

- [3] PyTorch, "Quickstart Tutorial", [Online]. Available: https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html, [Accessed on November 14th, 2024].

- [4] Scikit Learn, "PCA", [Online]. Available: https://scikit-learn.org/1.5/auto_examples/decomposition/plot_pca_iris.html, [Accessed on November 14th , 2024].

Appendix A:

- **Classification Report Custom GNB:**

Number of Classes	Precision	Recall	F1-Score	Support
0	0.75	0.81	0.79	100
1	0.94	0.89	0.91	100
2	0.78	0.62	0.69	100
3	0.64	0.76	0.70	100
4	0.73	0.74	0.73	100
5	0.78	0.76	0.77	100
6	0.78	0.81	0.79	100
7	0.87	0.80	0.83	100
8	0.84	0.88	0.86	100
9	0.88	0.88	0.88	100
Accuracy			0.80	1000
Macro Average	0.80	0.80	0.80	1000
Weighted Average	0.80	0.80	0.80	1000

Table 6: Classification Report Custom GNB

- **Classification Report Scikit's Learn GNB:**

Number of Classes	Precision	Recall	F1-Score	Support
0	0.75	0.81	0.79	100
1	0.94	0.89	0.91	100
2	0.78	0.62	0.69	100
3	0.64	0.76	0.70	100
4	0.73	0.74	0.73	100
5	0.78	0.76	0.77	100
6	0.78	0.81	0.79	100
7	0.87	0.80	0.83	100
8	0.84	0.88	0.86	100
9	0.88	0.88	0.88	100
Accuracy			0.80	1000
Macro Average	0.80	0.80	0.80	1000
Weighted Average	0.80	0.80	0.80	1000

Table 7: Classification Report Scikit's Learn GNB

- **Classification Report for Decision Tree with Depth 1:**

Number of Classes	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	100
1	0.00	0.00	0.00	100
2	0.00	0.00	0.00	100
3	0.00	0.00	0.00	100
4	0.00	0.00	0.00	100
5	0.00	0.00	0.00	100
6	0.15	0.99	0.27	100
7	0.00	0.00	0.00	100
8	0.00	0.00	0.00	100
9	0.28	0.98	0.43	100
Accuracy			0.20	1000
Macro Average	0.04	0.20	0.07	1000
Weighted Average	0.04	0.20	0.07	1000

Table 8: Classification Report Decision Tree with Depth 1

- **Classification Report for Decision Tree with Depth 5:**

Number of Classes	Precision	Recall	F1-Score	Support
0	0.46	0.66	0.54	100
1	0.69	0.66	0.68	100
2	0.61	0.22	0.32	100
3	0.36	0.51	0.42	100
4	0.64	0.49	0.55	100
5	0.48	0.39	0.43	100
6	0.60	0.79	0.69	100
7	0.52	0.49	0.50	100
8	0.61	0.57	0.59	100
9	0.65	0.68	0.67	
Accuracy			0.55	1000
Macro Average	0.56	0.55	0.54	1000
Weighted Average	0.56	0.55	0.54	1000

Table 9: Classification Report Decision Tree with Depth 5

- **Classification Report for Decision Tree with Depth 10:**

Number of Classes	Precision	Recall	F1-Score	Support
0	0.59	0.61	0.60	100
1	0.83	0.73	0.78	100
2	0.53	0.40	0.45	100
3	0.43	0,62	0.51	100
4	0.60	0.59	0.59	100
5	0.56	0.58	0.57	100
6	0.73	0.69	0.71	100
7	0.59	0.52	0.55	100
8	0.68	0.66	0.67	100
9	0.72	0.76	0.74	100
Accuracy			0.62	1000
Macro Average	0.63	0.62	0.62	1000
Weighted Average	0.63	0.62	0.62	1000

Table 10: Classification Report Decision Tree with Depth 10

- **Classification Report for Decision Tree with Depth 50:**

Number of Classes	Precision	Recall	F1-Score	Support
0	0.55	0.59	0.56	100
1	0.83	0.70	0.76	100
2	0.47	0.42	0.44	100
3	0.45	0.52	0.48	100
4	0.52	0.49	0.50	100
5	0.53	0.57	0.55	100
6	0.65	0.71	0.68	100
7	0.58	0.51	0.54	100
8	0.63	0.60	0.62	100
9	0.68	0.76	0.72	100
Accuracy			0.59	1000
Macro Average	0.59	0.59	0.59	1000
Weighted Average	0.59	0.59	0.59	1000

Table 11: Classification Report Decision Tree with Depth 50

- **Multi-Layer Perceptron:**

Epoch	Loss	Epoch	Loss
1	57.1851	11	19.2278
2	36.7342	12	11.8553
3	33.4139	13	8.2808
4	28.3038	14	19.8992
5	22.6891	15	9.4690
6	19.4260	16	7.6380
7	12.8859	17	4.4722
8	10.6764	18	3.6693
9	11.2649	19	7.8452
10	17.8570	20	14.3210

Table 12: Loss Calculation for MLP

Epoch	Loss	Epoch	Loss
1	60.3354	11	12.8641
2	33.8444	12	12.2545
3	31.2743	13	11.4561
4	28.2384	14	8.8897
5	24.9623	15	7.6053
6	21.8704	16	7.4844
7	20.2743	17	8.6729
8	17.7590	18	5.3955
9	16.1540	19	4.6266
10	14.2372	20	3.6564

Table 13: Loss Calculation for 1 Hidden Layer and 128 Hidden Units

Epoch	Loss	Epoch	Loss
1	57.8139	11	9.7527
2	32.7109	12	7.3516
3	28.2492	13	5.8034
4	25.2789	14	4.7318
5	21.2281	15	3.8792
6	18.6016	16	3.1496
7	16.4958	17	2.6413
8	13.7109	18	2.1577
9	12.8224	19	1.9098
10	10.9311	20	1.6971

Table 14: Loss Calculation for 1 Hidden Layer and 256 Hidden Units

Epoch	Loss	Epoch	Loss
1	53.6458	11	5.1423
2	34.1398	12	3.9366
3	29.2757	13	3.2469
4	24.6311	14	2.476
5	20.5064	15	2.0015
6	15.9122	16	1.6949
7	14.4336	17	1.4512
8	11.1911	18	1.278
9	9.2269	19	1.1892
10	6.7338	20	1.1076

Table 15: Loss Calculation for 1 Hidden Layer and 512 Hidden Units

Epoch	Loss	Epoch	Loss
1	57.4553	11	15.0961
2	35.6147	12	23.6033
3	31.1057	13	17.9936
4	27.8902	14	12.7061
5	25.1988	15	11.7524
6	20.1059	16	8.96
7	20.3158	17	8.6761
8	18.105	18	5.9556
9	20.7131	19	5.1147
10	18.3946	20	4.651

Table 16: Loss Calculation for 2 Hidden Layers and 128 Hidden Units

Epoch	Loss	Epoch	Loss
1	57.3557	11	9.8244
2	38.5153	12	11.6994
3	29.6481	13	8.0041
4	24.131	14	6.3764
5	23.9673	15	7.6326
6	20.9519	16	3.6965
7	14.1443	17	2.608
8	12.0018	18	2.0357
9	13.1659	19	2.1543
10	11.1215	20	4.034

Table 17: Loss Calculation for 2 Hidden Layers and 256 Hidden Units

Epoch	Loss	Epoch	Loss
1	58.0841	11	13.8485
2	38.6866	12	11.5496
3	33.1003	13	8.0945
4	20.9186	14	4.6501
5	18.0277	15	3.1279
6	17.805	16	4.0052
7	15.7927	17	4.8462
8	15.3276	18	12.2869
9	11.6964	19	8.7547
10	18.2679	20	15.3599

Table 18: Loss Calculation for 2 Hidden Layers and 512 Hidden Units

Epoch	Loss	Epoch	Loss
1	62.2995	11	16.76
2	39.1964	12	15.7808
3	32.0462	13	11.0294
4	28.4599	14	12.7187
5	24.7562	15	16.1432
6	22.4377	16	12.8695
7	23.8889	17	11.669
8	22.8312	18	9.0324
9	16.5383	19	11.5826
10	17.8666	20	7.6706

Table 19: Loss Calculation for 3 Hidden Layers and 128 Hidden Units

Epoch	Loss	Epoch	Loss
1	59.6742	11	11.2199
2	39.9696	12	6.7115
3	30.708	13	9.2321
4	28.1458	14	7.5849
5	21.2246	15	6.5238
6	23.1632	16	9.3228
7	17.0815	17	5.8854
8	13.9453	18	3.2226
9	17.9299	19	4.0895
10	16.6317	20	5.6236

Table 20: Loss Calculation for 3 Hidden Layers and 256 Hidden Units

Epoch	Loss	Epoch	Loss
1	65.6835	11	7.3652
2	42.1026	12	6.4609
3	35.574	13	9.0608
4	25.8705	14	9.4828
5	24.4731	15	9.4741
6	22.0784	16	15.3318
7	17.1121	17	11.9063
8	20.841	18	9.8695
9	19.8136	19	5.5591
10	11.5051	20	8.7073

Table 21: Loss Calculation for 3 Hidden Layers and 512 Hidden Units

Epoch	Loss	Accuracy	Epoch	Loss	Accuracy
1	1.9182	28.04%	11	0.4837	84.22%
2	1.5309	43.64%	12	0.3822	87.20%
3	1.3706	50.26%	13	0.3557	89.10%
4	1.1950	57.34%	14	0.2785	91.04%
5	1.0461	63.10%	15	0.2070	93.08%
6	0.8890	68.94%	16	0.1130	96.48%
7	0.8072	71.88%	17	0.1346	95.50%
8	0.6457	78.48%	18	0.1096	96.54%
9	0.5233	82.54%	19	0.2695	92.24%
10	0.4794	84.34%	20	0.2530	92.10%

Table 22: Loss and Accuracy Calculation for Kernel Size 3

Epoch	Loss	Accuracy	Epoch	Loss	Accuracy
1	2.0290	21.90 %	11	0.7290	75.50 %
2	1.8007	29.90 %	12	0.6215	78.66 %
3	1.6036	38.22 %	13	0.5746	80.64 %
4	1.4922	43.28 %	14	0.4694	84.18%
5	1.3646	49.22 %	15	0.4156	85.98 %
6	1.2355	55.04 %	16	0.3065	89.12%
7	1.1444	58.16 %	17	0.4150	86.32 %
8	1.0296	63.28 %	18	0.2472	91.90 %
9	0.9272	67.30 %	19	0.2986	90.28 %
10	0.8484	69.90 %	20	0.3139	89.66 %

Table 23: Loss and Accuracy Calculation for Kernel Size 5

Epoch	Loss	Accuracy	Epoch	Loss	Accuracy
1	2.0507	21.04%	11	1.0569	60.08%
2	1.8346	27.02%	12	0.9413	65.36%
3	1.7331	30.78%	13	0.9054	66.76%
4	1.6291	34.84%	14	0.7907	72.68%
5	1.5912	36.94%	15	0.7726	72.34%
6	1.4522	41.80%	16	0.6600	75.92%
7	1.4060	46.22%	17	0.5931	78.48%
8	1.3159	49.40%	18	0.5516	79.86%
9	1.2728	52.14%	19	0.5302	81.62%
10	1.1293	57.88%	20	0.4370	84.70%

Table 24: Loss and Accuracy Calculation for Kernel Size 7

Appendix B:

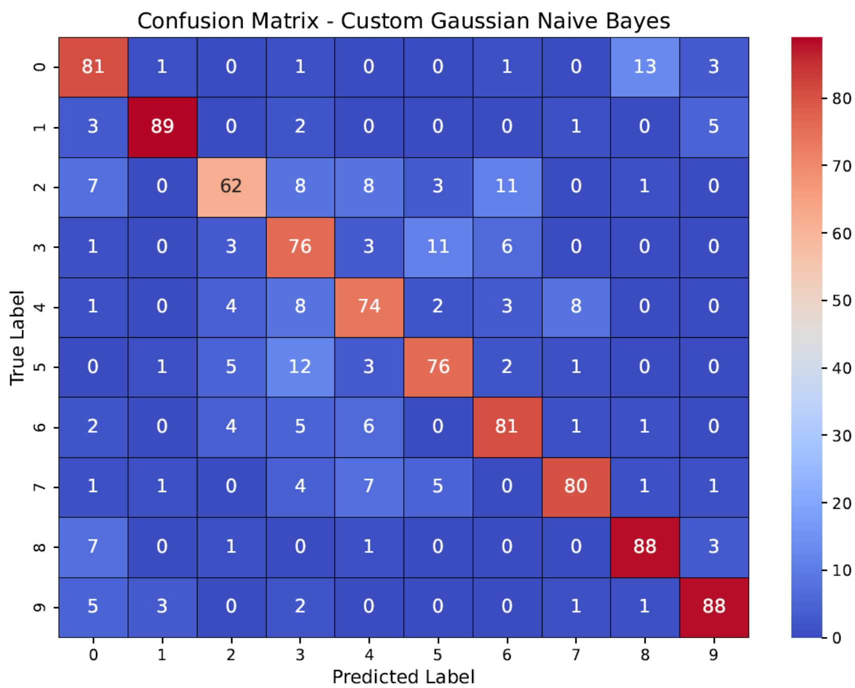

Figure 1: Confusion Matrix Custom GNB

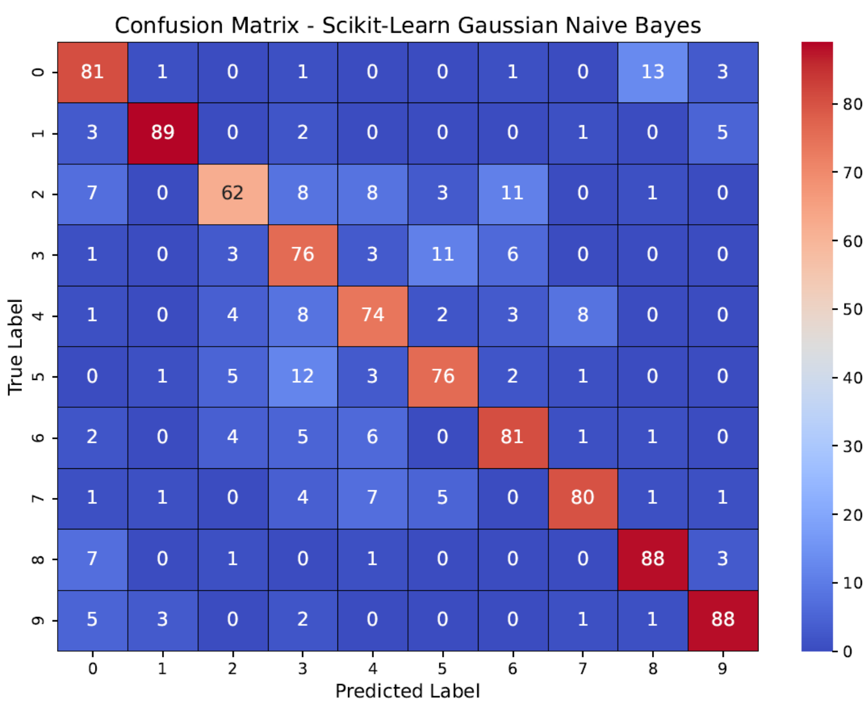

Figure 2: Confusion Matrix Scikit's Learn GNB

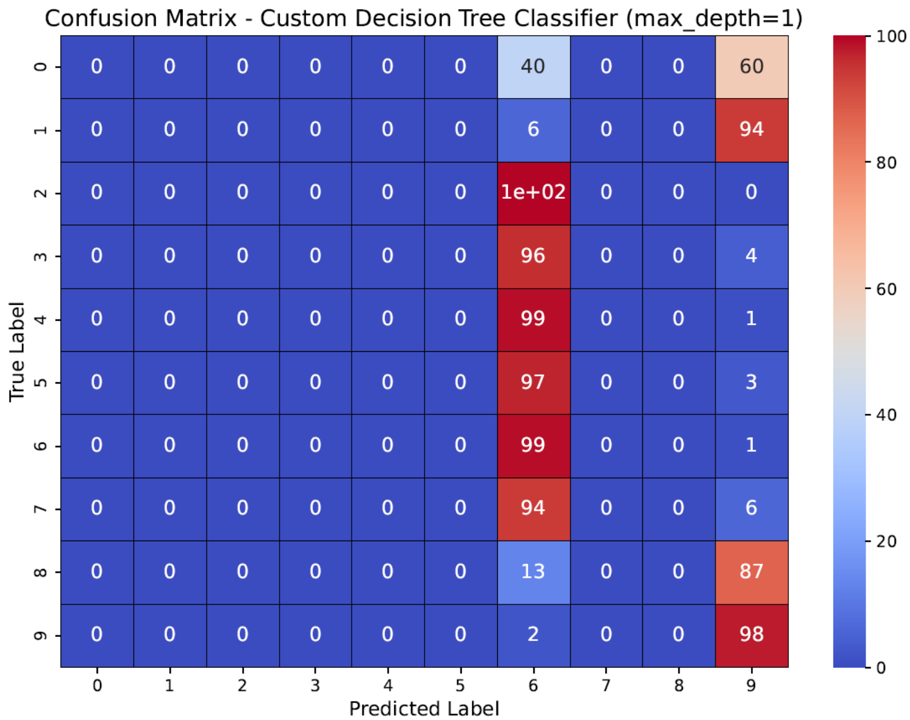

Figure 3: Confusion Matrix Decision Tree with Depth 1

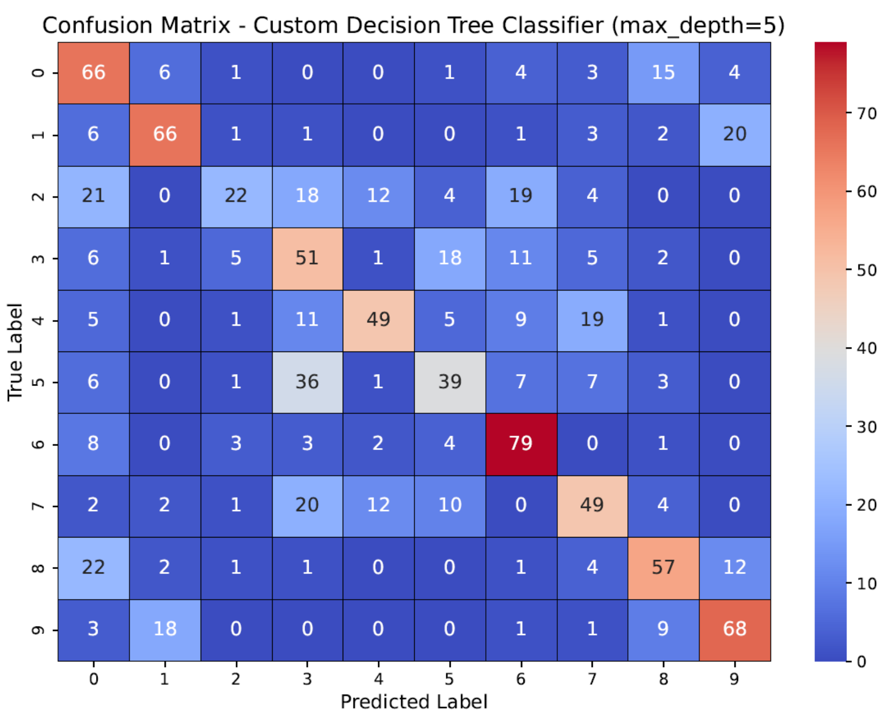

Figure 4: Confusion Matrix Decision Tree with Depth 5

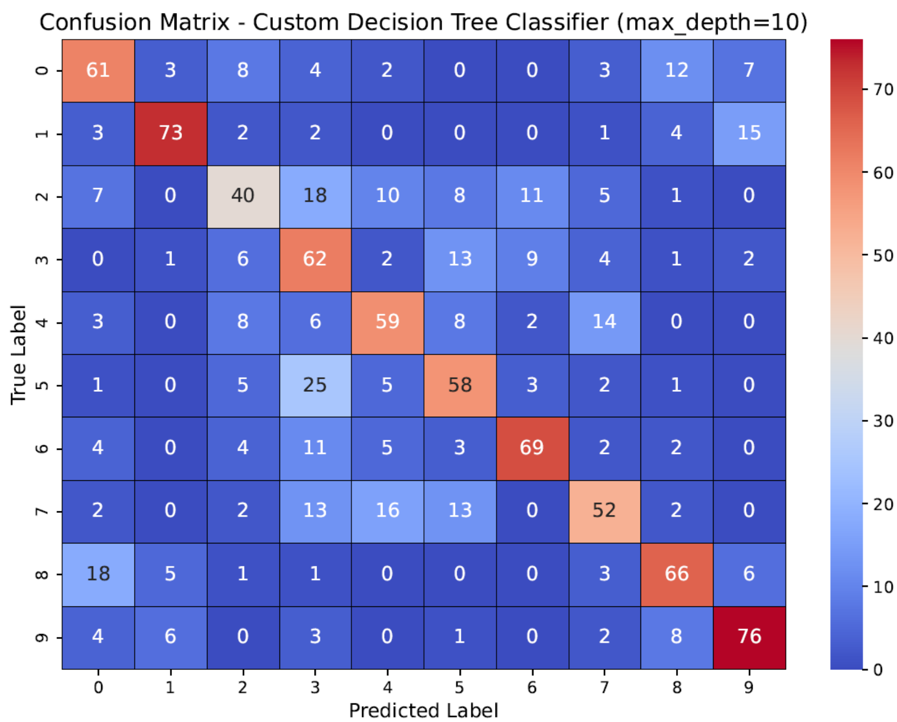

Figure 5: Confusion Matrix Decision Tree with Depth 10

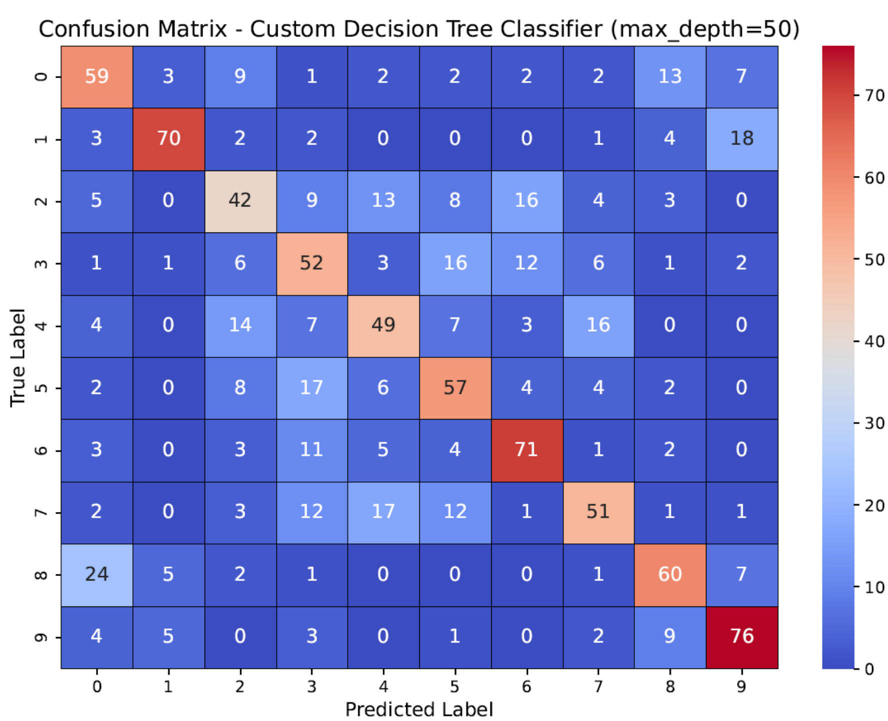

Figure 6: Confusion Matrix Decision Tree with Depth 50

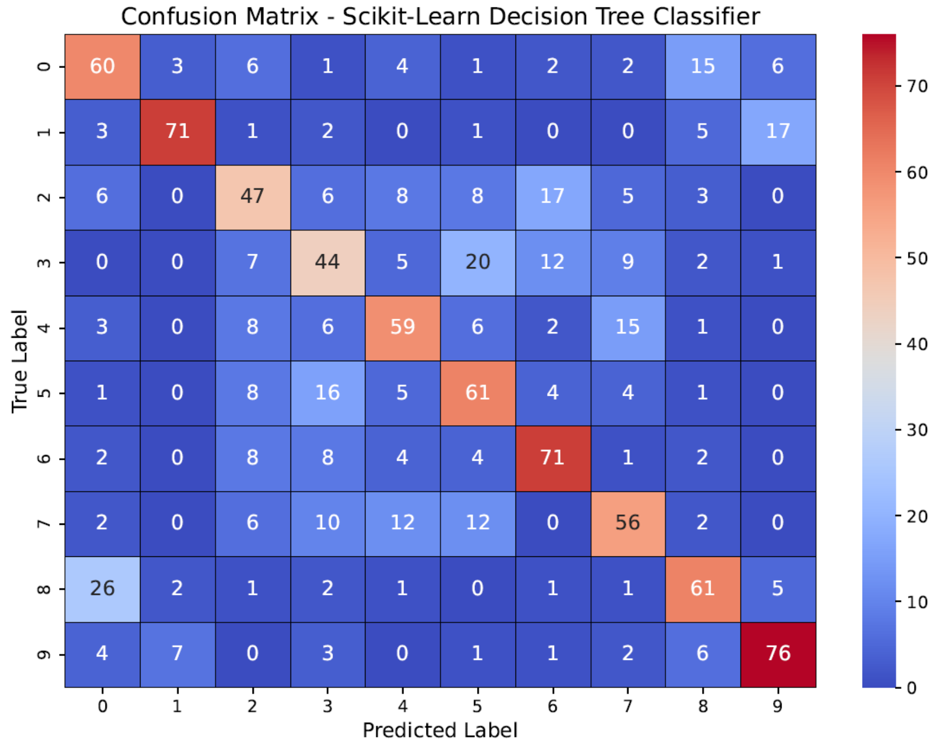

Figure 7: Confusion Matrix Scikit's Learn Decision Tree

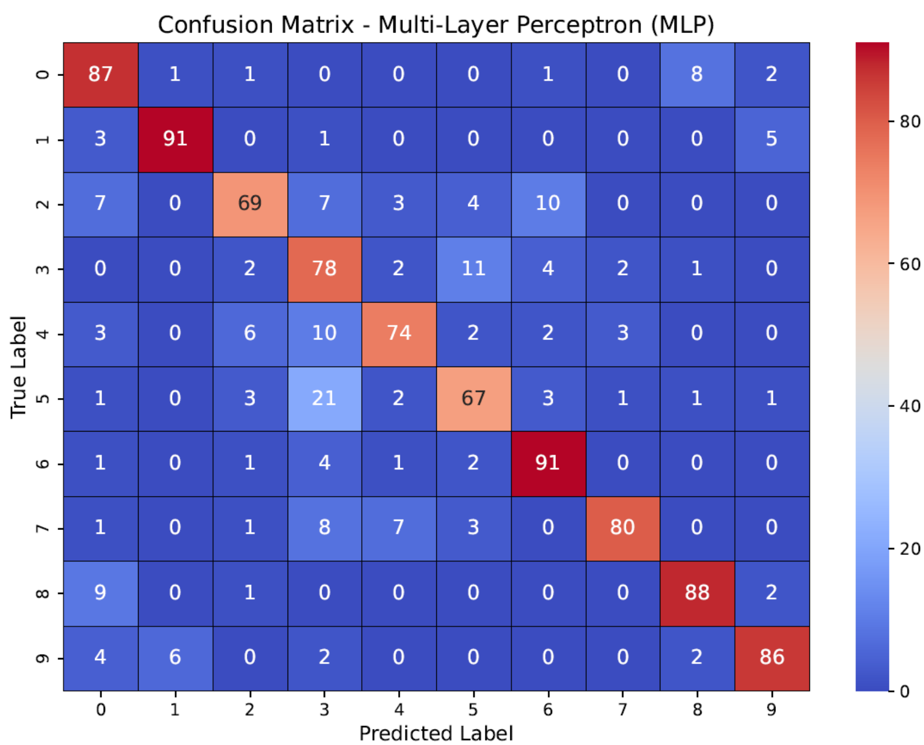

Figure 8: Confusion Matrix Multi-Layer Perceptron (MLP)

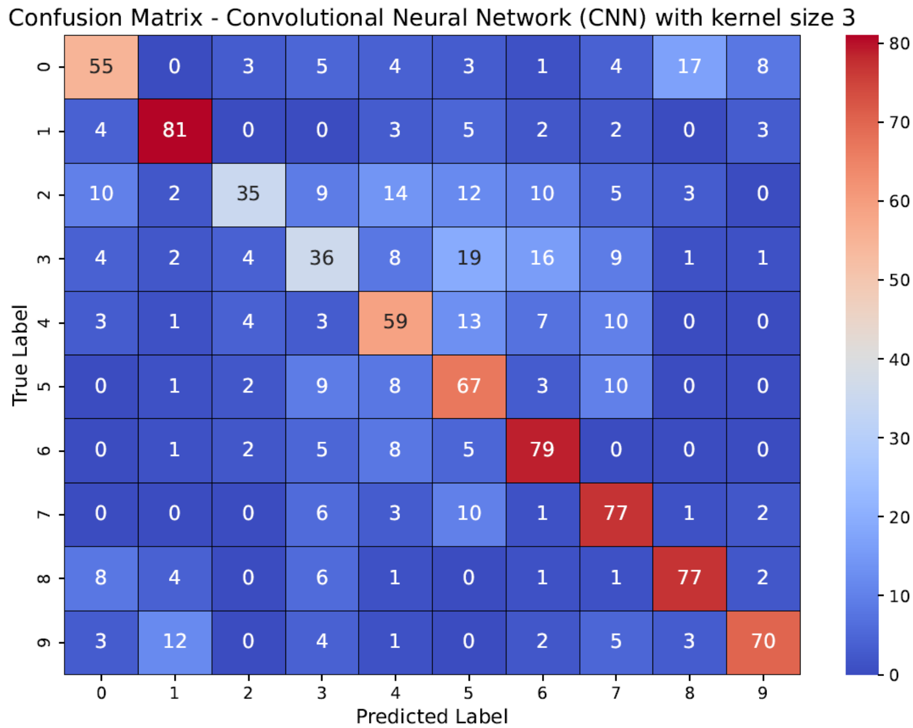

Figure 9: Confusion Matrix CNN with Kernel Size 3

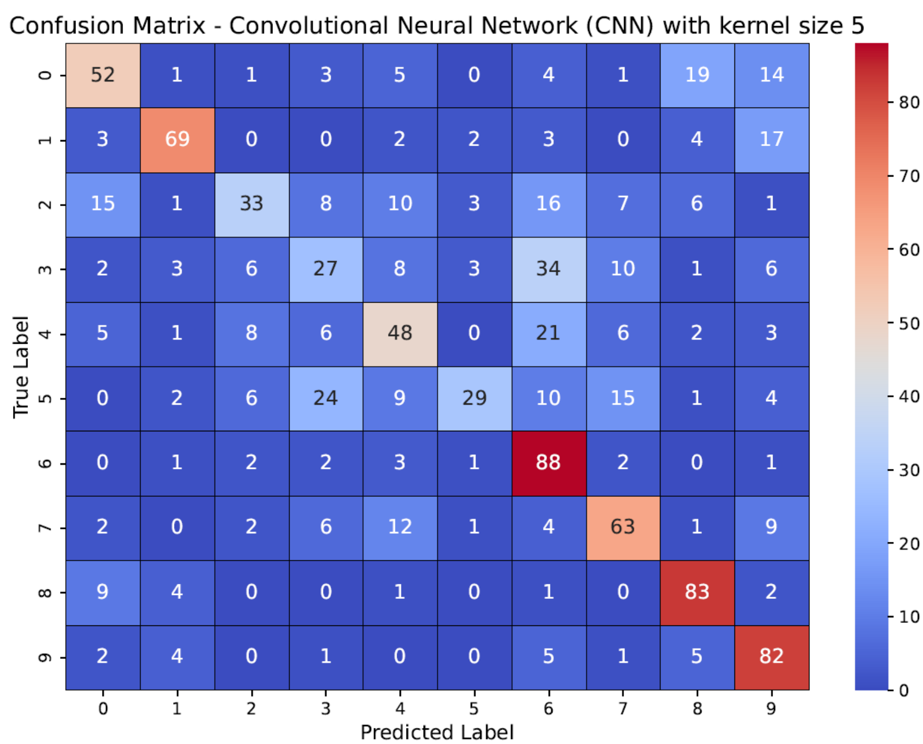

Figure 10: Confusion Matrix CNN with Kernel Size 5

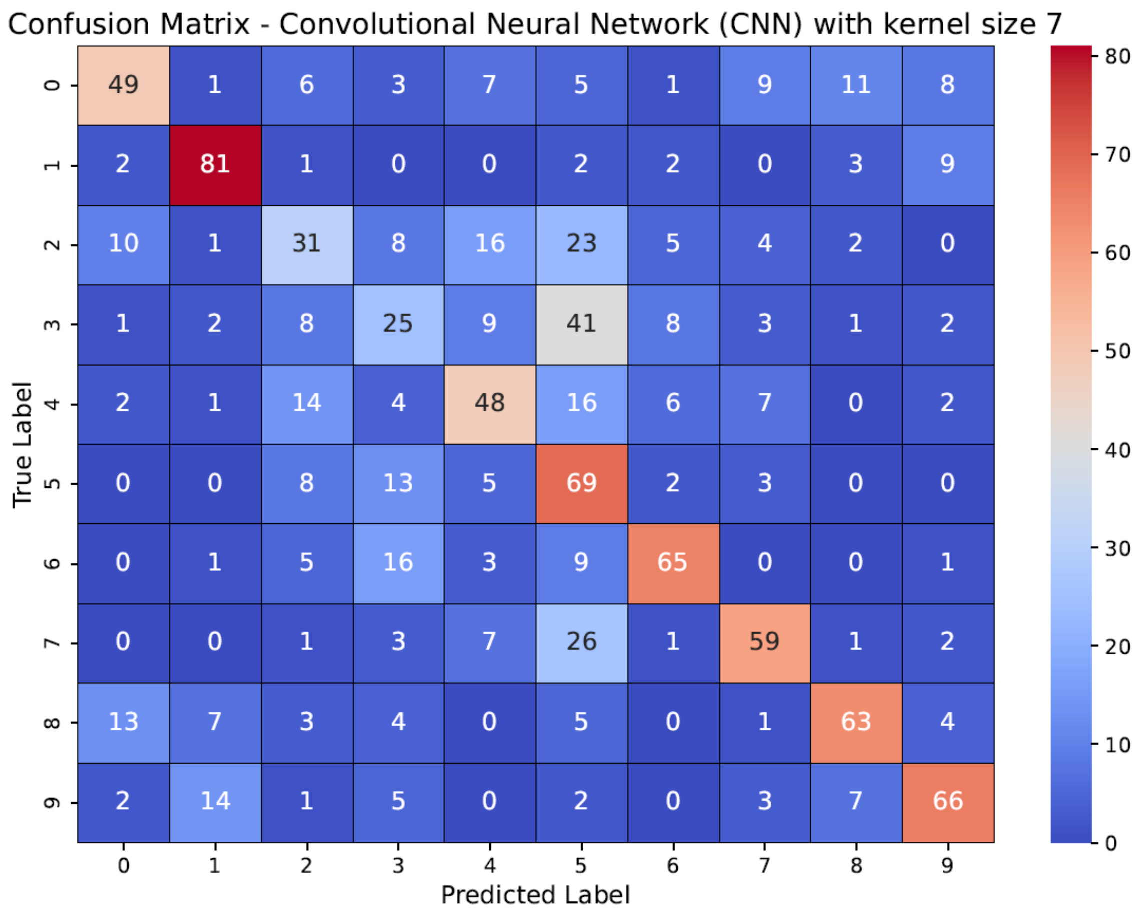

Figure 11: Confusion Matrix CNN with Kernel Size 7