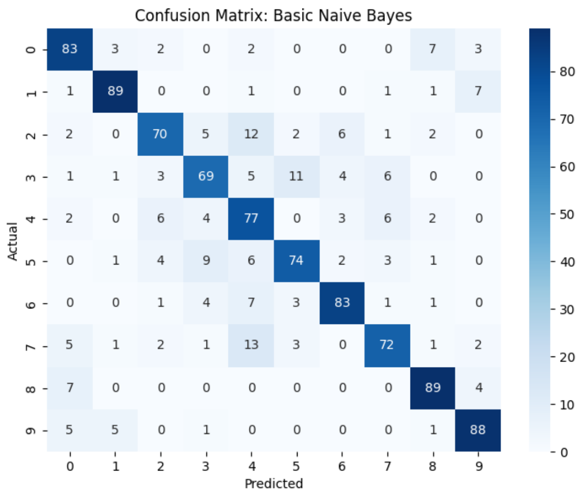
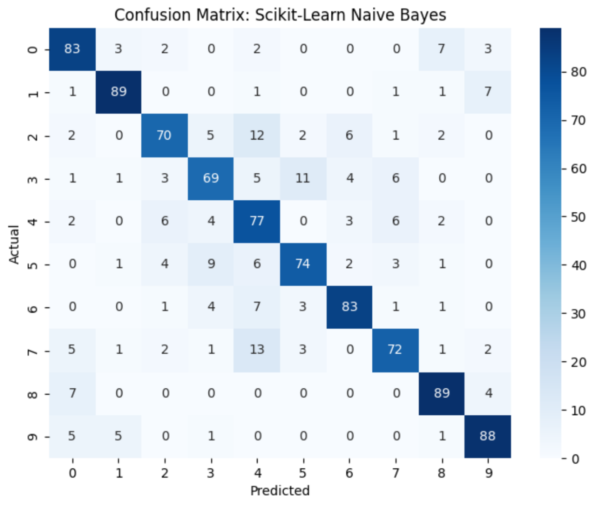
Image classification report – Emily Porter 40204837

Naïve Bayes



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	83	3	2	0	2	0	0	0	7	3
1	1	89	0	0	1	0	0	1	1	7
2	2	0	70	5	12	2	6	1	2	0
3	1	1	3	69	5	11	4	6	0	0
4	2	0	6	4	77	0	3	6	2	0
5	0	1	4	9	6	74	2	3	1	0
6	0	0	1	4	7	3	83	1	1	0
7	5	1	2	1	13	3	0	72	1	2
8	7	0	0	0	0	0	0	0	89	4
9	5	5	0	1	0	0	0	0	1	88



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	83	3	2	0	2	0	0	0	7	3
1	1	89	0	0	1	0	0	1	1	7
2	2	0	70	5	12	2	6	1	2	0
3	1	1	3	69	5	11	4	6	0	0
4	2	0	6	4	77	0	3	6	2	0
5	0	1	4	9	6	74	2	3	1	0
6	0	0	1	4	7	3	83	1	1	0
7	5	1	2	1	13	3	0	72	1	2
8	7	0	0	0	0	0	0	0	89	4
9	5	5	0	1	0	0	0	0	1	88

Basic Naive Bayes Metrics of Evaluation

Accuracy: 0.7940

Precision: 0.7973

Recall: 0.7940

F1-Score: 0.7940

Scikit-Learn Naive Bayes Metrics of Evaluation:

Accuracy: 0.7940

Precision: 0.7973

Recall: 0.7940

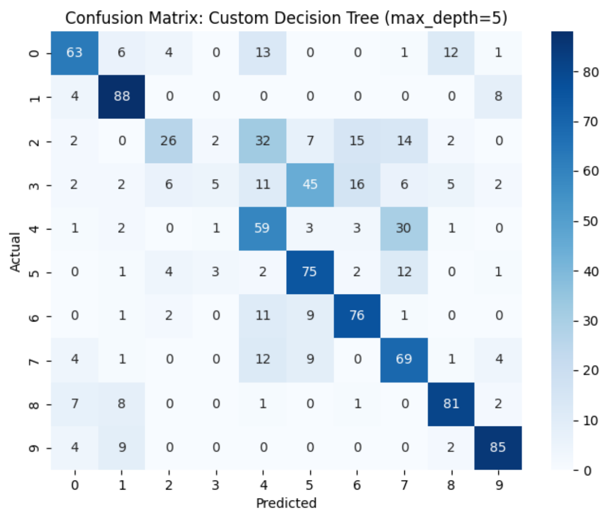
F1-Score: 0.7940

There are 2 variants for the Naïve Bayes algorithm, the Gaussian Naive Bayes algorithm and the scikit-learn Gaussian Naïve Bayes classifier. The biggest difference between them Is the custom Gaussian Naïve Bayes needs to be built from scratch whereas the scikit-learn Naïve bayes is pre-built. The custom one could be more flexible and customizable, however it’s not as easy to implement as a pre-built method. They’re both based on the same mathematical logic therefore in terms of performance for this level of project, there isn’t a difference. For the training methodology of my basic Naïve Bayes algorithm, I made a predict method where it calculated the prior probability of each class and the likelihood of classifying the correct image given the class.

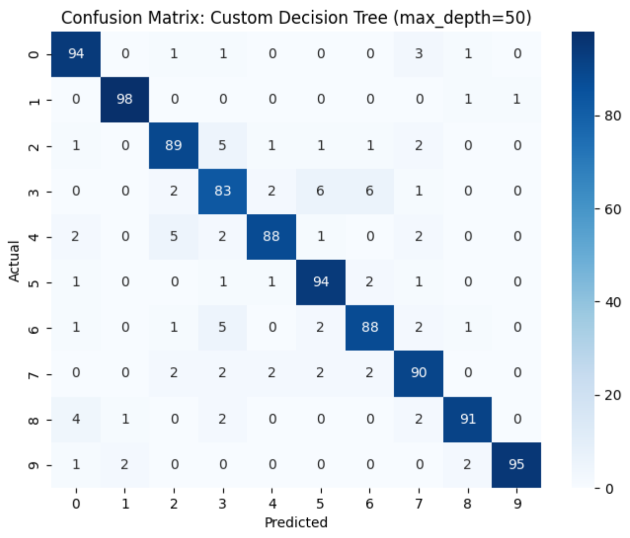
We can observe that the confusion matrix, as well as the metrics of evaluation are the same for both Naïve Bayes algorithms. The reason behind that will be as explained above, that the basic Naïve bayes algorithm I made follows the same mathematical logic as the scikit-learn classifier. Therefore, will result to the same.

For performance, classes 0, 1 and 9 (airplane, automobile and truck) did very well with 83, 89 and 88 correct guesses. This can be explained because these 3 classes have very distinct and separable features. Naïve bayes look at features independently and do not make connections between them, therefore images with stronger ones are easier to classify correctly. This also explains why classes like bird and horse might have a lower correct classification rate. The imbalance of images fed to the naïve bayes can also affect the performance of classification. This algorithm relies heavily on the amount of training images you feed it, therefore there could have been a lack of bird and horse training images.

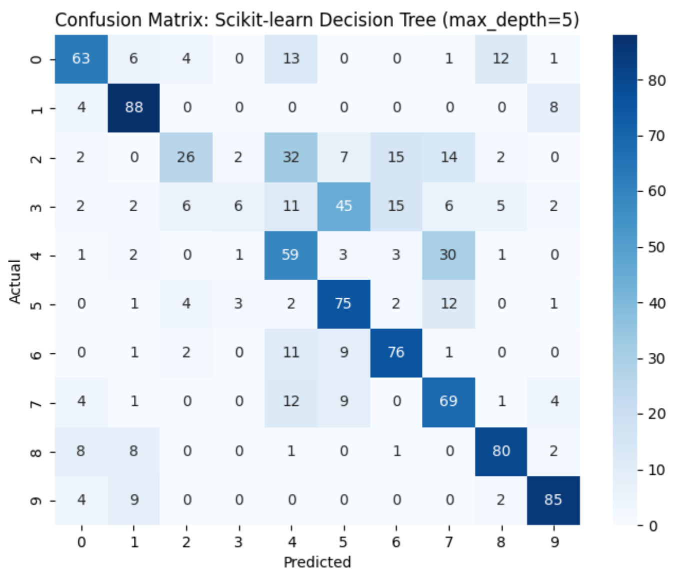
Decision tree

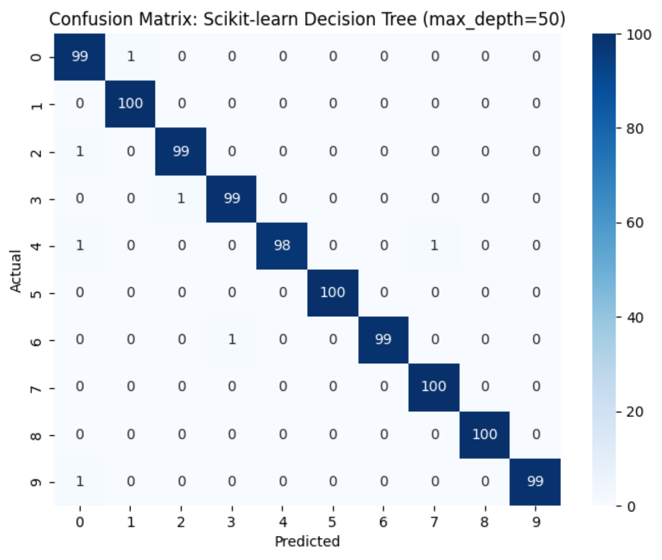


Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	63	6	4	0	13	0	0	1	12	1
1	4	88	0	0	0	0	0	0	0	8
2	2	0	26	2	32	7	15	14	2	0
3	2	2	6	5	11	45	16	6	5	2
4	1	2	0	1	59	3	3	30	1	0
5	0	1	4	3	2	75	2	12	0	1
6	0	1	2	0	11	9	76	1	0	0
7	4	1	0	0	12	9	0	69	1	4
8	7	8	0	0	1	0	1	0	81	2
9	4	9	0	0	0	0	0	0	2	85



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	94	0	1	1	0	0	0	3	1	0
1	0	98	0	0	0	0	0	0	1	1
2	1	0	89	5	1	1	1	2	0	0
3	0	0	2	83	2	6	6	1	0	0
4	2	0	5	2	88	1	0	2	0	0
5	1	0	0	1	1	94	2	1	0	0
6	1	0	1	5	0	2	88	2	1	0
7	0	0	2	2	2	2	2	90	0	0
8	4	1	0	2	0	0	0	2	91	0
9	1	2	0	0	0	0	0	0	2	95





Custom Decision Tree metrics of Evaluation (max depth 5)

Accuracy: 0.6270

Precision: 0.6264

Recall: 0.6270

F1 Score: 0.5969

Custom Decision Tree metrics of Evaluation (max depth 50):

Accuracy: 0.9100

Precision: 0.9109

Recall: 0.9100

F1 Score: 0.9101

Scikit-learn Decision Tree metrics of evaluation (max depth 5):

Accuracy: 0.6270

Precision: 0.6305

Recall: 0.6270

F1 Score: 0.5980

Scikit-learn Decision Tree metrics of evaluation (max depth 50):

Accuracy: 0.9930

Precision: 0.9931

Recall: 0.9930

F1 Score: 0.9930

I created 4 variants for the decision tree analysis. 2 basic decision trees and 2 were generated by the scikit's implementation. I varied the depth by 5 and 50 to see how it would perform for both the basic and scikit implementation. The major differences between them includes, ease of use, handling continuous versus categorical data and efficiency. The basic implementation uses the Gini coefficient to decide when to split, whereas scikit can also use entropy as a splitting factor. The custom tree also doesn't contain built in pruning so it could overfit if the tree depth is too high. Regarding handling continuous and categorical data, Scikit can handle both, whereas my basic one can only handle continuous data as it uses a binary split. Therefore if I had categorical data my code would not be able to handle more than 2 classes at a time for comparison. Finally for efficiency, the scikit-learn decision tree uses Cython code (which is a mix of both python and C) which fastens the computational speed, and helps to handle bigger amounts of data, whereas mine is fully coded in python forcingly making it slower to classify than the scikit-learn implementation.

The training methodology for my decision trees goes as follows. I created a function that calculates the Gini coefficient and a find_best_split function to make the decision when to split the data. To find the best split, the entire data set is at the root node and then we recursively split the data into subsets. The Gini index (entropy) is computed to measure when the best time is to split the data, the more you split the more minimized the Gini impurity is and the better it classifies. I also made a change_depth function that will go through 2 different depts when producing each model. I decided to only include 2 maximum depths (5 and 50) for simplicity for our project and also to have a drastic difference to observe.

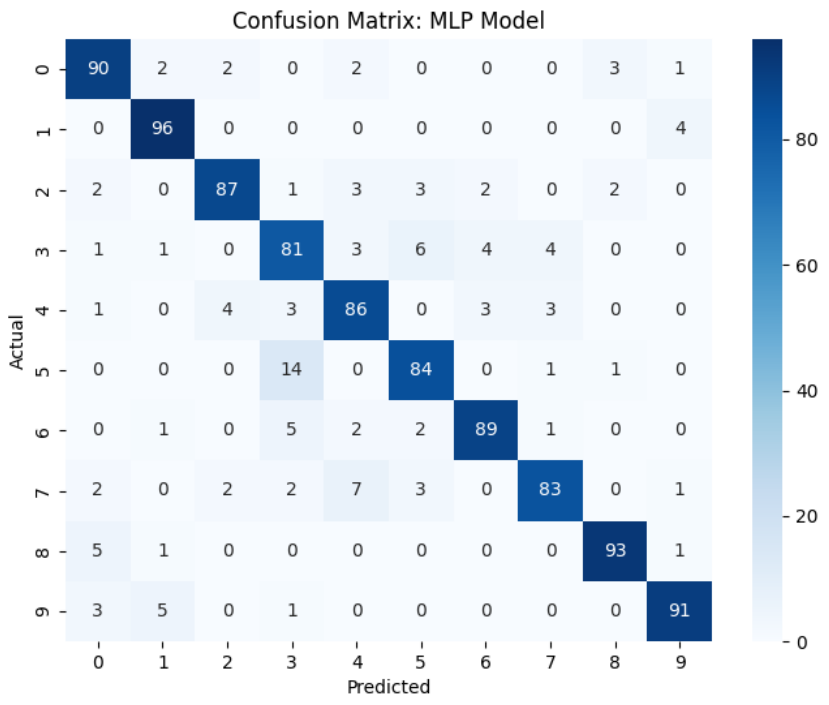
The difference between dept 5 and 50 is that there are only 5 splits done before reaching the leaf nodes and 50 you get to split more before reaching the leaf nodes which gives you better precision, however, might lead to over fitting which I will discuss further.

For the max depth of 5 decision trees, obviously there is way less precision and accuracy with a F1-score of around 0.59 for both the basic and scikit implementation. This is because we are underfitting, its not splitting enough times to catch more complicated patterns in the data and the model doesn't have enough depth to separate the features throughout all the classes which leads to misclassification.

Now, what we are saying with decision trees of max depth 50 is the opposite, where we almost have a perfect F1 score (0.99) for both. This is explained by have 55 more splits in the tree to analyse the feature and recognize the patterns. However, at this depth, over fitting might be one of the consequences. Overfitting refers to when the tree is too deep, and it starts to memorize the training images which misleads us to think that it is highly effective and accurate when its not.

I will use the confusion matrix of depth 5 to explain why certain classes were less properly classified as the depth 50 almost has perfect classification. We can observe that class 1 and 9 did well. Which are automobiles and trucks, and that's because they have distinct and non-overlapping features. Non overlapping features are important for decision tree splitting. Cars have a clear shape and the number of wheels therefore it is easier to identify. For classes that did less well like 2, 3 and 5 (birds, cats and dogs), these images have a lot of overlapping features, cats and birds are similar in color, cats and dogs have similar body shapes, textures, fur, etc. Therefore, it is hard to separate these features, especially in a shallow tree. Decision tree split based on the best feature at each node, so when 2 classes have overlapping features, it loses effectiveness to accurately separating the features.

Multi layered perceptron



```
Epoch [1/10], Loss: 1.4211
Epoch [2/10], Loss: 0.7363
Epoch [3/10], Loss: 0.5984
Epoch [4/10], Loss: 0.5304
Epoch [5/10], Loss: 0.4858
Epoch [6/10], Loss: 0.4535
Epoch [7/10], Loss: 0.4201
Epoch [8/10], Loss: 0.4059
Epoch [9/10], Loss: 0.3805
Epoch [10/10], Loss: 0.3623

MLP Model metrics of Evaluation:
Accuracy: 0.8800
Precision: 0.8814
Recall: 0.8800
F1 Score: 0.8803
```

I was able to implement the 3-layer multi-layer perceptron (MLP) with the correct architecture, however when attempting on varying the depth of the MLP and the sizes of the hidden layers it kept giving me errors and I could not make it run. Before analyzing the results of my working MLP, I want to go further into details onto potentially why my code wasn't running.

The goal of this section was to dynamically vary the depth and hidden layer sizes in the MLP, I tried to implement it based on my current MLP however I kept getting "Tensor dimension mismatch" error. This error might come up when layers are not matched correctly for forward passes. This can happen when the model isn't flexible enough and if the input-output sizes aren't updated properly it will result to mismatched tensor dimensions. Another error could be that I tried to implement a dictionary for layer configuration, but I think that the flow of data from one layer to another wasn't being done correctly.

Let's observe the results with the MLP we have working and discuss them. The classes that did well are 0, 1 and 9 (Automobiles, Trucks and Airplanes), the ones that did moderately well are classes 2 and 7 (birds and horses) and the ones that were less accurate 3, 6 and 5 (cat, dogs and deer). Note for class 6 (dog), there was 89 correct predictions, but it had many confusions between cats and horses.

Similarly to the other algorithms, the more distinct the features of the training images are, the easier the algorithm can separate them. It is the same for MLP’s. Cars and trucks have super distinct shapes, the wheels and the length, therefore it is easy to classify correctly. And same goes for classes with similar features, like how dog and cats look similar, or horses and deer also have overlapping features. But, in the CIFAR-10 dataset, certain classes have similar backgrounds or contexts that could confuse the model, an example can be that horses and deer are often in grassy fields, therefore the MLP just classifies both together. Also, the MLP relies on fully connected layers, which sees the patterns in the data, which is different then convolutional networks where it learns smaller patches of patterns (edges, textures, etc.). So, because MLP’s deal with pattern recognition differently than a CNN, it looks at the bigger picture rather than in smaller local spatial feature, it has a harder time recognising smaller details like the edge of a cat’s fur and confusing it with bird feathers.

I also printed the loss at every iteration (epoch) of the training dataset. For each epoch, the model performs a forward pass and a loss calculation. A forward pass is when the model makes a prediction based on the current parameters of the training sample. The loss is the difference between the predicted output and the real target.

Convolutional Neural Network
I unfortunately could not make my code run error free for the CNN, I tried debugging but I kept getting repeated errors like “RuntimeError: mat1 and mat2 shapes cannot be multiplied (64x2048 and 512x4096)” which is a result of dimension mismatch between matrices. This can be a result of incorrect flattening or output size of the layers are not correctly calculated.

The model is supposed to follow the VGG11 architecture which includes many convolutional layers followed by fully connected layers, but a common issue is managing the dimensions of the connected layers, which is why I think I kept getting the error. Therefore, I could not experiment with my CNN nor generate a confusion matrix or the metrics of evaluation attached to them.

Summary

I am aware that my summary will be missing a crucial overview of the CNN and the lack of experimentation with my MLP, however I will do the summary and rank the best out of what I have.

For Naïve bayes, it's fast and simple to implement, however it performs weak on image training due to looking at features independently and not making any links between them. The decision tree is great in terms of interpretability, it's easy for the user to understand how it's making it's decision for classification. It's also great to classify distinct image classes like cars. Weaknesses include overfitting and being weak when classes are too similar. Finally the 3 layer MLP had better accuracy on more complex patterns of the CIFAR-10 dataset. Due to it's fully connected layers, the MLP is better to classify more complicated links between features (cat vs dog). The MLP also does not assume features are independent like naïve bayes. It's weaknesses includes slower training time, and also if my CNN was implemented properly we can assume it would perform better because it is a deeper network.

If we were to classify the best performing algorithm, I would choose the MLP. Even though the metrics of evaluation of the MLP are slightly weaker than the decision tree at max depth of 50, there is way less of a chance of overfitting which is my theory on my decision tree at depth 50 performed almost perfectly.

In terms of accuracy the MLP has a score of 0.8800 which is high and reliable due to the higher complexity of the perceptron than the decision tree or naïve bayes.