Report COMP472: Artificial Intelligence

By Ties Schasfoort, SID: 40322820

Fall 2024

Content:

Model Architectures and Training:

Bayes:

I implemented the Gaussian Naive Bayes (GNB) algorithm from scratch using basic Python and Numpy, instead of using a library such as sklearn.naive_bayes.GaussianNB. There is only one main model since it is a probabilistic classifier and there are no variables that can be changed to change the performance of this classifier. Due to this fact, there is also no training needed, it's mere probability. Thus, there is no iterative training unlike some other models. This fact also leads to no need for optimization algorithms.

Decision Tree:

For the decision tree #1 I have made 3 variants to the main model. The main model had a depth of 50, for the other 3 I changed the depth to 10, 20, and 100. Thus, the following line changed:

```
dt = DecisionTreeClassifier(max_depth = 50)
```

For training this model does not involve a traditional iterative training process like neural networks. Instead, the training involves recursively splitting the dataset based on the Gini impurity criterion to build the decision tree.

No optimization algorithms or techniques like mini-batch gradient descent or Adam optimizer are used since the model doesn't involve training.

MLP:

For the MLP I implemented 7 models. The main one involves all 3 layers with hidden layers of size 512. There are three other models: One where the second layer is removed, one where the third layer is removed, and one where both the second and third layers are removed. The other 3 are: models where all layers are used but with size 128, 256, and 1024.

Training methodology:

1. Number of epochs = 20, meaning the model iterates over the entire training dataset 20 times to learn patterns effectively.
2. Batch size = 64, meaning the training data is processed in batches of 64 samples. This is to increase computational efficiency by reducing memory overhead and allowing faster computation of gradients while maintaining sufficient gradient stability.
3. Optimizer = SGD, with learning rate 0.01 which controls the step size for updating model parameters. And also, with momentum 0.9 which helps accelerate convergence and smooth out oscillations by using past gradients.
4. Loss function = CrossEntropyLoss, which computes the softmax of the model's raw output logits and evaluates the negative log-likelihood of the true class labels.

Techniques/algorithms used:

1. Mini-Batch Gradient Descent = the training data is processed in batches of 64 samples, which is a form of mini-batch gradient descent. This helps balance computational efficiency and model convergence by leveraging smaller, manageable subsets of data for gradient updates.
2. Momentum
3. Learning Rate

CNN:

For the CNN I used 5 different models. The first, main one has a kernel size of 3x3 and involves all the given layers. Then, there are two where I kept the same kernel size but removed the last 2 convolutional layers and all but the first one respectively. The other two models involve all layers again, but this time with a kernel size of 2x2 and 7x7 respectively.

When removing layers, the input to the fully-connected layers had to change as well. When removing the last 2 convolutional layers the first argument of the first fully-connected layer changed using the calculate_features_size() method.

And when only having the first convolutional layer it changed to this:

```
self.classifier = nn.Sequential(
nn.Linear(self.fc_input_size, 10)
)
```

When changing the kernel sizes, I just changed the third argument in Conv2d() for all of the convolutional layers. When changing the kernel size to 7, the padding had to change to 3. This is because the formula for padding based on kernel size n is: (n - 1)/2, according to https://stats.stackexchange.com/questions/297678/how-to-calculate-optimal-zero-padding-for-convolutional-neural-networks

Training methodology:

1. Number of epochs = 5, I decreased the number of epochs for this model compared to the MLP since it takes much longer to run the CNN due to its many layers and different kernel sizes. For example, running all layers with kernel size 7 took 12 minutes.
2. Batch size = 64.
3. Optimizer = SGD, with learning rate 0.01 and momentum 0.9.
4. Loss function = CrossEntropyLoss.

Techniques/algorithms used:

1. Mini-Batch Gradient Descent
2. Momentum
3. Learning Rate

Evaluation:

Metrics for main models:

Metric (%)	Bayes #1	Bayes #2	Tree #1	Tree #2	MLP	CNN
Accuracy	12.8	12.8	7.1	8.2	12.4	42.1
Precision	12.01	12.01	7.85	9.34	13.32	50.41
Recall	12.8	12.8	7.1	8.2	12.4	42.1
F1	12.39	12.39	7.46	8.75	12.85	45.88

To explain the metrics better, I have assumed that the pictures are of faces and the model tries to match the correct face to the correct person/class. Even though the data in the CIFAR-10 dataset are of objects like planes, ships and trucks, just for this section I am using faces to explain the concepts clearer.

Accuracy:

This is the overall percentage of correct predictions made by the model.

CNN (42.1%) has the highest accuracy, suggesting that it performs better overall than other models in terms of correctly classifying facial images.

The Bayes #1 and Bayes #2 models show very low accuracy. This might be due to the limited capability of the Bayesian models to capture complex patterns in facial images. This capacity is limited since these models are not trained at all.

Tree #1 and Tree #2 also show low accuracy, with Tree #1 being the least effective (7.1%).

Precision:

A high precision means the model makes fewer false positive errors, which is important when the cost of false positives is high (for example, misclassifying a non-face image as a face).

The CNN (50.41%) achieves the highest precision, indicating that when it classifies a facial image as positive (a face), it is more likely to be correct compared to other models.

MLP (13.32%) and the Bayes models (12.01%) also perform reasonably well, but their precision is much lower than CNN's, indicating they generate more false positives.

Tree #1 (7.85%) and Tree #2 (9.34%) have lower precision, implying they are more prone to false positives.

Recall:

Measures how many actual positive instances (faces for example) are correctly identified by the model.

CNN has the highest percentage again, with a recall of 42.1%, suggesting it is better at identifying true faces.

The Bayes models (12.8%) show a higher recall than the Tree models (both 7.1% and 8.2%), meaning they are somewhat better at capturing facial images but still fall short compared to CNN.

Tree #1 and Tree #2 have low recall, meaning they miss a lot of faces.

F1:

This is metric is useful when dealing with imbalanced classes (e.g., when faces are rare compared to non-faces).

CNN (45.88%) has the highest F1 score, suggesting it has the best balance between precision and recall.

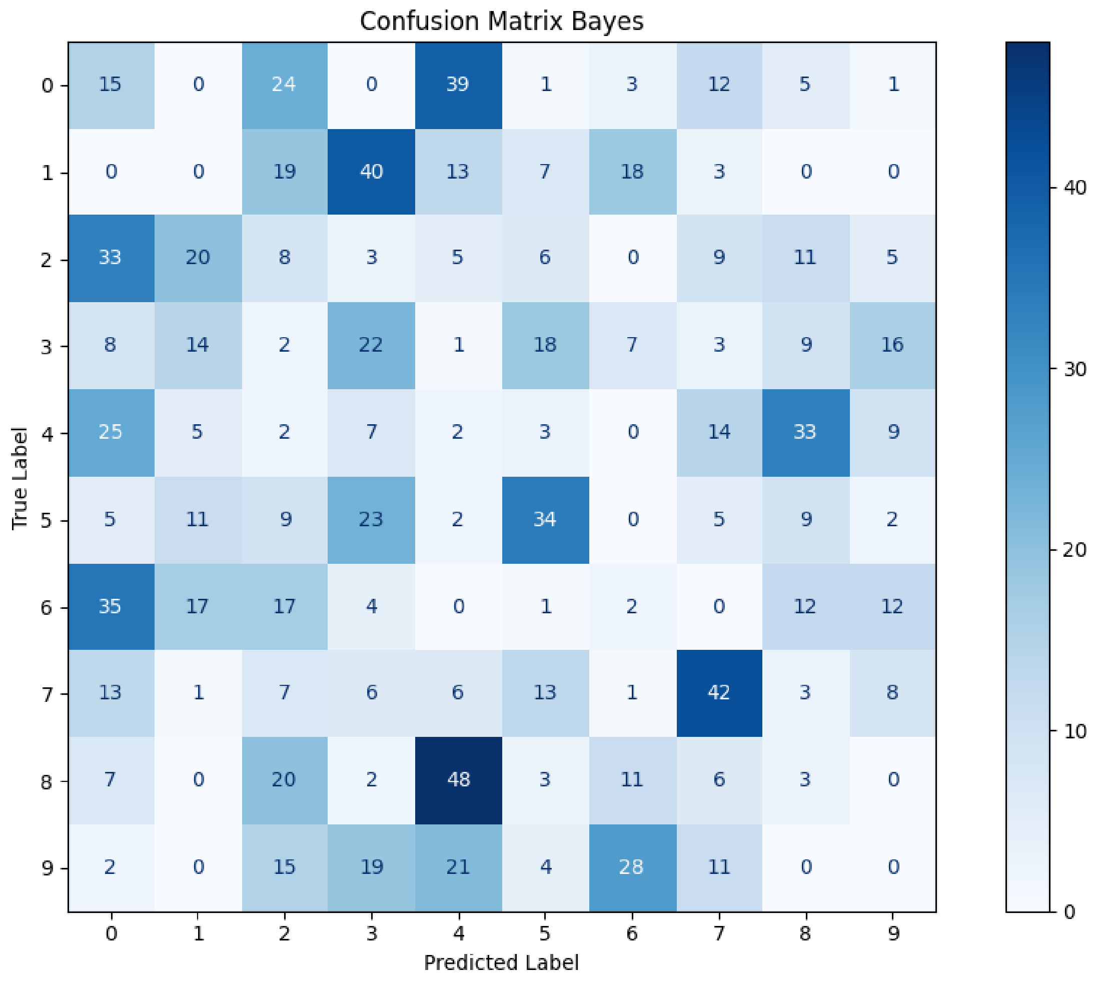
The MLP (12.85%) and Bayes #1 and #2 (12.39%) models have much lower F1 scores, indicating they are not effectively balancing precision and recall.

Tree #2 (8.75%) performs slightly better than Tree #1 (7.46%) in terms of F1, but both are still worse than CNN.

Now I will evaluate each model and its variants individually.

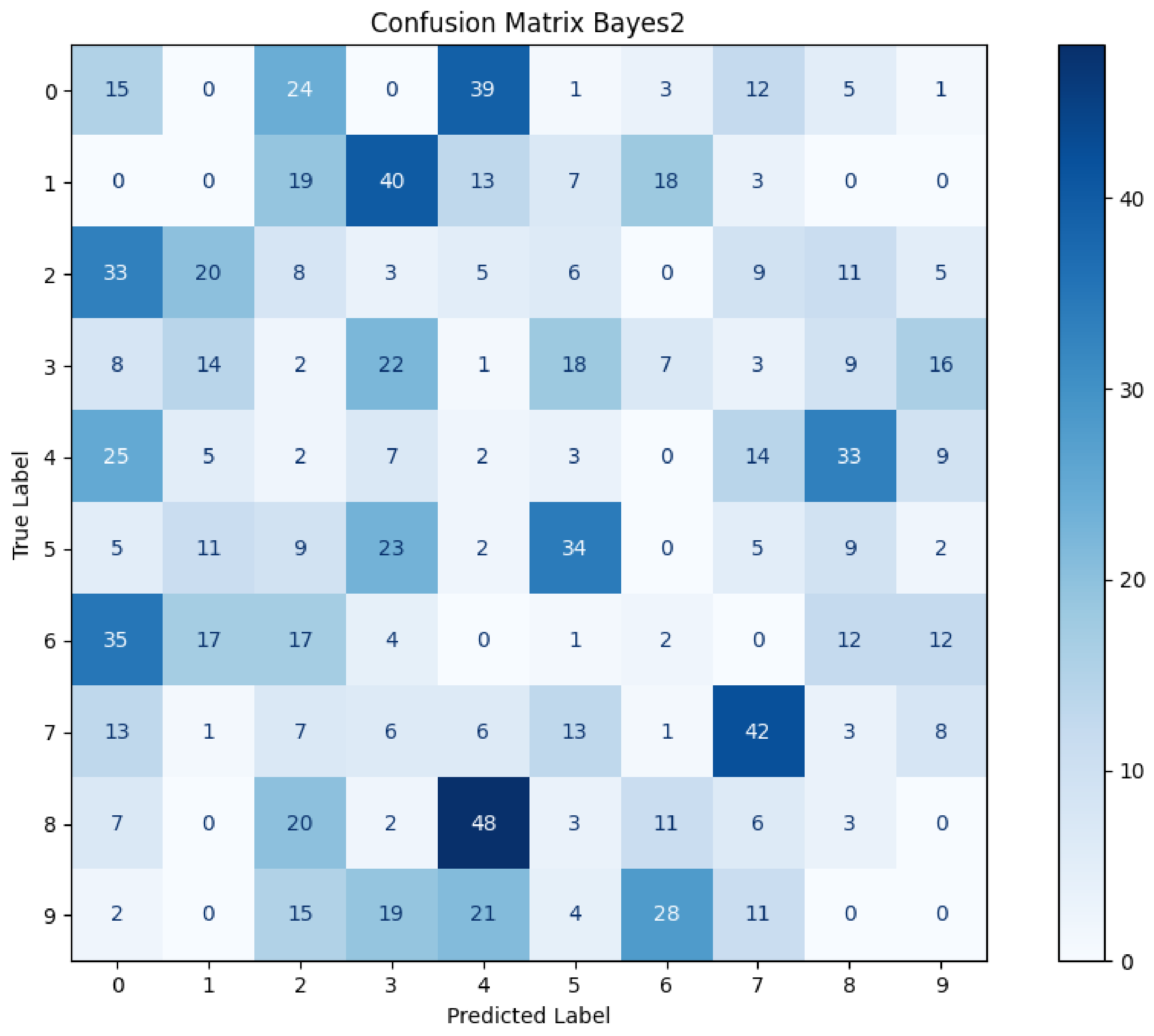
Bayes:

Model 1 (without Scikit):



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	15	0	24	0	39	1	3	12	5	1
1	0	0	19	40	13	7	18	3	0	0
2	33	20	8	3	5	6	0	9	11	5
3	8	14	2	22	1	18	7	3	9	16
4	25	5	2	7	2	3	0	14	33	9
5	5	11	9	23	2	34	0	5	9	2
6	35	17	17	4	0	1	2	0	12	12
7	13	1	7	6	6	13	1	42	3	8
8	7	0	20	2	48	3	11	6	3	0
9	2	0	15	19	21	4	28	11	0	0

Model 2 (with Scikit):



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	15	0	24	0	39	1	3	12	5	1
1	0	0	19	40	13	7	18	3	0	0
2	33	20	8	3	5	6	0	9	11	5
3	8	14	2	22	1	18	7	3	9	16
4	25	5	2	7	2	3	0	14	33	9
5	5	11	9	23	2	34	0	5	9	2
6	35	17	17	4	0	1	2	0	12	12
7	13	1	7	6	6	13	1	42	3	8
8	7	0	20	2	48	3	11	6	3	0
9	2	0	15	19	21	4	28	11	0	0

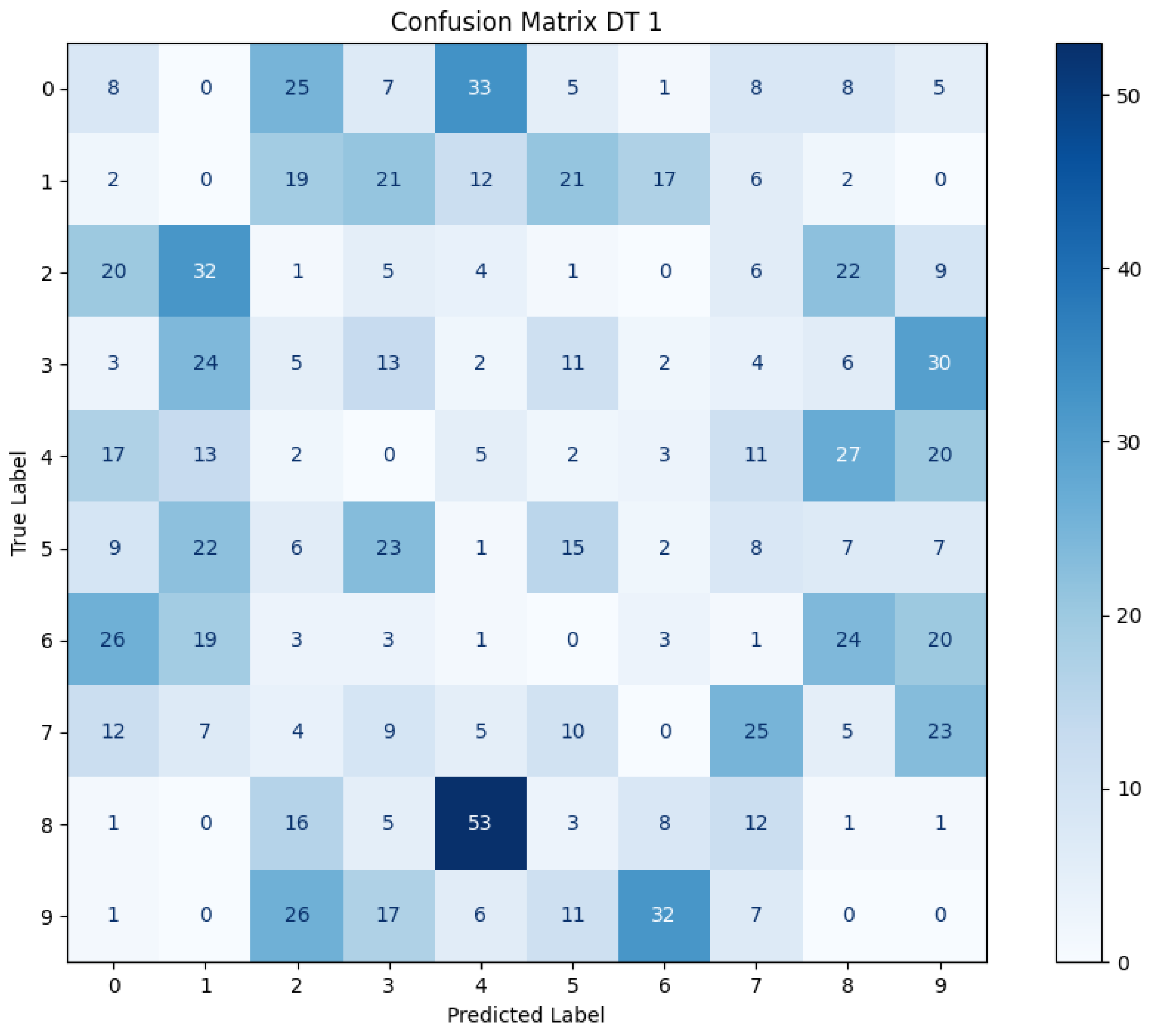
The most frequently confused classes for both these models are classes 1,4,6, and 9.

Naive Bayes assumes that features are conditionally independent, which is often not true for image data. In the case of CIFAR-10, pixel intensities or other features are highly correlated within an image, making the conditional independence assumption inappropriate. Due to this, Bayes models are likely to confuse classes that share similar features. For example, the model might misclassify airplanes as birds because both classes might have similar color patterns or shapes that the Bayes model mistakenly treats as features of the other class.

Classes that were relatively well-recognized were classes 5 and 7. Here the opposite might be true where images of those classes had independent features which the classifier could separate better and use to classify correctly.

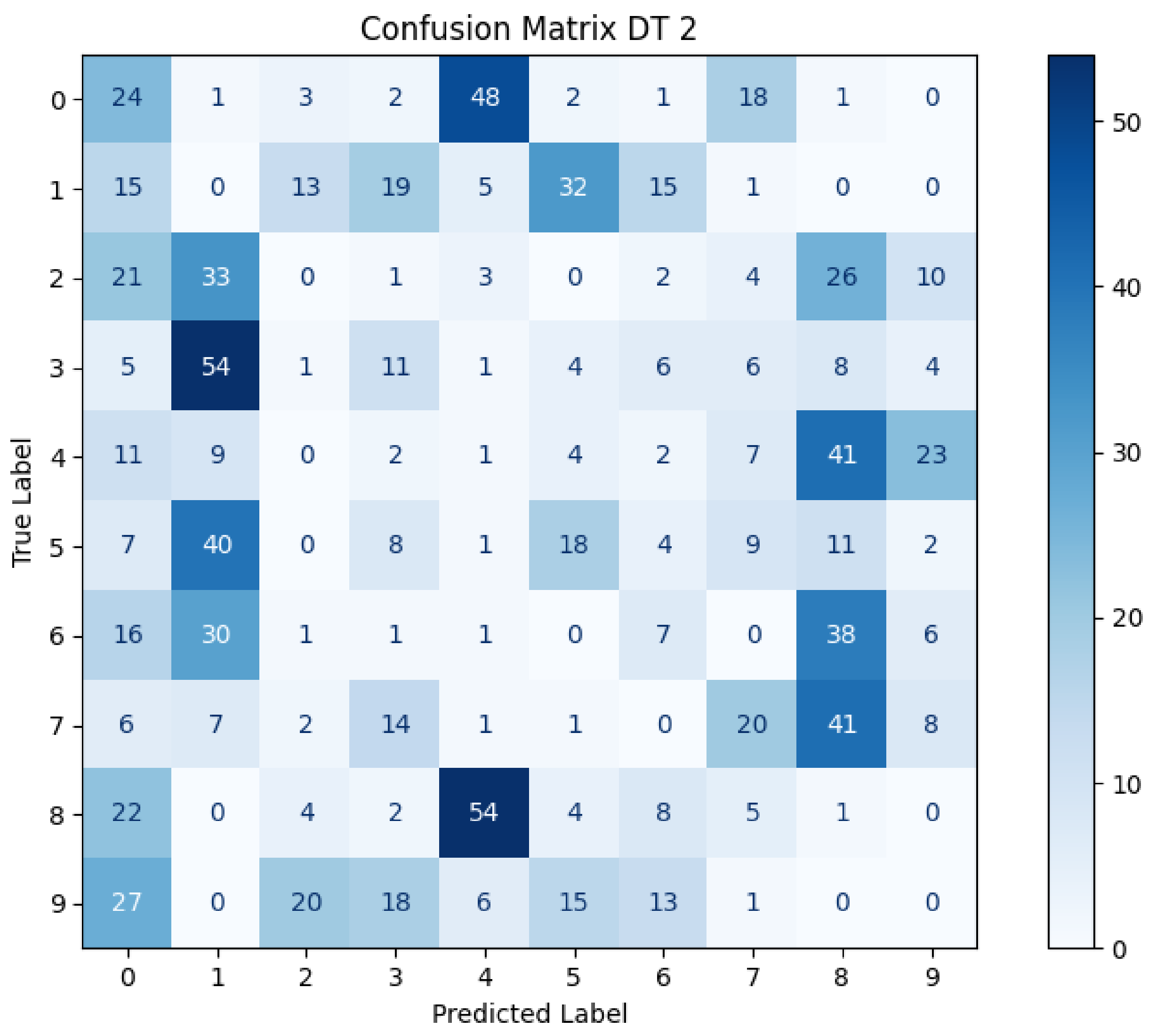
Decision Tree:

Decision tree without Scikit:



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	8	0	25	7	33	5	1	8	8	5
1	2	0	19	21	12	21	17	6	2	0
2	20	32	1	5	4	1	0	6	22	9
3	3	24	5	13	2	11	2	4	6	30
4	17	13	2	0	5	2	3	11	27	20
5	9	22	6	23	1	15	2	8	7	7
6	26	19	3	3	1	0	3	1	24	20
7	12	7	4	9	5	10	0	25	5	23
8	1	0	16	5	53	3	8	12	1	1
9	1	0	26	17	6	11	32	7	0	0

Decision tree with Scikit:



For both these models, the classes that were most frequently confused are classes 1,2,6, and 9.

Decision trees are highly prone to overfitting, especially when the tree depth is not controlled. In CIFAR-10, this means that the model memorizes specific pixel values rather than learning generalizable patterns. This could lead to the confusion of classes.

Classes that were relatively correctly identified were 0 and 7. This could be due to the images of these classes being clear, for example cars may be correctly identified if they are clearly visible in the image without objects blocking the view or noise in the image.

These are the metrics for different depths of the decision tree, using model without Scikit:

Depth 10:

```
Accuracy: 0.091
Precision: 0.09531930526137974
Recall: 0.09100000000000001
F1-Measure: 0.09310958697078736
```

Depth 20:

```
Accuracy: 0.071
Precision: 0.078523696943087
Recall: 0.07100000000000001
F1-Measure: 0.07457256069693423
```

Depth 50:

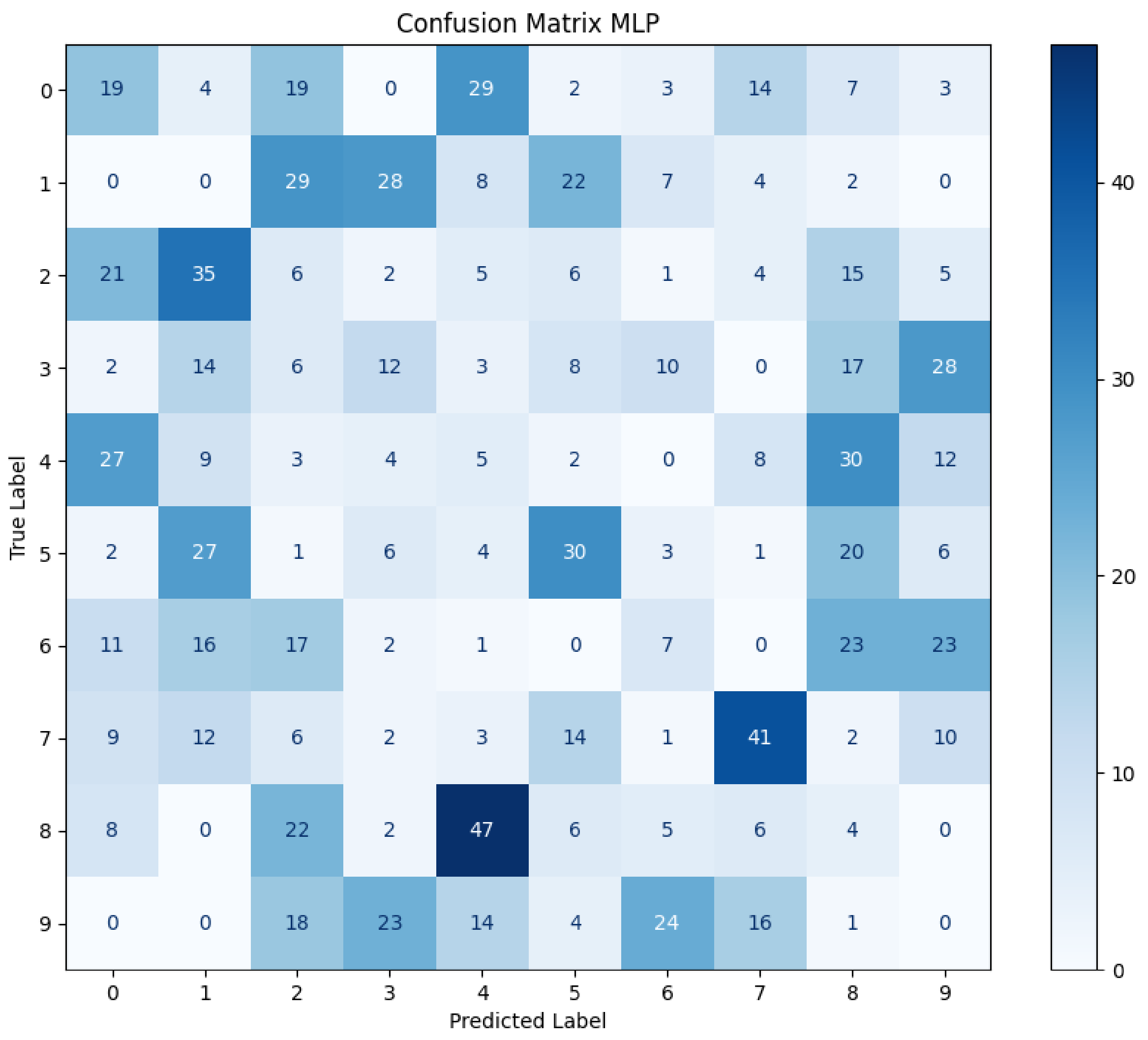
```
Accuracy: 0.071
Precision: 0.078523696943087
Recall: 0.07100000000000001
F1-Measure: 0.07457256069693423
```

Depth 100:

```
Accuracy: 0.071
Precision: 0.078523696943087
Recall: 0.07100000000000001
F1-Measure: 0.07457256069693423
```

From these metrics it can be concluded that the optimal depth is 10. At depth 20, the performance begins to deteriorate, with a noticeable decrease in all metrics compared to depth 10. This suggests that increasing the depth beyond a certain point does not lead to improved performance. The model might have started overfitting, capturing more noise and irrelevant details in the data, which doesn't generalize well to new, unseen data.

MLP:



	0	1	2	3	4	5	6	7	8	9
0	19	4	19	0	29	2	3	14	7	3
1	0	0	29	28	8	22	7	4	2	0
2	21	35	6	2	5	6	1	4	15	5
3	2	14	6	12	3	8	10	0	17	28
4	27	9	3	4	5	2	0	8	30	12
5	2	27	1	6	4	30	3	1	20	6
6	11	16	17	2	1	0	7	0	23	23
7	9	12	6	2	3	14	1	41	2	10
8	8	0	22	2	47	6	5	6	4	0
9	0	0	18	23	14	4	24	16	1	0

For this model, the classes that were most frequently confused are classes 1,2,8, and 9.

This confusion can be created by the model's inability to distinguish between classes that rely on subtle visual differences. This is because the MLP lacks spatial awareness and treats input features as independent, ignoring their local spatial relationships. Thus subtle variations in shapes or textures, like those between objects in classes 1, 2, 8, and 9, may not be effectively captured.

Classes that were identified pretty well were classes 5 and 7.

This success can be due to the fact that these classes contain strong, globally distinguishable features which are easier for the MLP to classify. Also, objects with clear and dominant patterns (like airplanes or ships) are less prone to misclassification because the MLP can effectively capture these features.

Metrics of MLP with different depths (size 512):

All layers:

```
Accuracy: 0.124
Precision: 0.1332459339814714
Recall: 0.124
F1-Measure: 0.12845680829997114
```

Just the first layer:

```
Accuracy: 0.1062124248496994
Precision: 0.11845686616231677
Recall: 0.10683673469387756
F1-Measure: 0.1123471304533827
```

Layers 1 and 2:

```
Metrics:
Accuracy: 0.124
Precision: 0.1239245817547618
Recall: 0.124
F1-Measure: 0.12396227940632855
```

Layers 1 and 3:

```
Metrics:
Accuracy: 0.135
Precision: 0.1330140076075602
Recall: 0.135
F1-Measure: 0.13399964567011755
```

From these metrics it can be concluded that the addition of layer 2 does not significantly improve performance, as seen when comparing metrics for layers 1 and 2 vs. layers 1 and

3. This could indicate that the second layer does not extract distinct or meaningful features for CIFAR-10 and might be redundant.

Increasing the depth from 1 to 2 does capture more detailed features, but once we introduce a third layer (in this case layer 2), the model risks overfitting.

Metrics of MLP with all layers but different sizes of layers:

Size 128:

```
Accuracy: 0.129
Precision: 0.14274993640989633
Recall: 0.129
F1-Measure: 0.1355271102555152
```

Took 5.7 seconds.

Size 256:

```
Accuracy: 0.134
Precision: 0.1457299039181313
Recall: 0.134
F1-Measure: 0.13961901714122638
```

Took 5.8 seconds

Size 512:

```
Accuracy: 0.124
Precision: 0.1332459339814714
Recall: 0.124
F1-Measure: 0.12845680829997114
```

Took 5.9 seconds

Size 1024:

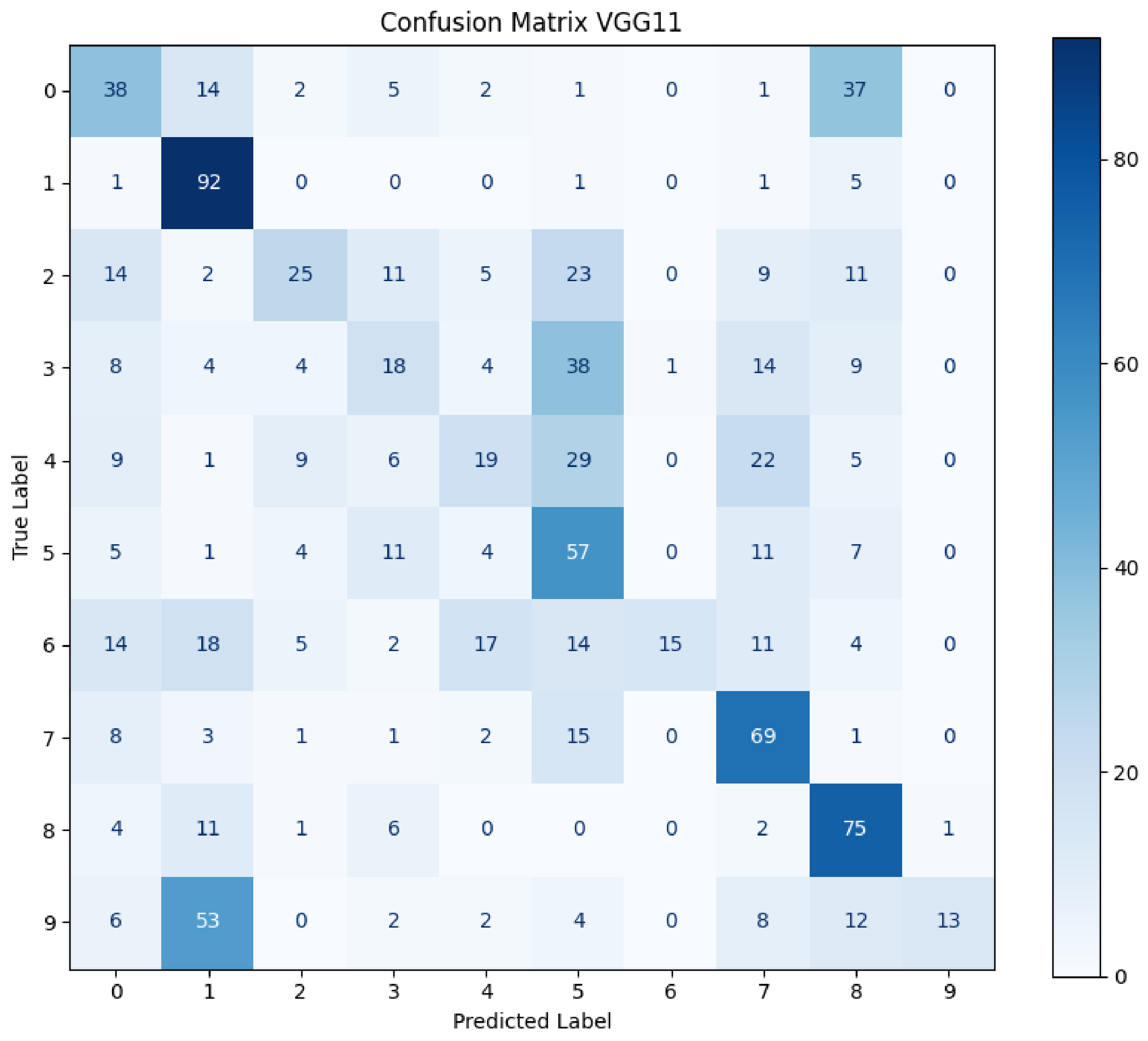
```
Accuracy: 0.116
Precision: 0.12919522501303557
Recall: 0.11600000000000002
F1-Measure: 0.12224256080611175
```

Took 10 seconds

From these metrics and computation times we can conclude that a layer size of 256 performs the best. Even thought it is the second-fastest after size 128, there is a mere 0.1 second difference but it’s further metrics are clear from the others. This suggests that this size is sufficient to learn and generalize from the CIFAR-10 dataset.

Larger sizes (512, 1024) did not translate to better performance, as the increased capacity likely led to overfitting, where the model memorized noise or unnecessary details instead of focusing on generalizable patterns. These sizes do not only cause worse performance, they also took a higher computational cost.

CNN:



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	38	14	2	5	2	1	0	1	37	0
1	1	92	0	0	0	1	0	1	5	0
2	14	2	25	11	5	23	0	9	11	0
3	8	4	4	18	4	38	1	14	9	0
4	9	1	9	6	19	29	0	22	5	0
5	5	1	4	11	4	57	0	11	7	0
6	14	18	5	2	17	14	15	11	4	0
7	8	3	1	1	2	15	0	69	1	0
8	4	11	1	6	0	0	0	2	75	1
9	6	53	0	2	2	4	0	8	12	13

For this CNN, the classes that were most frequently confused are classes 6 and 9.

There are some reasons as to why classes can be confused in a CNN, firstly, if the input images lack sufficient resolution or if the CNN's filters fail to capture the distinguishing micro-patterns, subtle differences may go unnoticed. Another reason could be that the two class overlap in features, meaning they have the same textures, colors, or shapes.

Classes that were identified well were classes 1,5, 7, and 8.

These classes could be classified successfully because the model’s convolutional layers allow it to break the input down into simple patterns. Later layers would then use these patterns to form more complex shapes and features. This process helps the model recognize and classify objects that have clear and unique characteristics. Another reason is that classes with clear textures or specific features (like animal stripes or car wheels) are easier for the CNN to recognize because it focuses on spatial patterns in the images.

Metrics of CNN with different depths (with a 3x3 kernel):

All layers:

```
Accuracy: 0.421
Precision: 0.5040706636126033
Recall: 0.421
F1-Measure: 0.4588054896307378
```

Removed the last 2 convolutional layers:

```
Accuracy: 0.43
Precision: 0.554005190446216
Recall: 0.43
F1-Measure: 0.48418897421434626
```

Removed all but the first convolutional layer:

```
Accuracy: 0.302
Precision: 0.3437505255241379
Recall: 0.302
F1-Measure: 0.3215255879939944
```

From these metrics it can be concluded that an increase in the depth helps the CNN model capture more detailed and complex features, leading to better performance overall. The addition of the last two layers does however lead to overfitting since the model performs better excluding those layers.

Metrics of CNN with different kernel size (with all layers):

2x2 kernel:

```
Accuracy: 0.246
Precision: 0.4208108727856067
Recall: 0.246
F1-Measure: 0.3104912620059897
```

Took 130 seconds.

3x3 kernel:

```
Accuracy: 0.421
Precision: 0.5040706636126033
Recall: 0.421
F1-Measure: 0.4588054896307378
```

Took 145 seconds.

7x7 kernel:

```
Accuracy: 0.212
Precision: 0.08265259049907342
Recall: 0.21200000000000002
F1-Measure: 0.11893565338166383
```

Took 665 seconds

From these metrics we can conclude the following:

The small 2x2 kernel emphasizes finer details but fails to capture broader spatial patterns, which are critical for recognizing complex object features in CIFAR-10. The computation time is the lowest among the kernel sizes.

The 3x3 kernel achieves optimal spatial granularity, effectively capturing fine details while recognizing broader patterns. The computation time is higher than that of the 2x2 kernel, however its 15 seconds difference is very feasible when considering the way better granularity. This is by far the best kernel size to choose considering the high feature recognition and low computational cost.

The large 7x7 kernel captures broader spatial features but loses finer details, which are crucial for identifying smaller objects or intricate patterns in CIFAR-10 images. This fact with the significant computation time makes this the least efficient kernel size to choose.

The best model:

After computing metrics for all 4 models, the CNN performs the best by far. Even though it takes significantly more computation time (145 seconds compared to the 5.8 seconds by the MLP), the percentages of its metrics almost quadruple those of the other models.

One of the reasons why CNN performs so much better is because it automatically learns spatial hierarchies of features (e.g., edges, textures, and patterns) using convolutional layers, which are specifically designed to capture local patterns in data. This capability is great for the CIFAR-10 dataset which is concerned with image recognition, where relationships between neighboring pixels matter.