# COMP 472: Artificial Intelligence
# Project Report

Steven Gingras (40098045)

November 27, 2024

# 1 Model Architectures and Training

## 1.1 Naive Bayes

### 1.1.1 Implementation Details

Input: ResNet-18 Feature Vectors `[50,]`
Training Methodology: The Gaussian Naive Bayes model is trained simply by computing the means and variances of each class.
Model Structure: The model consists of a list of means and variances where the index represents the feature means/variances of that particular class:

```
means = [class0Means, class1Means, ..., class9Means]
variances = [class0Variances, class1Variances, ..., class9Variances]
```

These values allow the model to make class predictions by returning the class yielding the highest **sum of log likelihood** value.
The following formulas are implemented in the `gaussianLikelihood(x, mu, sigma)` and `predictClass(x, means, variances)` methods:

$$\text{Gaussian Likelihood} = \frac{1}{\sqrt{2\pi\sigma}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma}\right) \tag{1}$$

$$\text{Score} = \sum_{i=0}^{n-1} \log(\text{Gaussian Likelihood}) \tag{2}$$

$$\text{Predicted Class} = argmax(Scores) \tag{3}$$

Where:
$n$: Number of classes (10)
$\mu$: class$_i$ Mean
$\sigma$: class$_i$ Variance

### 1.1.2 Bayes Model Variants

There are two variants of this model, one has been implemented manually using the above algorithm and the other uses the Scikit implementation of the Gaussian Naive Bayes model.

1. Main Model: Manually Implemented

2. Scikit Model: Uses Scikit's model

## 1.2 Decision Tree

### 1.2.1 Implementation Details

Input: ResNet-18 Feature Vectors `[50,]`
Training Methodology: The Decision Tree model is trained by creating a **binary tree** from the data.
Parent nodes: split the data into two child nodes based on both: a **feature index** and a **split point**.
All data points with `x[featureIndex] < splitPoint` go to the left child while the remaining points go to the right child.
Leaf nodes: return the **majority class** contained in its data.

Gini Coefficient
The Gini coefficient is used to measure the level of impurity in a dataset. The following formula is implemented in the `computeGini(data)` function:

$$\text{Gini} = 1 - \sum_{i=0}^{C-1} \frac{n_i}{N}\left(1 - \frac{n_i}{N}\right) \tag{4}$$

Best Data Splits
A split is considered best if it results in the lowest weighted Gini coefficient represented by the following formula:

$$\text{Weighted Gini} = Gini_L \cdot \frac{n_L}{N} + Gini_R \cdot \frac{n_R}{N} \tag{5}$$

The `bestSplit(data)` method iterates through each `featureIndex`, gathers all values of that feature in the data, determines the minimum and maximum and iterates through each `int` in that range as a `splitPoint`. Each iteration is considered as a potential split and the above formula is used to compute its weighted Gini.
From the iterations, the `featureIndex` and `splitPoint` yielding the lowest weighted Gini are returned as the best split in the data (unless the Gini gain is insignificant).

Tree building
The `buildTree(data, depth, maxDepth)` method recursively builds the tree.
The stopping conditions where **leaf nodes are returned instead of splitting** are as follows:

- `maxDepth` has been reached.
- The data already has 0 impurity.
- The data has less than 20 elements.
- Gini gain from the best split is insignificant (less than 0.01).

Otherwise, the data is split using the process detailed above.

### 1.2.2 Decision Tree Variants

The variants of this model are characterized by differing `maxDepth` hyperparameters for the above algorithm. There is also a variant that uses the Scikit implementation of a Decision Tree. The differences in the variants are as follows:

1. Main Model: `maxDepth = 50`
2. Scikit Model: Uses Scikit's model with `maxDepth = 50`
3. Reduced Depth Model: `maxDepth = 40`
4. Increased Depth Model: `maxDepth = 60`

## 1.3 Multi-Layer Perceptron

### 1.3.1 Implementation Details

Input: ResNet-18 Feature Vectors `[50,]`
Training Methodology: The Multi-Layer Perceptron model is trained using 3 layers as follows:

1. Linear(50, 512), ReLU()
2. Linear(512, 512), BatchNorm(512), ReLU()
3. Linear(512, 10)

Training Loop Algorithm
For each epoch and each batch:

1. Outputs are obtained: Using the model's current state, inputs are passed through the network to obtain the outputs.
2. Loss is computed: The loss function is applied to the outputs to obtain the loss.
3. Gradients are reset and calculated: Partial derivates are computed using the chain rule and backpropagation.
4. Weight and bias parameters are updated using the gradients at a ratio of the learning rate.

Loss Function
The loss function used is the `nn.CrossEntropyLoss()` from the PyTorch nn library. The formula used to compute this loss is the following:

$$\text{Loss} = l_n = -log\frac{\exp(x_c)}{\sum_{i=0}^{C-1}\exp(x_i)} \tag{6}$$

Where $x_c$ is the output logit of the **correct** class.
The average of loss in the batch is then taken.
Optimizer
The optimizer function used is the SGD optimizer with a momentum of 0.9. The learning rate used for this function is: $lr = \mathbf{0.001}$, since values higher than this lead to more significant overfitting while higher values lead to less accurate evaluations. The optimizer uses the loss computed in the previous step to compute gradients. Finally, weights are adjusted for each layer by the gradients by a factor of the learning rate.

Batch Size

The batch size used for this algorithm was 32, not too high so as to overload the memory but high enough to make better **generalizations** on the data.

Number of Epochs

The number of epochs used was 9 since this led to the highest test accuracy (8 and 10 were lower than 9).

### 1.3.2 MLP Variants

The variants of this model are characterized by differing `numOfLayers` and `hiddenLayerSize` hyperparameters for the above algorithm. The differences in the variants are as follows:

1. Main Model: `numOfLayers = 3, hiddenLayerSize = 512`
2. Reduced Depth Model: `numOfLayers = 2, hiddenLayerSize = 512`
3. Increased Depth Model: `numOfLayers = 4, hiddenLayerSize = 512`
4. Reduced Layer Size Model: `numOfLayers = 3, hiddenLayerSize = 256`
5. Increased Layer Size Model: `numOfLayers = 3, hiddenLayerSize = 1024`

## 1.4 Convolutional Neural Network

Input: ResNet-18 Images `[3, 224, 224]`
Training Methodology: The Convolutional Neural Network model is trained using 11 layers as follows:

1. Conv(**3**, 64, 3, 1, 1), BatchNorm(64), ReLU(), MaxPool(2, 2)
2. Conv(64, 128, 3, 1, 1), BatchNorm(128), ReLU(), MaxPool(2, 2)
3. Conv(128, 256, 3, 1, 1), BatchNorm(256), ReLU()
4. Conv(256, 256, 3, 1, 1), BatchNorm(256), ReLU(), MaxPool(2, 2)
5. Conv(256, 512, 3, 1, 1), BatchNorm(512), ReLU()
6. Conv(512, 512, 3, 1, 1), BatchNorm(512), ReLU(), MaxPool(2, 2)
7. Conv(512, 512, 3, 1, 1), BatchNorm(512), ReLU()
8. Conv(512, 512, 3, 1, 1), BatchNorm(512), ReLU(), MaxPool(2, 2)
9. **Flatten()**, Linear(**25088**,4096), ReLU(), Dropout(0.5)
10. Linear(4096,4096), ReLU(), Dropout(0.5)
11. Linear(4096,10)

Notes:

- 3 input channels are used for the outer layer to allow for the 3 RGB channels.
- a Flatten() is used prior to the 9th layer to format the 2D tensors into an acceptable shape for Linear()

Training Loop Algorithm, Loss Function, Optimizer, Batch Size
Processes and explanations for these in the CNN model are the same as in the MLP model. Please see the explanations above for these.
Number of Epochs
The number of epochs used was 20 since this led to the highest test accuracy.

### 1.4.1 CNN Variants

The variants of this model are characterized by differing `numOfLayers` and `kernelSize` hyperparameters for the above algorithm. The differences in the variants are as follows:

1. Main Model: `numOfLayers = 11, kernelSize = 3`
2. Reduced Depth Model: `numOfLayers = 10, kernelSize = 3`
3. Increased Depth Model: `numOfLayers = 12, kernelSize = 3`
4. Reduced Kernel Size Model: `numOfLayers = 11, kernelSize = 2`
5. Increased Kernel Size Model: `numOfLayers = 11, kernelSize = 4`

## 2 Evaluation

### 2.1 Metrics

| | Testing Set: Unseen Data | | | | Training Set: Seen Data | | | |
|---|---|---|---|---|---|---|---|---|
| Model | Accuracy | Precision | Recall | F1 | Accuracy | Precision | Recall | F1 |
| Naive Bayes Main | 79.40 | 80.00 | 79.40 | 79.79 | 81.68 | 82.00 | 81.68 | 81.84 |
| Naive Bayes Scikit | 79.30 | 79.91 | 79.30 | 79.61 | 81.70 | 82.01 | 81.70 | 81.86 |
| Decision Tree Main | 56.20 | 56.61 | 56.20 | 56.41 | 79.16 | 79.40 | 79.16 | 79.28 |
| Decision Tree Scikit | 61.20 | 61.00 | 61.20 | 61.10 | 72.24 | 72.08 | 72.24 | 72.16 |
| Decision Tree Reduced Depth | 57.80 | 58.24 | 57.80 | 58.02 | 81.42 | 81.60 | 81.42 | 81.51 |
| Decision Tree Increased Depth | 57.80 | 58.24 | 57.80 | 58.02 | 81.42 | 81.60 | 81.42 | 81.51 |
| MLP Main | 84.20 | 84.30 | 84.20 | 84.25 | 92.58 | 92.59 | 92.58 | 92.58 |
| MLP Reduced Depth | 82.80 | 82.87 | 82.80 | 82.83 | 87.18 | 87.27 | 87.18 | 87.22 |
| MLP Increased Depth | 83.10 | 83.50 | 83.10 | 83.30 | 98.06 | 98.07 | 98.06 | 98.06 |
| MLP Reduced Layer Size | 83.20 | 83.30 | 83.20 | 83.25 | 89.82 | 89.85 | 89.82 | 89.83 |
| MLP Increased Layer Size | 83.20 | 83.68 | 83.20 | 83.44 | 95.78 | 95.80 | 95.78 | 95.79 |
| CNN Main | 57.10 | 61.40 | 57.10 | 59.17 | 73.96 | 77.04 | 73.96 | 75.47 |
| CNN Reduced Depth | 65.70 | 67.48 | 65.70 | 66.58 | 98.56 | 98.58 | 98.56 | 98.57 |
| CNN Increased Depth | 61.10 | 66.58 | 61.10 | 63.72 | 94.54 | 95.43 | 94.54 | 94.98 |
| CNN Reduced Kernel Size | 61.30 | 63.17 | 61.30 | 62.22 | 98.24 | 98.27 | 98.24 | 98.25 |
| CNN Increased Kernel Size | 64.70 | 67.19 | 64.70 | 65.92 | 99.24 | 99.26 | 99.24 | 99.25 |

Table 1: Model Metrics on Unseen and Seen Data (in percentages)

### 2.2 Insights

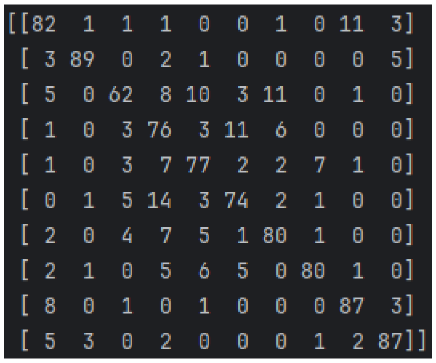
- The accuracy, precision, recall and f1 measure were consistenly similar in magnitude within test cases.
- If a model were to have a high precision measure while recall was lower, it would indicate that the model is well suited to avoiding false positives but not to capturing true positives.
- In the context of facial recognition a high precision measure would be less likely to make a false match but also less likely to make a true match.

## 2.3 Confusion Matrices (Left: Unseen Data — Right: Seen Data)

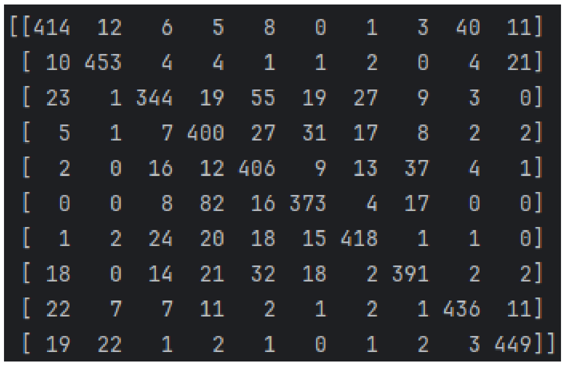
| Axis | Index Value | 0 index |
|---|---|---|
| Row | Ground Truth Class | Top |
| Column | Predicted Class | Left |

Table 2: Confusion Matrix Legend

Main Bayes Model:

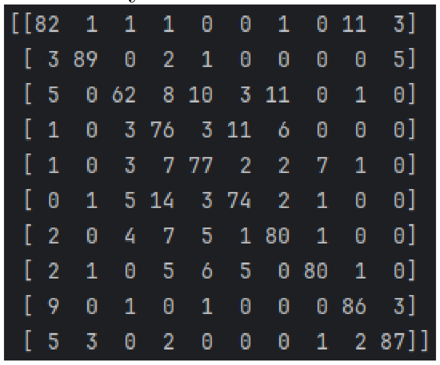


```
[[82  1  1  1  0  0  1  0 11  3]
 [ 3 89  0  2  1  0  0  0  0  5]
 [ 5  0 62  8 10  3 11  0  1  0]
 [ 1  0  3 76  3 11  6  0  0  0]
 [ 1  0  3  7 77  2  2  7  1  0]
 [ 0  1  5 14  3 74  2  1  0  0]
 [ 2  0  4  7  5  1 80  1  0  0]
 [ 2  1  0  5  6  5  0 80  1  0]
 [ 8  0  1  0  1  0  0  0 87  3]
 [ 5  3  0  2  0  0  0  1  2 87]]
```

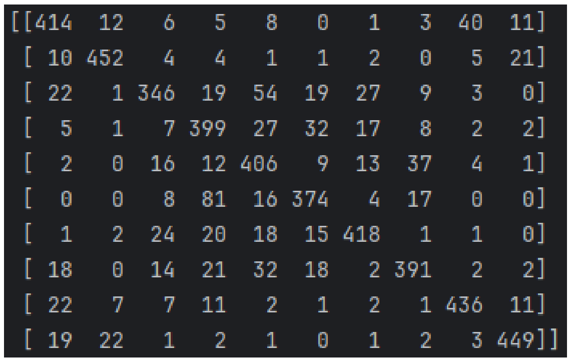


```
[[414  12   6   5   8   0   1   3  40  11]
 [ 10 453   4   4   1   1   2   0   4  21]
 [ 23   1 344  19  55  19  27   9   3   0]
 [  5   1   7 400  27  31  17   8   2   2]
 [  2   0  16  12 406   9  13  37   4   1]
 [  0   0   8  82  16 373   4  17   0   0]
 [  1   2  24  20  18  15 418   1   1   0]
 [ 18   0  14  21  32  18   2 391   2   2]
 [ 22   7   7  11   2   1   2   1 436  11]
 [ 19  22   1   2   1   0   1   2   3 449]]
```

Scikit Bayes Model:

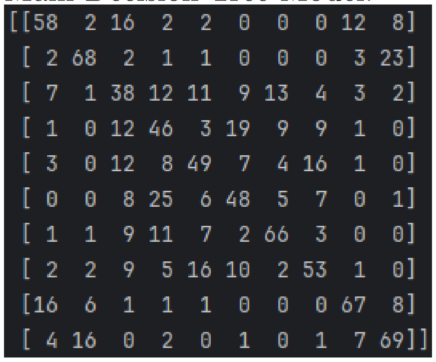


```
[[82  1  1  1  0  0  1  0 11  3]
 [ 3 89  0  2  1  0  0  0  0  5]
 [ 5  0 62  8 10  3 11  0  1  0]
 [ 1  0  3 76  3 11  6  0  0  0]
 [ 1  0  3  7 77  2  2  7  1  0]
 [ 0  1  5 14  3 74  2  1  0  0]
 [ 2  0  4  7  5  1 80  1  0  0]
 [ 2  1  0  5  6  5  0 80  1  0]
 [ 9  0  1  0  1  0  0  0 86  3]
 [ 5  3  0  2  0  0  0  1  2 87]]
```

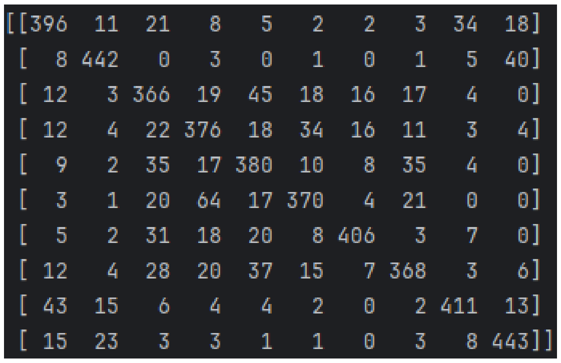


```
[[414  12   6   5   8   0   1   3  40  11]
 [ 10 452   4   4   1   1   2   0   5  21]
 [ 22   1 346  19  54  19  27   9   3   0]
 [  5   1   7 399  27  32  17   8   2   2]
 [  2   0  16  12 406   9  13  37   4   1]
 [  0   0   8  81  16 374   4  17   0   0]
 [  1   2  24  20  18  15 418   1   1   0]
 [ 18   0  14  21  32  18   2 391   2   2]
 [ 22   7   7  11   2   1   2   1 436  11]
 [ 19  22   1   2   1   0   1   2   3 449]]
```

Main Decision Tree Model:

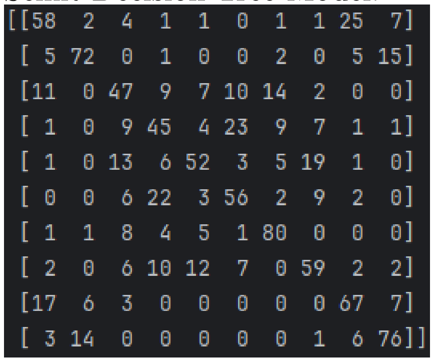


```
[[58  2 16  2  2  0  0  0 12  8]
 [ 2 68  2  1  1  0  0  0  3 23]
 [ 7  1 38 12 11  9 13  4  3  2]
 [ 1  0 12 46  3 19  9  9  1  0]
 [ 3  0 12  8 49  7  4 16  1  0]
 [ 0  0  8 25  6 48  5  7  0  1]
 [ 1  1  9 11  7  2 66  3  0  0]
 [ 2  2  9  5 16 10  2 53  1  0]
 [16  6  1  1  1  0  0  0 67  8]
 [ 4 16  0  2  0  1  0  1  7 69]]
```

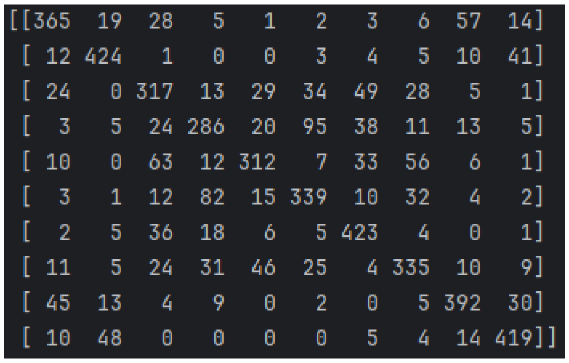


```
[[396  11  21   8   5   2   2   3  34  18]
 [  8 442   0   3   0   1   0   1   5  40]
 [ 12   3 366  19  45  18  16  17   4   0]
 [ 12   4  22 376  18  34  16  11   3   4]
 [  9   2  35  17 380  10   8  35   4   0]
 [  3   1  20  64  17 370   4  21   0   0]
 [  5   2  31  18  20   8 406   3   7   0]
 [ 12   4  28  20  37  15   7 368   3   6]
 [ 43  15   6   4   4   2   0   2 411  13]
 [ 15  23   3   3   1   1   0   3   8 443]]
```

Scikit Decision Tree Model:

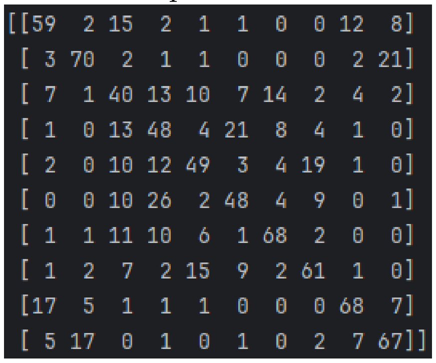


```
[[58  2  4  1  1  0  1  1 25  7]
 [ 5 72  0  1  0  0  2  0  5 15]
 [11  0 47  9  7 10 14  2  0  0]
 [ 1  0  9 45  4 23  9  7  1  1]
 [ 1  0 13  6 52  3  5 19  1  0]
 [ 0  0  6 22  3 56  2  9  2  0]
 [ 1  1  8  4  5  1 80  0  0  0]
 [ 2  0  6 10 12  7  0 59  2  2]
 [17  6  3  0  0  0  0  0 67  7]
 [ 3 14  0  0  0  0  0  1  6 76]]
```

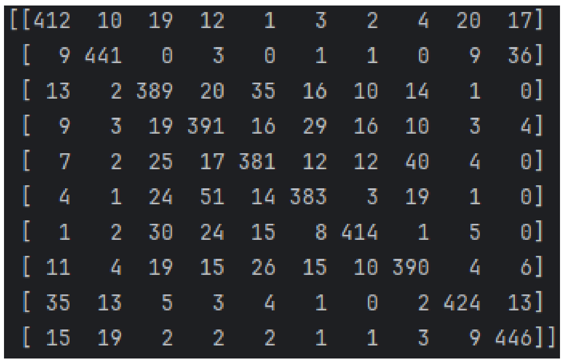


```
[[365  19  28   5   1   2   3   6  57  14]
 [ 12 424   1   0   0   3   4   5  10  41]
 [ 24   0 317  13  29  34  49  28   5   1]
 [  3   5  24 286  20  95  38  11  13   5]
 [ 10   0  63  12 312   7  33  56   6   1]
 [  3   1  12  82  15 339  10  32   4   2]
 [  2   5  36  18   6   5 423   4   0   1]
 [ 11   5  24  31  46  25   4 335  10   9]
 [ 45  13   4   9   0   2   0   5 392  30]
 [ 10  48   0   0   0   0   5   4  14 419]]
```

Reduced Depth Decision Tree Model:

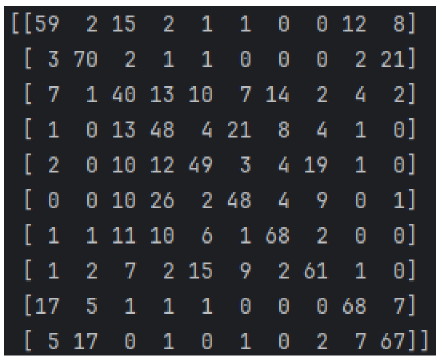


```
[[59  2 15  2  1  1  0  0 12  8]
 [ 3 70  2  1  1  0  0  0  2 21]
 [ 7  1 40 13 10  7 14  2  4  2]
 [ 1  0 13 48  4 21  8  4  1  0]
 [ 2  0 10 12 49  3  4 19  1  0]
 [ 0  0 10 26  2 48  4  9  0  1]
 [ 1  1 11 10  6  1 68  2  0  0]
 [ 1  2  7  2 15  9  2 61  1  0]
 [17  5  1  1  1  0  0  0 68  7]
 [ 5 17  0  1  0  1  0  2  7 67]]
```

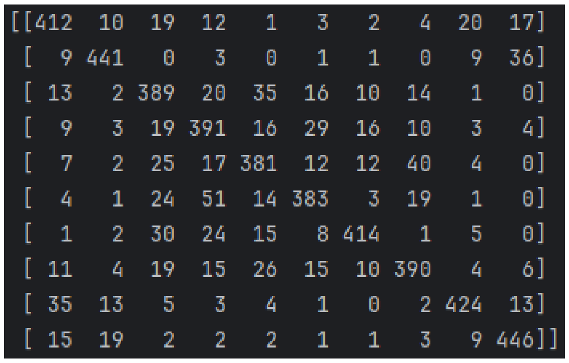


```
[[412  10  19  12   1   3   2   4  20  17]
 [  9 441   0   3   0   1   1   0   9  36]
 [ 13   2 389  20  35  16  10  14   1   0]
 [  9   3  19 391  16  29  16  10   3   4]
 [  7   2  25  17 381  12  12  40   4   0]
 [  4   1  24  51  14 383   3  19   1   0]
 [  1   2  30  24  15   8 414   1   5   0]
 [ 11   4  19  15  26  15  10 390   4   6]
 [ 35  13   5   3   4   1   0   2 424  13]
 [ 15  19   2   2   2   1   1   3   9 446]]
```

Increased Depth Decision Tree Model:

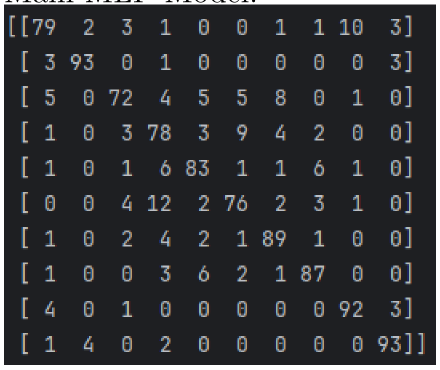


```
[[59  2 15  2  1  1  0  0 12  8]
 [ 3 70  2  1  1  0  0  0  2 21]
 [ 7  1 40 13 10  7 14  2  4  2]
 [ 1  0 13 48  4 21  8  4  1  0]
 [ 2  0 10 12 49  3  4 19  1  0]
 [ 0  0 10 26  2 48  4  9  0  1]
 [ 1  1 11 10  6  1 68  2  0  0]
 [ 1  2  7  2 15  9  2 61  1  0]
 [17  5  1  1  1  0  0  0 68  7]
 [ 5 17  0  1  0  1  0  2  7 67]]
```

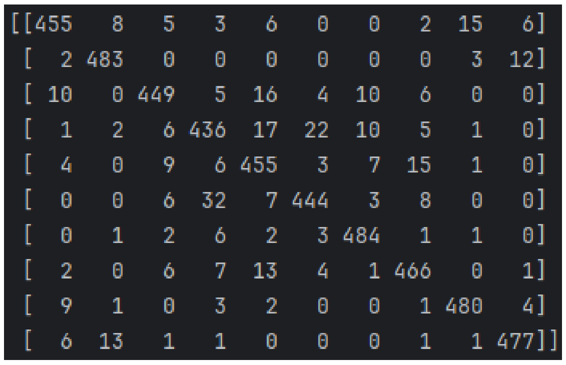


```
[[412  10  19  12   1   3   2   4  20  17]
 [  9 441   0   3   0   1   1   0   9  36]
 [ 13   2 389  20  35  16  10  14   1   0]
 [  9   3  19 391  16  29  16  10   3   4]
 [  7   2  25  17 381  12  12  40   4   0]
 [  4   1  24  51  14 383   3  19   1   0]
 [  1   2  30  24  15   8 414   1   5   0]
 [ 11   4  19  15  26  15  10 390   4   6]
 [ 35  13   5   3   4   1   0   2 424  13]
 [ 15  19   2   2   2   1   1   3   9 446]]
```

Main MLP Model:

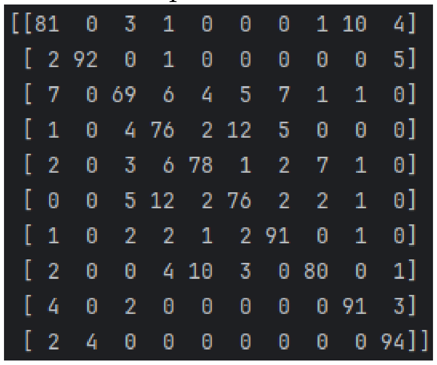


```
[[79  2  3  1  0  0  1  1 10  3]
 [ 3 93  0  1  0  0  0  0  0  3]
 [ 5  0 72  4  5  5  8  0  1  0]
 [ 1  0  3 78  3  9  4  2  0  0]
 [ 1  0  1  6 83  1  1  6  1  0]
 [ 0  0  4 12  2 76  2  3  1  0]
 [ 1  0  2  4  2  1 89  1  0  0]
 [ 1  0  0  3  6  2  1 87  0  0]
 [ 4  0  1  0  0  0  0  0 92  3]
 [ 1  4  0  2  0  0  0  0  0 93]]
```

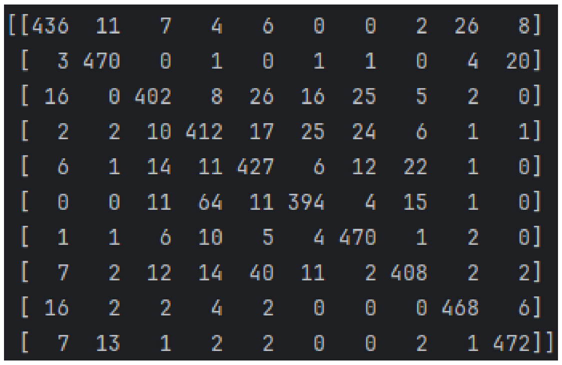


```
[[455   8   5   3   6   0   0   2  15   6]
 [  2 483   0   0   0   0   0   0   3  12]
 [ 10   0 449   5  16   4  10   6   0   0]
 [  1   2   6 436  17  22  10   5   1   0]
 [  4   0   9   6 455   3   7  15   1   0]
 [  0   0   6  32   7 444   3   8   0   0]
 [  0   1   2   6   2   3 484   1   1   0]
 [  2   0   6   7  13   4   1 466   0   1]
 [  9   1   0   3   2   0   0   1 480   4]
 [  6  13   1   1   0   0   0   1   1 477]]
```

Reduced Depth MLP Model:

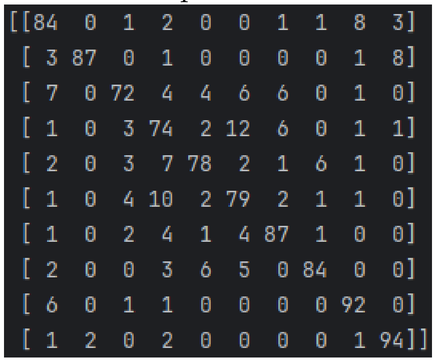


```
[[81  0  3  1  0  0  0  1 10  4]
 [ 2 92  0  1  0  0  0  0  0  5]
 [ 7  0 69  6  4  5  7  1  1  0]
 [ 1  0  4 76  2 12  5  0  0  0]
 [ 2  0  3  6 78  1  2  7  1  0]
 [ 0  0  5 12  2 76  2  2  1  0]
 [ 1  0  2  2  1  2 91  0  1  0]
 [ 2  0  0  4 10  3  0 80  0  1]
 [ 4  0  2  0  0  0  0  0 91  3]
 [ 2  4  0  0  0  0  0  0  0 94]]
```

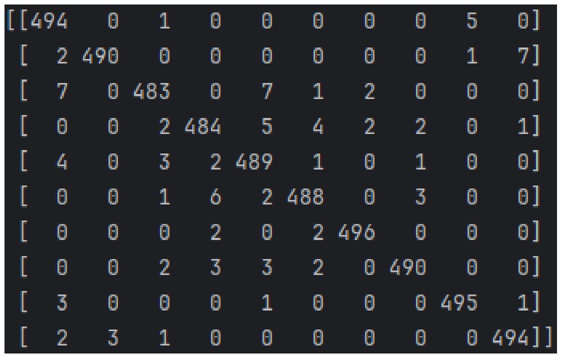


```
[[436  11   7   4   6   0   0   2  26   8]
 [  3 470   0   1   0   1   1   0   4  20]
 [ 16   0 402   8  26  16  25   5   2   0]
 [  2   2  10 412  17  25  24   6   1   1]
 [  6   1  14  11 427   6  12  22   1   0]
 [  0   0  11  64  11 394   4  15   1   0]
 [  1   1   6  10   5   4 470   1   2   0]
 [  7   2  12  14  40  11   2 408   2   2]
 [ 16   2   2   4   2   0   0   0 468   6]
 [  7  13   1   2   2   0   0   2   1 472]]
```

Increased Depth MLP Model:

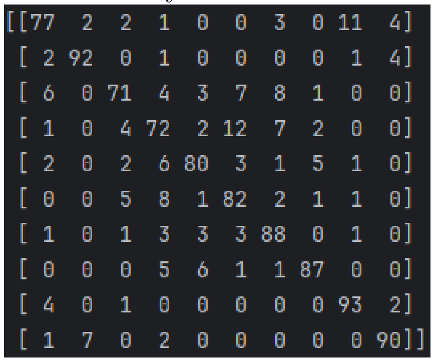


```
[[84  0  1  2  0  0  1  1  8  3]
 [ 3 87  0  1  0  0  0  0  1  8]
 [ 7  0 72  4  4  6  6  0  1  0]
 [ 1  0  3 74  2 12  6  0  1  1]
 [ 2  0  3  7 78  2  1  6  1  0]
 [ 1  0  4 10  2 79  2  1  1  0]
 [ 1  0  2  4  1  4 87  1  0  0]
 [ 2  0  0  3  6  5  0 84  0  0]
 [ 6  0  1  1  0  0  0  0 92  0]
 [ 1  2  0  2  0  0  0  0  1 94]]
```

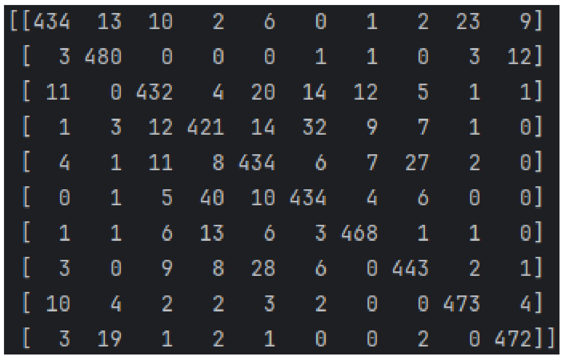


```
[[494   0   1   0   0   0   0   0   5   0]
 [  2 490   0   0   0   0   0   0   1   7]
 [  7   0 483   0   7   1   2   0   0   0]
 [  0   0   2 484   5   4   2   2   0   1]
 [  4   0   3   2 489   1   0   1   0   0]
 [  0   0   1   6   2 488   0   3   0   0]
 [  0   0   0   2   0   2 496   0   0   0]
 [  0   0   2   3   3   2   0 490   0   0]
 [  3   0   0   0   1   0   0   0 495   1]
 [  2   3   1   0   0   0   0   0   0 494]]
```

Reduced Layer Size MLP Model:

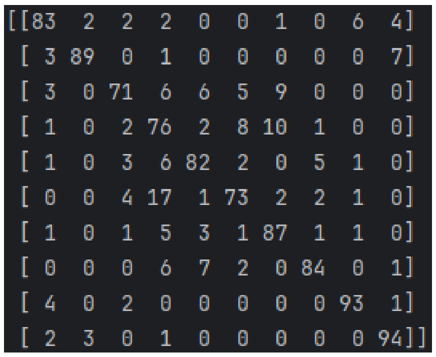


```
[[77  2  2  1  0  0  3  0 11  4]
 [ 2 92  0  1  0  0  0  0  1  4]
 [ 6  0 71  4  3  7  8  1  0  0]
 [ 1  0  4 72  2 12  7  2  0  0]
 [ 2  0  2  6 80  3  1  5  1  0]
 [ 0  0  5  8  1 82  2  1  1  0]
 [ 1  0  1  3  3  3 88  0  1  0]
 [ 0  0  0  5  6  1  1 87  0  0]
 [ 4  0  1  0  0  0  0  0 93  2]
 [ 1  7  0  2  0  0  0  0  0 90]]
```

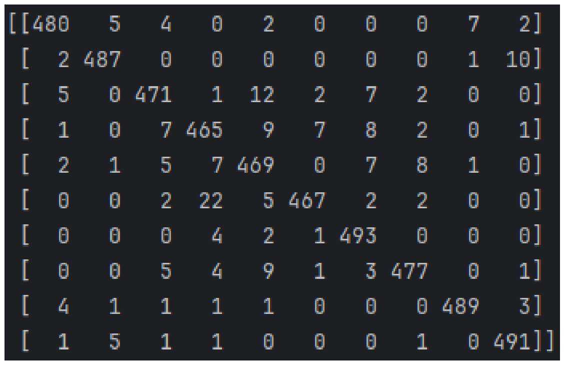


```
[[434  13  10   2   6   0   1   2  23   9]
 [  3 480   0   0   0   1   1   0   3  12]
 [ 11   0 432   4  20  14  12   5   1   1]
 [  1   3  12 421  14  32   9   7   1   0]
 [  4   1  11   8 434   6   7  27   2   0]
 [  0   1   5  40  10 434   4   6   0   0]
 [  1   1   6  13   6   3 468   1   1   0]
 [  3   0   9   8  28   6   0 443   2   1]
 [ 10   4   2   2   3   2   0   0 473   4]
 [  3  19   1   2   1   0   0   2   0 472]]
```

Increased Layer Size MLP Model:

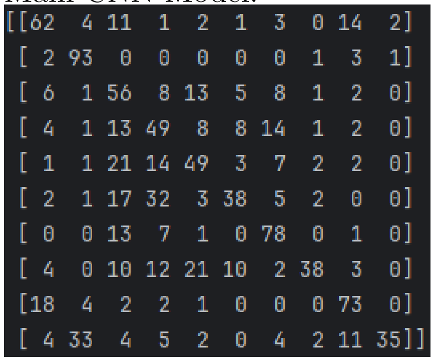


```
[[83  2  2  2  0  0  1  0  6  4]
 [ 3 89  0  1  0  0  0  0  0  7]
 [ 3  0 71  6  6  5  9  0  0  0]
 [ 1  0  2 76  2  8 10  1  0  0]
 [ 1  0  3  6 82  2  0  5  1  0]
 [ 0  0  4 17  1 73  2  2  1  0]
 [ 1  0  1  5  3  1 87  1  1  0]
 [ 0  0  0  6  7  2  0 84  0  1]
 [ 4  0  2  0  0  0  0  0 93  1]
 [ 2  3  0  1  0  0  0  0  0 94]]
```

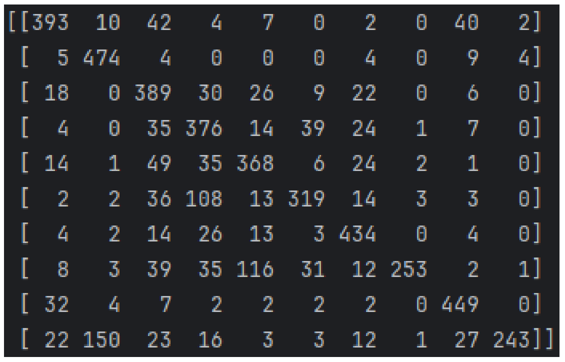


```
[[480   5   4   0   2   0   0   0   7   2]
 [  2 487   0   0   0   0   0   0   1  10]
 [  5   0 471   1  12   2   7   2   0   0]
 [  1   0   7 465   9   7   8   2   0   1]
 [  2   1   5   7 469   0   7   8   1   0]
 [  0   0   2  22   5 467   2   2   0   0]
 [  0   0   0   4   2   1 493   0   0   0]
 [  0   0   5   4   9   1   3 477   0   1]
 [  4   1   1   1   1   0   0   0 489   3]
 [  1   5   1   1   0   0   0   1   0 491]]
```

Main CNN Model:

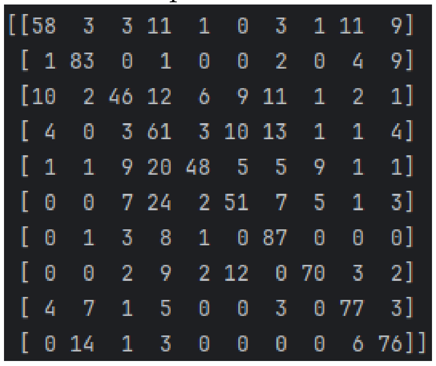


```
[[62  4 11  1  2  1  3  0 14  2]
 [ 2 93  0  0  0  0  0  1  3  1]
 [ 6  1 56  8 13  5  8  1  2  0]
 [ 4  1 13 49  8  8 14  1  2  0]
 [ 1  1 21 14 49  3  7  2  2  0]
 [ 2  1 17 32  3 38  5  2  0  0]
 [ 0  0 13  7  1  0 78  0  1  0]
 [ 4  0 10 12 21 10  2 38  3  0]
 [18  4  2  2  1  0  0  0 73  0]
 [ 4 33  4  5  2  0  4  2 11 35]]
```

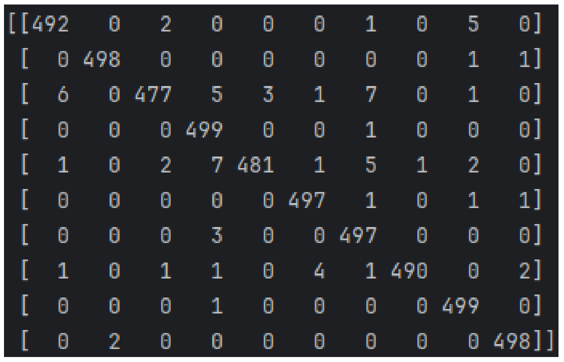


```
[[393  10  42   4   7   0   2   0  40   2]
 [  5 474   4   0   0   0   4   0   9   4]
 [ 18   0 389  30  26   9  22   0   6   0]
 [  4   0  35 376  14  39  24   1   7   0]
 [ 14   1  49  35 368   6  24   2   1   0]
 [  2   2  36 108  13 319  14   3   3   0]
 [  4   2  14  26  13   3 434   0   4   0]
 [  8   3  39  35 116  31  12 253   2   1]
 [ 32   4   7   2   2   2   2   0 449   0]
 [ 22 150  23  16   3   3  12   1  27 243]]
```

Reduced Depth CNN Model:

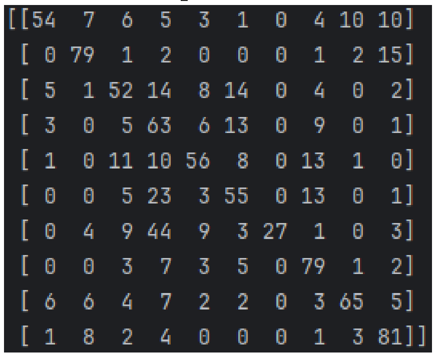


```
[[58  3  3 11  1  0  3  1 11  9]
 [ 1 83  0  1  0  0  2  0  4  9]
 [10  2 46 12  6  9 11  1  2  1]
 [ 4  0  3 61  3 10 13  1  1  4]
 [ 1  1  9 20 48  5  5  9  1  1]
 [ 0  0  7 24  2 51  7  5  1  3]
 [ 0  1  3  8  1  0 87  0  0  0]
 [ 0  0  2  9  2 12  0 70  3  2]
 [ 4  7  1  5  0  0  3  0 77  3]
 [ 0 14  1  3  0  0  0  0  6 76]]
```

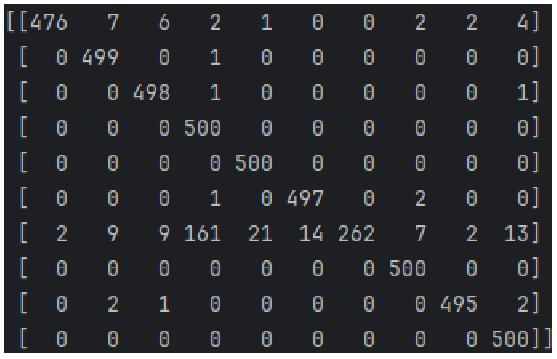


```
[[492   0   2   0   0   0   1   0   5   0]
 [  0 498   0   0   0   0   0   0   1   1]
 [  6   0 477   5   3   1   7   0   1   0]
 [  0   0   0 499   0   0   1   0   0   0]
 [  1   0   2   7 481   1   5   1   2   0]
 [  0   0   0   0   0 497   1   0   1   1]
 [  0   0   0   3   0   0 497   0   0   0]
 [  1   0   1   1   0   4   1 490   0   2]
 [  0   0   0   1   0   0   0   0 499   0]
 [  0   2   0   0   0   0   0   0   0 498]]
```

Increased Depth CNN Model:

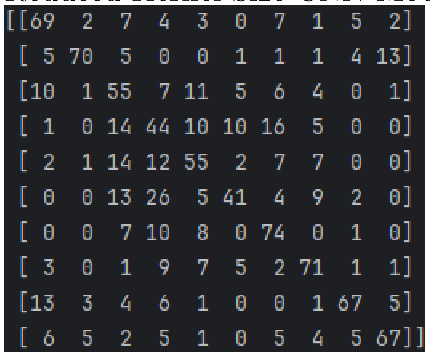


```
[[54  7  6  5  3  1  0  4 10 10]
 [ 0 79  1  2  0  0  0  1  2 15]
 [ 5  1 52 14  8 14  0  4  0  2]
 [ 3  0  5 63  6 13  0  9  0  1]
 [ 1  0 11 10 56  8  0 13  1  0]
 [ 0  0  5 23  3 55  0 13  0  1]
 [ 0  4  9 44  9  3 27  1  0  3]
 [ 0  0  3  7  3  5  0 79  1  2]
 [ 6  6  4  7  2  2  0  3 65  5]
 [ 1  8  2  4  0  0  0  1  3 81]]
```

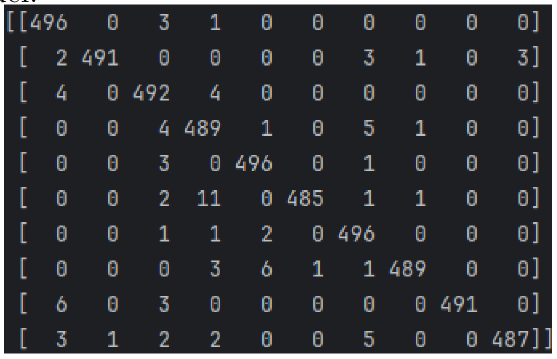


```
[[476   7   6   2   1   0   0   2   2   4]
 [  0 499   0   1   0   0   0   0   0   0]
 [  0   0 498   1   0   0   0   0   0   1]
 [  0   0   0 500   0   0   0   0   0   0]
 [  0   0   0   0 500   0   0   0   0   0]
 [  0   0   0   1   0 497   0   2   0   0]
 [  2   9   9 161  21  14 262   7   2  13]
 [  0   0   0   0   0   0   0 500   0   0]
 [  0   2   1   0   0   0   0   0 495   2]
 [  0   0   0   0   0   0   0   0   0 500]]
```

Reduced Kernel Size CNN Model:

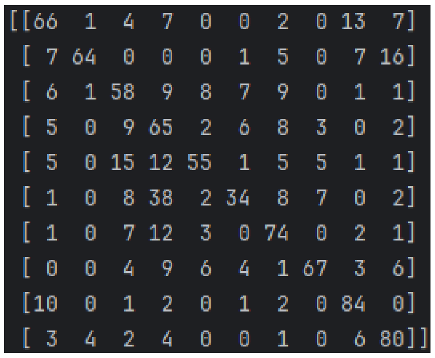


```
[[69  2  7  4  3  0  7  1  5  2]
 [ 5 70  5  0  0  1  1  1  4 13]
 [10  1 55  7 11  5  6  4  0  1]
 [ 1  0 14 44 10 10 16  5  0  0]
 [ 2  1 14 12 55  2  7  7  0  0]
 [ 0  0 13 26  5 41  4  9  2  0]
 [ 0  0  7 10  8  0 74  0  1  0]
 [ 3  0  1  9  7  5  2 71  1  1]
 [13  3  4  6  1  0  0  1 67  5]
 [ 6  5  2  5  1  0  5  4  5 67]]
```

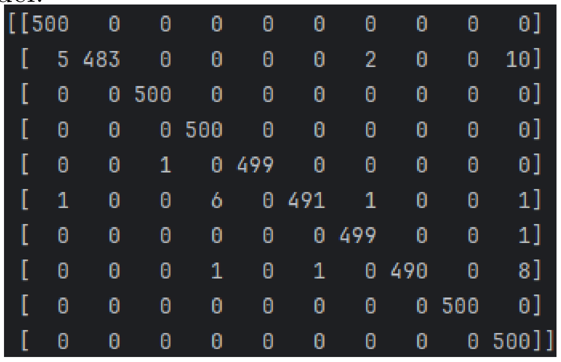


```
[[496   0   3   1   0   0   0   0   0   0]
 [  2 491   0   0   0   0   3   1   0   3]
 [  4   0 492   4   0   0   0   0   0   0]
 [  0   0   4 489   1   0   5   1   0   0]
 [  0   0   3   0 496   0   1   0   0   0]
 [  0   0   2  11   0 485   1   1   0   0]
 [  0   0   1   1   2   0 496   0   0   0]
 [  0   0   0   3   6   1   1 489   0   0]
 [  6   0   3   0   0   0   0   0 491   0]
 [  3   1   2   2   0   0   5   0   0 487]]
```

Increased Kernel Size CNN Model:



```
[[66  1  4  7  0  0  2  0 13  7]
 [ 7 64  0  0  0  1  5  0  7 16]
 [ 6  1 58  9  8  7  9  0  1  1]
 [ 5  0  9 65  2  6  8  3  0  2]
 [ 5  0 15 12 55  1  5  5  1  1]
 [ 1  0  8 38  2 34  8  7  0  2]
 [ 1  0  7 12  3  0 74  0  2  1]
 [ 0  0  4  9  6  4  1 67  3  6]
 [10  0  1  2  0  1  2  0 84  0]
 [ 3  4  2  4  0  0  1  0  6 80]]
```



```
[[500   0   0   0   0   0   0   0   0   0]
 [  5 483   0   0   0   0   2   0   0  10]
 [  0   0 500   0   0   0   0   0   0   0]
 [  0   0   0 500   0   0   0   0   0   0]
 [  0   0   1   0 499   0   0   0   0   0]
 [  1   0   0   6   0 491   1   0   0   1]
 [  0   0   0   0   0   0 499   0   0   1]
 [  0   0   0   1   0   1   0 490   0   8]
 [  0   0   0   0   0   0   0   0 500   0]
 [  0   0   0   0   0   0   0   0   0 500]]
```

## 2.4 Confusion-Heavy Classes

From the confusion matrices above, it is evident that the confusion-heavy classes are classes: 2, 3, 4, 5, 6 and 7.
Most of the misclassifications occur between these classes which can be explained by the classes they represent:

- Class 2: Bird
- Class 3: Cat
- Class 4: Deer
- Class 5: Dog
- Class 6: Frog
- Class 7: Horse

These are all animals and will all tend to have the same characteristics (four legs, head, mouth, etc.) so it makes sense that these were the highest source of confusion.

## 2.5 Well Recognized Classes

On the other hand, the well recognized classes are: 0, 1, 8, 9.
We can explain, once again, by the classes they represent:

- Class 0: Airplane
- Class 1: Automobile

- Class 8: Ship
- Class 9: Truck

Although these all fall under the category of transportation vehicles, they contain more distinctive features which help differentiate and characterize each one more clearly:

- Wings $\rightarrow$ Airplane
- Wheels and Small $\rightarrow$ Automobile
- On water $\rightarrow$ Ship
- Trailer and Rectangular $\rightarrow$ Truck

## 2.6 Depth Adjustment Effects

For the decision tree, changing the depth did not lead to significant changes in performance on unseen data and also did not show evidence of leading to overfitting.
The MLP model also did not lead to significant changes in performance on unseen data but showed much more prominent signs of overfitting due to the high increases in metrics on seen data.
The CNN model did have a significant increase in performance on unseen data for both decreased and increased depth. However, there is high evidence for overfitting given that the performance on seen data was near perfect.
Overall, the results show that increased depth did lead to increased accuracies but not significantly. Increasing depth by too high of a magnitude will certainly lead to overfitting since the model memorizes the seen data instead of forming generalizations on the data.

## 2.7 Layer and Kernel Size Adjustment Effects

Changing the layer size on the MLP did not have a significant effect on its recognition abilities for unseen data. However, reducing the layer size did lead to reductions in metrics on seen data which shows that overfitting is reduced and generalization is improved.
Both increasing and decreasing the kernel size of the CNN model increased the metrics on unseen data as well as on seen data. The key point is that the improvement in metrics on unseen data were of a significantly smaller magnitude than that of on the seen data. Metrics on seen data approached 100 percent which is characteristic of overfitting.

## 2.8 Summary of Primary Findings

Key Points:

- Metrics for the Naive Bayes, Decision Tree and MLP models were quite good with only minimal signs of overfitting. Metrics for unseen data were consistently higher but not excessively.
- For the CNN model, lower than expected metrics were obtained on unseen data while high metrics were obtained on seen data. This is characteristic of overfitting which was present across all variant models of CNN.
- Increasing depth was directly proportional to increased overfitting.

Best Performing Model:
Of all 16 models implemented in this project, the MLP model and its variants were the best performing models on unseen data.
More specifically, of the MLP model variants, the main MLP model performed the best with an accuracy of **84.20** percent on unseen data.
Compared to the Naive Bayes and Decision Tree models, it's understandable that the MLP models are better suited to CIFAR10 image recognition due to its increased complexity and ability to capture non-linear patterns in the data.
However, the CNN is more complex and even better at recognizing these non-linear patterns than the MLP. With that being said, it is expected that the CNN model would outperform the MLP model which the results didn't show in this case.
As we have previously seen, the results show that the CNN models were overfitting to the training data which would explain its underperformance on the testing data.