COMP 472 - Artificial Intelligence

Project: Image Classification on CIFAR-10

Jamie Low
40314444
COMP 472
Kefaya Qaddoum
November 24, 2025

3. Naive Bayes

3.1) Model Architecture

Two versions of Gaussian Naive Bayes were built and tested:

3.1.1) Custom Implementation (NumPy Only)

The model follows the standard Naive Bayes assumption that features are independent and computes class posteriors using: P(class | x) ∝ P(class) × ∏ P(feature_i | class)
For each of the 10 classes, the model learns:

- The mean of each feature (50 values)
- The variance of each feature (50 values)
- The prior probability of the class based on its frequency in the training set

During prediction, it computes the Gaussian likelihood for each feature and selects the class with the highest resulting posterior.

3.1.2) Scikit-learn Implementation

A second model was built using GaussianNB() from scikit-learn.

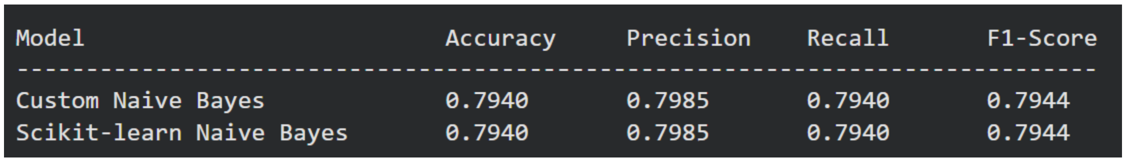
3.2) Training Methodology

- Input features: 50-dimensional vectors (ResNet-18 embeddings reduced with PCA)
- Training set: 5,000 samples (500 per class)
- Test set: 1,000 samples (100 per class)
- Hyperparameters: None

Training is just one pass over the data to compute the class means, variances, and priors.

3.3) Evaluation Results

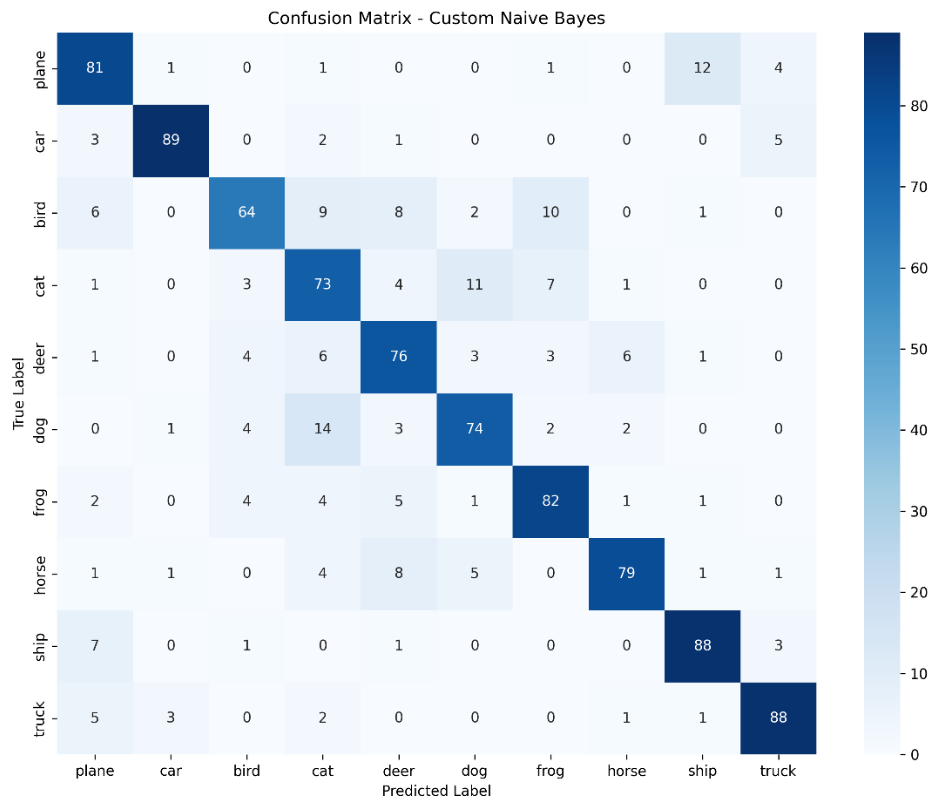
Performance Metrics: accuracy, precision, recall, and F1-score.

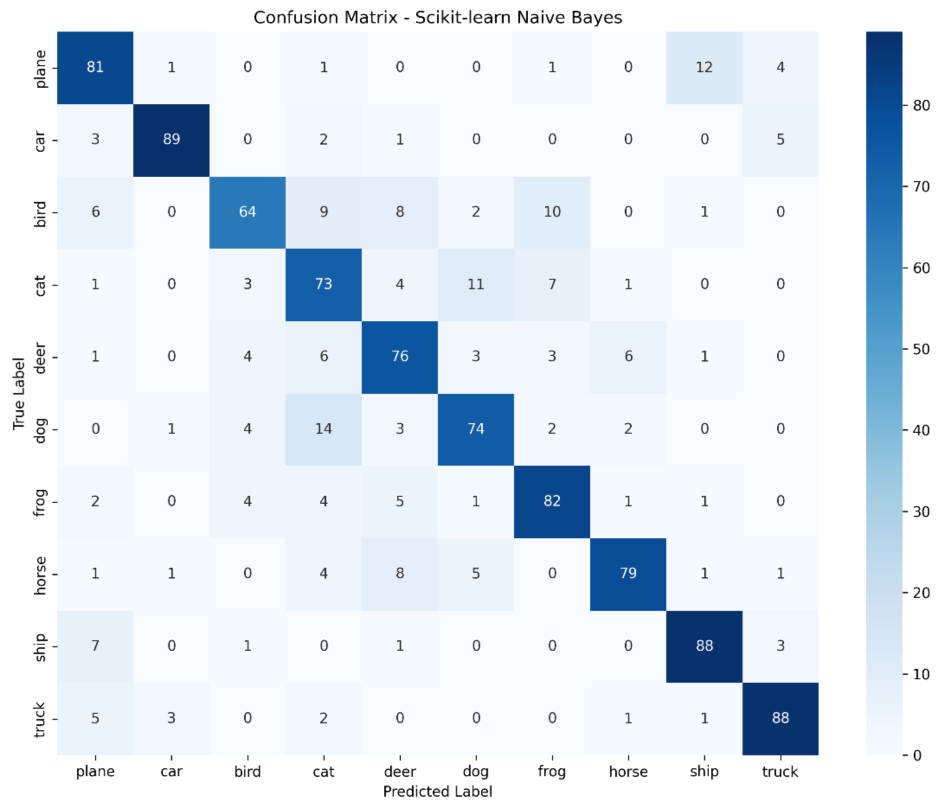


```
Model                          Accuracy     Precision    Recall       F1-Score
-----------------------------------------------------------------------------
Custom Naive Bayes             0.7940       0.7985       0.7940       0.7944
Scikit-learn Naive Bayes       0.7940       0.7985       0.7940       0.7944
```

Both versions of the model reached the same accuracy: 79.40%, the custom implementation matches scikit-learn's behavior.

3.4) Confusion Matrix Analysis





Because both models work identically, they produce the same confusion matrix.
Some key observations:

- Car had the highest accuracy (89%). This makes sense because it's visually very distinct compared to the mostly animal-based classes.
- Bird had the lowest accuracy (64%).
- Cat vs Dog: These two classes frequently mix. For example, Dog reached 74% accuracy, with 14 samples misclassified as Cat. Their similar shapes and textures likely contribute to this confusion, especially after dimensionality reduction.

3.5) Summary

The final accuracy of 79.40% shows that the ResNet-18 features, combined with PCA, produce a compact 50-dimensional representation that still separates the classes well. The Gaussian assumption fits these features reasonably, which is reflected in the balanced precision and recall across classes.

4. Decision Tree

4.1) Model Architectures

4.1.1) Base Model: Custom Decision Tree

A custom Decision Tree classifier was implemented entirely with NumPy. The tree is built recursively: at each node, the algorithm evaluates all features and all unique feature values as potential split points, selecting the threshold that produces the lowest weighted Gini impurity. After choosing the best split, the model grows the left and right branches until it reaches the maximum depth or another stopping condition, such as a pure node or too few samples.

The custom implementation supports adjustable tree depth, and each leaf node predicts the majority class of the samples reaching it.

4.1.2) Model Variants

To study the effect of depth, several versions of the tree were trained with different maximum depths: max_depth ∈ {5, 10, 20, 30, 50}

4.1.3) Scikit-learn Implementation

A scikit-learn DecisionTreeClassifier was also trained, using max_depth = 50, to compare against the custom implementation.

4.2) Training Methodology

Training a decision tree does not require epochs or gradient-based optimization. Instead, the tree is built using a greedy splitting strategy:

- At each node, evaluate all possible splits
- Select the one with the lowest Gini impurity
- Recursively expand the resulting child nodes

Key hyperparameters:

- max_depth: {5, 10, 20, 30, 50}
- min_samples_split = 2
- criterion = Gini impurity

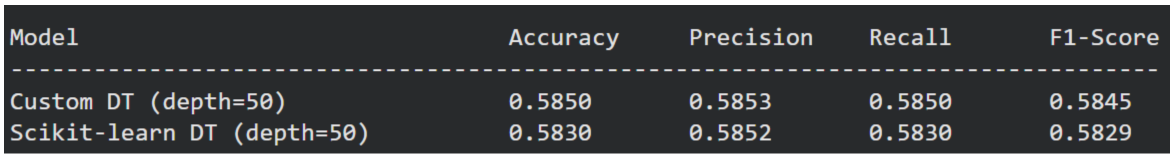
Stopping conditions:

- Maximum depth reached
- Node is pure (only one class present)
- Not enough samples to split
- No split improves weighted impurity

Because the tree relies on evaluating splits directly, no optimizers or iterative updates are used.

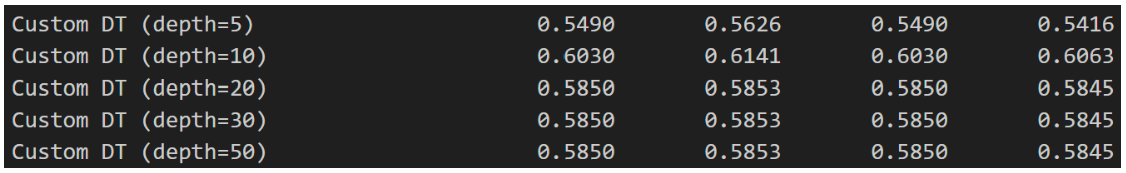
4.3) Evaluation of Various Depths

Performance was evaluated using accuracy, precision, recall, and F1-score.

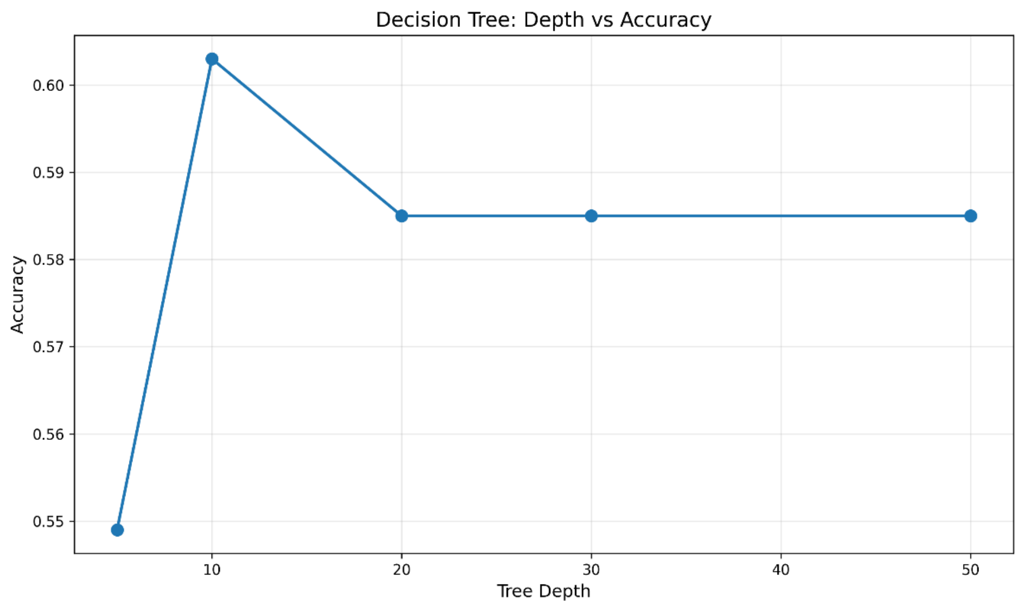


```
Model                          Accuracy    Precision   Recall      F1-Score
---------------------------------------------------------------------------
Custom DT (depth=50)           0.5850      0.5853      0.5850      0.5845
Scikit-learn DT (depth=50)     0.5830      0.5852      0.5830      0.5829
```

Both the custom tree and the scikit-learn version performed similarly. The custom implementation slightly edged ahead in accuracy and recall.



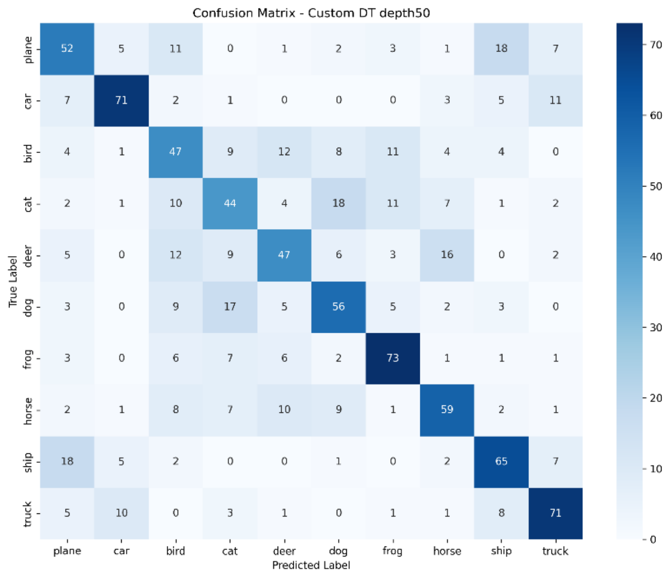
```
Custom DT (depth=5)                  0.5490      0.5626      0.5490      0.5416
Custom DT (depth=10)                 0.6030      0.6141      0.6030      0.6063
Custom DT (depth=20)                 0.5850      0.5853      0.5850      0.5845
Custom DT (depth=30)                 0.5850      0.5853      0.5850      0.5845
Custom DT (depth=50)                 0.5850      0.5853      0.5850      0.5845
```



Across the depth experiments:

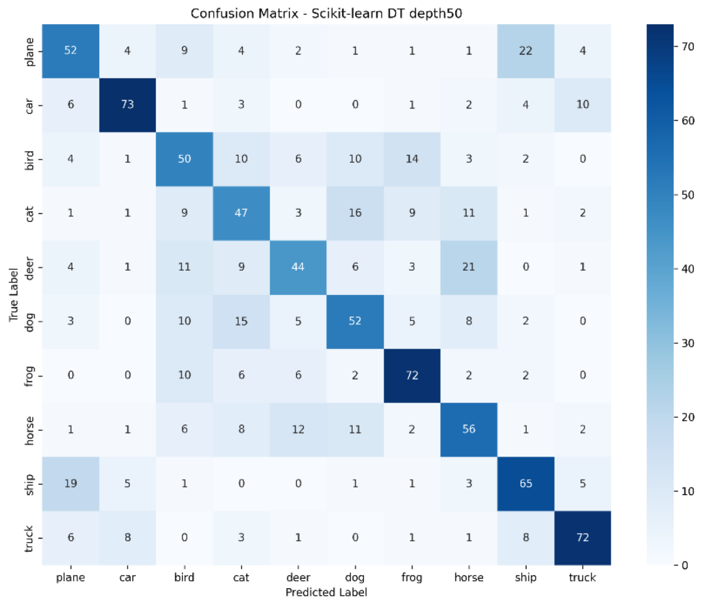
- Depth 10 achieved the best accuracy (60%)
- Depth 5 underfit, producing weaker performance
- Deeper trees (20–50) plateaued or slightly dropped, suggesting overfitting to the training set

4.4) Confusion Matrices



Custom Decision Tree:

- Frog had the highest accuracy (73%). Its distinct green coloring and overall shape may make it easier for the model to separate from other classes.
- Cat had the lowest accuracy (44%). Like Naive Bayes, Cat and Dog frequently confused each other, which likely explains the lower performance on these two classes.



Scikit-learn Decision Tree:

- Car scored the highest accuracy (73%). As a non-animal class, its strong visual features make it easier to identify.
- Deer had the lowest accuracy (44%). Many Deer images were misclassified as Horse (21 cases), likely due to their similar body shapes and proportions.

4.5) Summary

- Several classes were consistently confused, likely due to overlapping visual characteristics.
- Very deep trees (30–50) tended to overfit, leading to unstable predictions.
- A moderate depth (around 10) offered the best generalization and overall accuracy.

5. Multi-Layer Perceptron (MLP)

5.1) Base MLP Architecture

The main neural network used in this experiment is a three-layer MLP trained on the 50-dimensional PCA features extracted from CIFAR-10. The model follows the architecture:

- Linear (50 → 512) → ReLU
- Linear (512 → 512) → BatchNorm(512) → ReLU
- Linear (512 → 10)

Batch normalization is applied after the second hidden layer to stabilize training and help the model converge more smoothly. The final layer outputs logits for the 10 CIFAR-10 classes.

5.1.1) Model Variants

Shallow MLP (2 layers)

- Linear (50 → 512) → ReLU
- Linear (512 → 10)

  This removes one hidden layer, reducing model depth and capacity.

Deep MLP (4 layers)

- Linear (50 → 512) → BatchNorm → ReLU
- Linear (512 → 512) → BatchNorm → ReLU
- Linear (512 → 512) → BatchNorm → ReLU
- Linear (512 → 10)

  An extra hidden layer increases the depth and parameter count.

Small MLP (hidden size = 256)

- 50 → 256
- 256 → 256

  This variant tests how reducing the hidden dimension affects performance.

Large MLP (hidden size = 1024)

- 50 → 1024

- 1024 → 1024
  This significantly increases the model’s width to check whether larger layers improve results.

5.2) Training Methodology

All MLP models were trained under the same setup to keep comparisons consistent:

- Loss: CrossEntropyLoss
- Optimizer: SGD (momentum = 0.9)
- Learning rate: 0.01
- Batch size: 64
- Epochs: 50
- Input: 50-dimensional PCA features

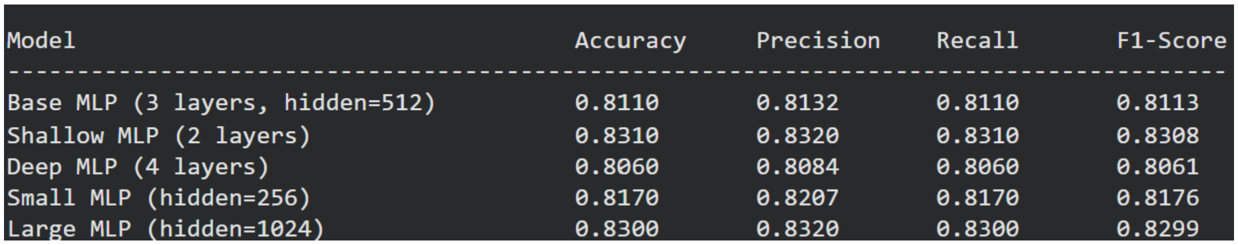
Training used standard mini-batch gradient descent:

1. Forward pass
2. Compute cross-entropy loss
3. Backpropagation
4. Parameter updates using SGD + momentum

Batch normalization layers maintained running statistics during training. No dropout or weight decay was included.

5.3) Evaluation

Metrics: accuracy, precision, recall, and F1-score.



```
Model                                Accuracy    Precision   Recall      F1-Score
------------------------------------------------------------------------------
Base MLP (3 layers, hidden=512)      0.8110      0.8132      0.8110      0.8113
Shallow MLP (2 layers)               0.8310      0.8320      0.8310      0.8308
Deep MLP (4 layers)                  0.8060      0.8084      0.8060      0.8061
Small MLP (hidden=256)               0.8170      0.8207      0.8170      0.8176
Large MLP (hidden=1024)              0.8300      0.8320      0.8300      0.8299
```

Insights and Comparisons

Shallow vs. Base vs. Deep

- The Shallow MLP (2 layers) achieved the best accuracy (0.8310).
- The Deep MLP (4 layers) performed worse than the base model.

This suggests:

- Increasing depth didn’t help the model generalize better.
- The deeper network likely overfit the compact 50-D PCA features.
- Once the features are already low-dimensional and informative, a shallow network is often enough.

Hidden Layer Size: 256 vs 512 vs 1024

- The Small MLP (256 units) performed reasonably well but slightly below the larger models (0.8170).

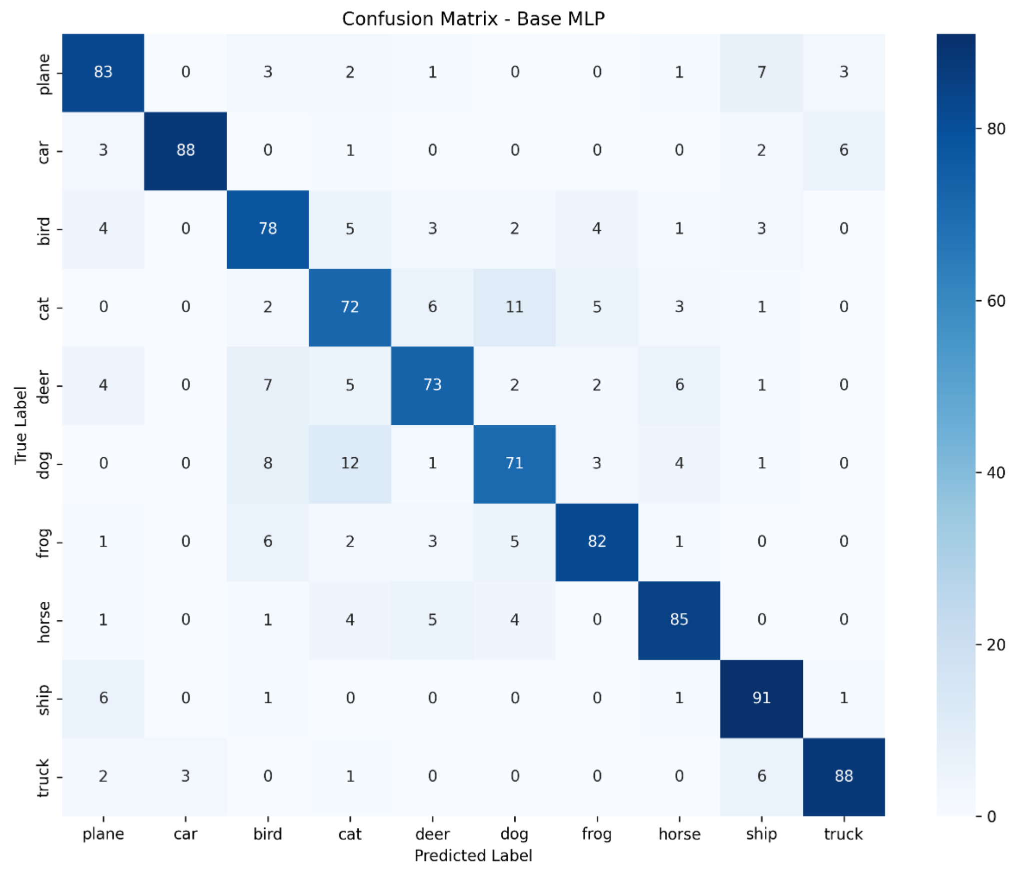
- The Large MLP (1024 units) nearly matched the shallow model (0.8300), indicating that:
    - Wider layers can capture more complex relationships
    - But the improvement becomes marginal after a certain point

Overall, hidden sizes between 512–1024 appear to work best for this feature set.

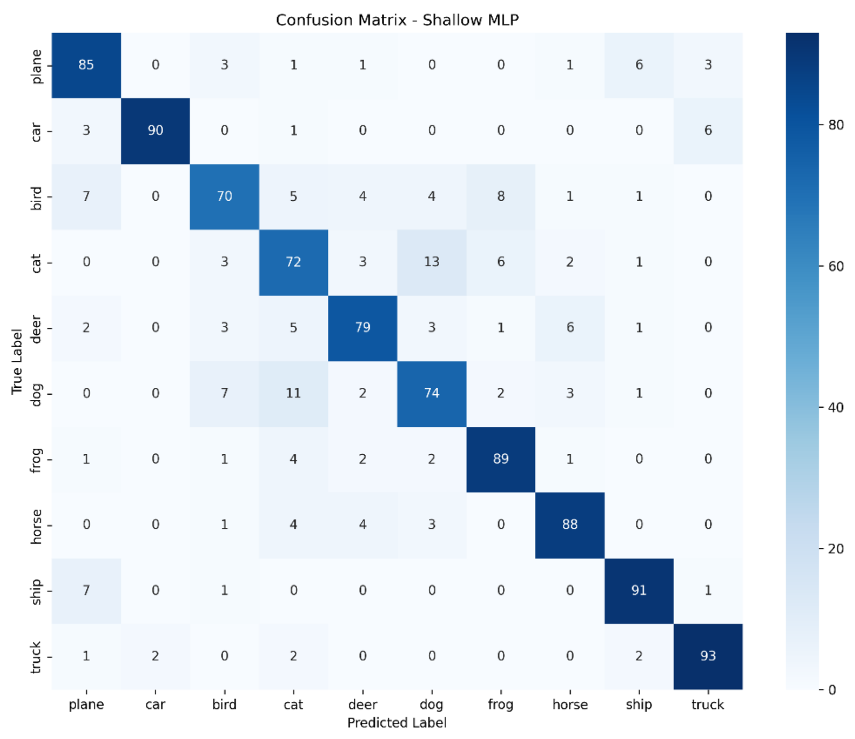
5.4) Confusion Matrices Analysis

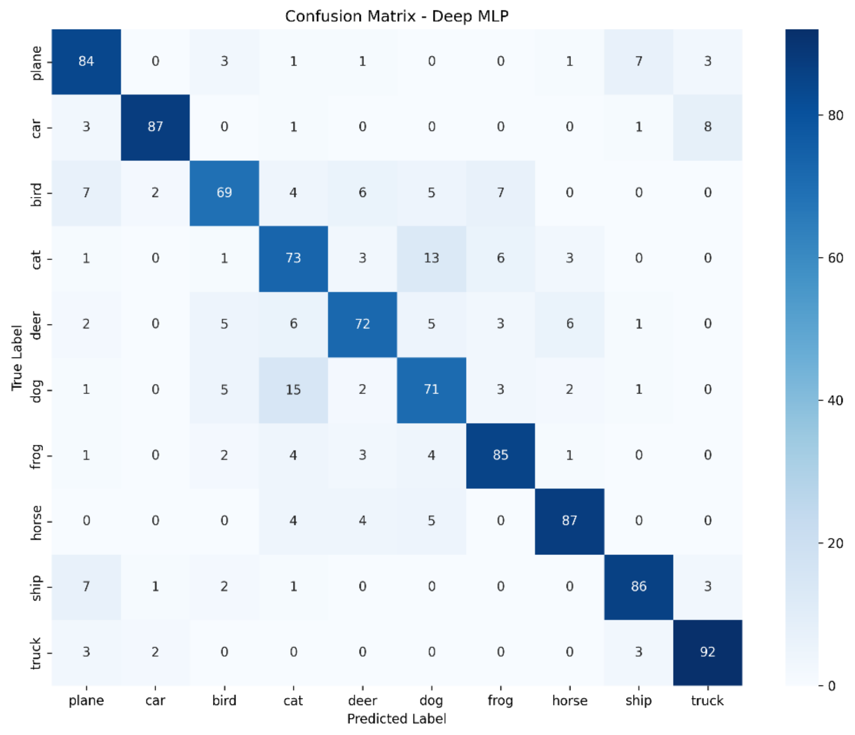
Across all MLP variants, several patterns appeared consistently:

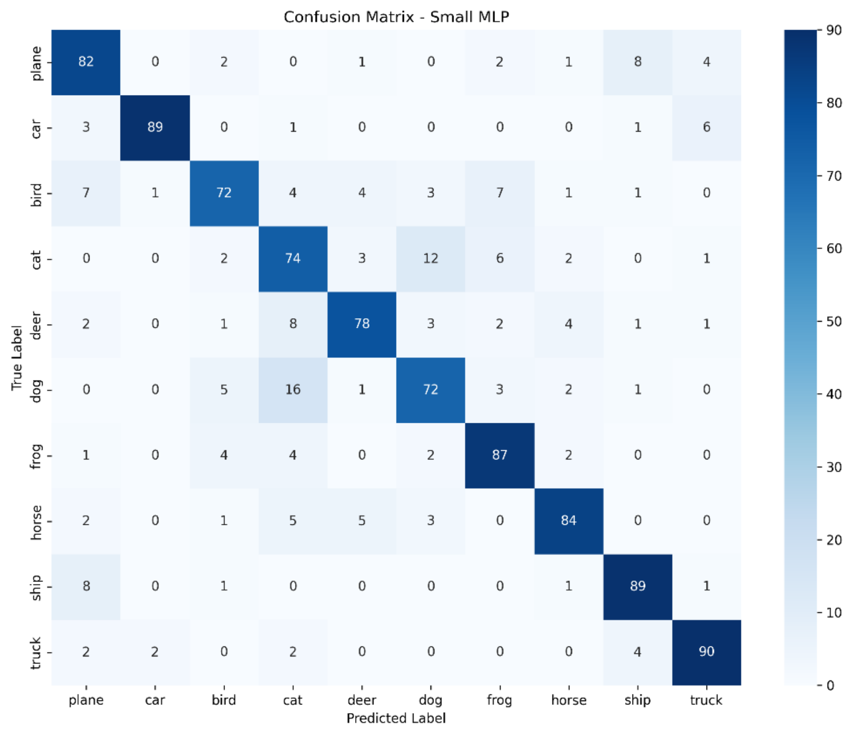
- Cat vs Dog: These two classes frequently confused each other, likely due to similar textures and shapes once reduced to 50-D PCA features.
- Best-recognized classes: Car, Ship, and Truck achieved the highest accuracies across all models. This is likely because these categories have strong, consistent geometric structures that remain recognizable after PCA compression. Their intra-class variation is lower, and many samples contain predictable backgrounds (like roads for vehicles, water for ships), making their feature representations more compact and easier for the MLP to separate compared to more variable animal classes.

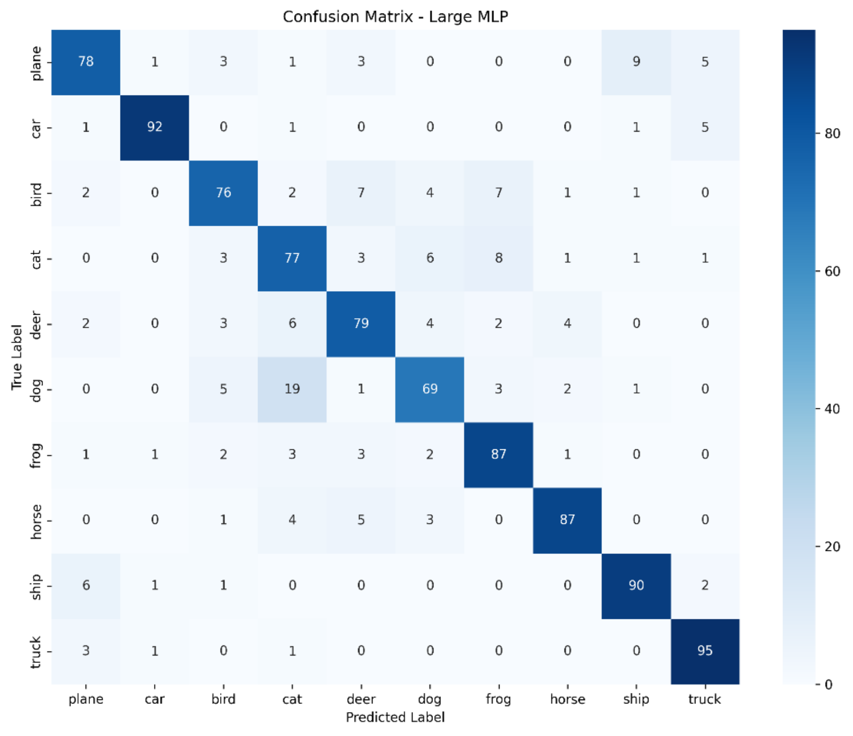


True Label \ Predicted Label	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	83	0	3	2	1	0	0	1	7	3
car	3	88	0	1	0	0	0	0	2	6
bird	4	0	78	5	3	2	4	1	3	0
cat	0	0	2	72	6	11	5	3	1	0
deer	4	0	7	5	73	2	2	6	1	0
dog	0	0	8	12	1	71	3	4	1	0
frog	1	0	6	2	3	5	82	1	0	0
horse	1	0	1	4	5	4	0	85	0	0
ship	6	0	1	0	0	0	0	1	91	1
truck	2	3	0	1	0	0	0	0	6	88


Confusion Matrix - Shallow MLP
True Label
Predicted Label
plane car bird cat deer dog frog horse ship truck
plane: 85 0 3 1 1 0 0 1 6 3
car: 3 90 0 1 0 0 0 0 0 6
bird: 7 0 70 5 4 4 8 1 1 0
cat: 0 0 3 72 3 13 6 2 1 0
deer: 2 0 3 5 79 3 1 6 1 0
dog: 0 0 7 11 2 74 2 3 1 0
frog: 1 0 1 4 2 2 89 1 0 0
horse: 0 0 1 4 4 3 0 88 0 0
ship: 7 0 1 0 0 0 0 0 91 1
truck: 1 2 0 2 0 0 0 0 2 93
0 20 40 60 80


Confusion Matrix - Deep MLP
True Label
Predicted Label
plane car bird cat deer dog frog horse ship truck
plane: 84 0 3 1 1 0 0 1 7 3
car: 3 87 0 1 0 0 0 0 1 8
bird: 7 2 69 4 6 5 7 0 0 0
cat: 1 0 1 73 3 13 6 3 0 0
deer: 2 0 5 6 72 5 3 6 1 0
dog: 1 0 5 15 2 71 3 2 1 0
frog: 1 0 2 4 3 4 85 1 0 0
horse: 0 0 0 4 4 5 0 87 0 0
ship: 7 1 2 1 0 0 0 0 86 3
truck: 3 2 0 0 0 0 0 0 3 92
0 20 40 60 80





5.5) Summary

Effect of Depth

- Adding more layers increased complexity but didn't improve performance.

- The deep MLP overfitted the compressed 50-D inputs.
- Shallower networks benefited from:
    - Fewer parameters
    - Faster training
    - Lower overfitting risk

Since the input features were already heavily compressed, additional depth didn't offer meaningful gains.

Effect of Hidden Layer Size

- Smaller hidden layers limited expressiveness, slightly reducing accuracy.
- Very large layers (1024) improved capacity but came with higher compute cost and only modest improvement.
- Increasing width helps up to a point, but the returns diminish with low-dimensional input.

6. Convolutional Neural Network (CNN) – VGG11

6.1) Model Architectures and Training

Base Model: VGG11

The main CNN used in this experiment is a VGG11 network composed of eight 3×3 convolutional layers with batch normalization, ReLU activations, and max-pooling after each block. The feature extractor is followed by three fully connected layers (4096 → 4096 → 10), with dropout (0.5) applied to the first two layers to reduce overfitting. The model is trained directly on the 32×32 RGB CIFAR-10 images.

Variants:

- Shallow VGG – Reduces depth from eight convolutional layers to six to examine how fewer layers affect generalization.
- Large-Kernel VGG – Replaces early 3×3 kernels with 5×5 kernels to explore how larger receptive fields trade off against spatial detail on small images.

6.2) Training Methodology

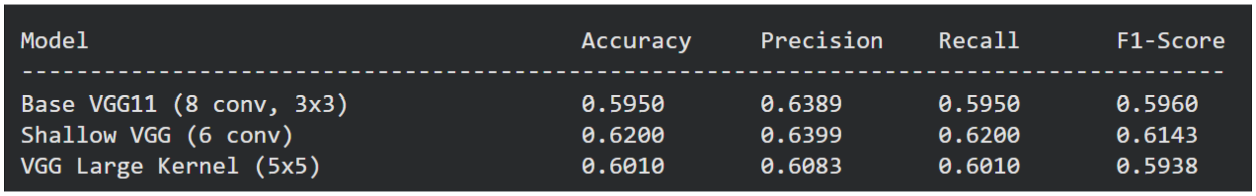
All CNN models were trained using the same pipeline:

- Optimizer: SGD + momentum (0.9)
- Loss: CrossEntropyLoss
- Epochs: 20
- Learning Rate: 0.01
- Batch Size: 64

Images were normalized using the CIFAR-10 dataset statistics. Training followed standard forward/backward passes with gradient updates.

6.3) Evaluation

Performance was assessed using accuracy, precision, recall, and F1-score for all models.



```
Model                             Accuracy    Precision    Recall      F1-Score
-------------------------------------------------------------------------------------
Base VGG11 (8 conv, 3x3)          0.5950      0.6389       0.5950      0.5960
Shallow VGG (6 conv)              0.6200      0.6399       0.6200      0.6143
VGG Large Kernel (5x5)            0.6010      0.6083       0.6010      0.5938
```

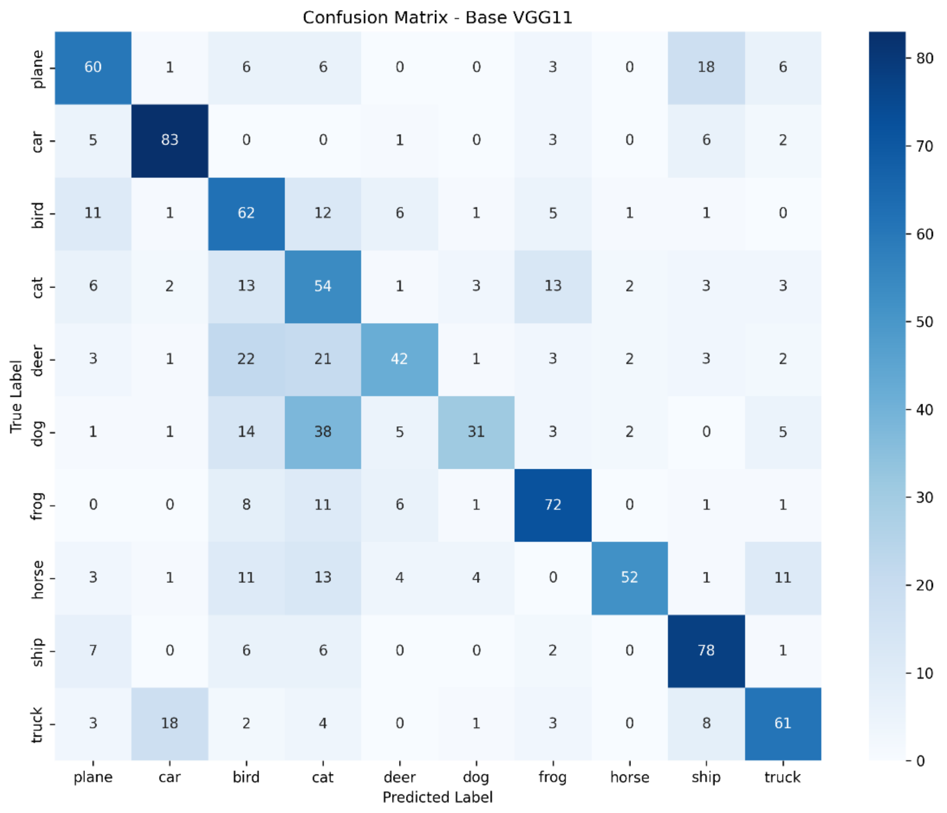
Observations:

- Shallow VGG (6 conv layers) achieved the highest accuracy (62.0%), outperforming the base VGG11.
  → This suggests the deeper model overfitted the relatively small CIFAR-10 dataset.
- Large-Kernel VGG (5×5 kernels) achieved 60.1% accuracy. This is slightly higher than the base model but still lower than the shallow variant.
  → Larger kernels capture broader patterns but reduce fine-grained detail, which can be harmful on 32×32 images.

Depth & Kernel Insights

- Depth: More layers increase capacity but also overfitting risk when data is limited.
- Kernel Size: Larger kernels enlarge the receptive field but can blur spatial details, which matters for small-scale images like CIFAR-10.

6.4) Confusion Matrix Analysis

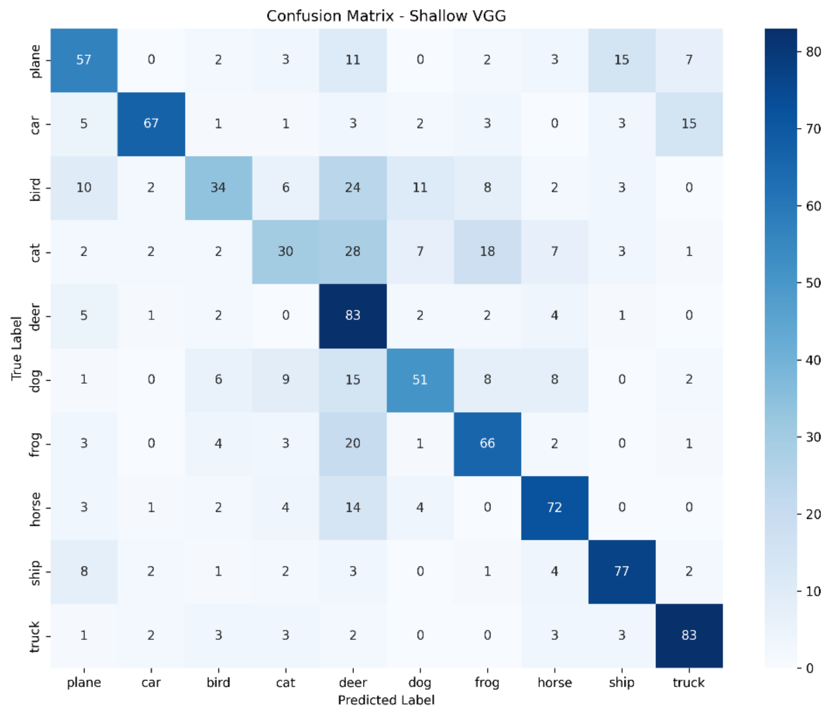


True \ Predicted	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	60	1	6	6	0	0	3	0	18	6
car	5	83	0	0	1	0	3	0	6	2
bird	11	1	62	12	6	1	5	1	1	0
cat	6	2	13	54	1	3	13	2	3	3
deer	3	1	22	21	42	1	3	2	3	2
dog	1	1	14	38	5	31	3	2	0	5
frog	0	0	8	11	6	1	72	0	1	1
horse	3	1	11	13	4	4	0	52	1	11
ship	7	0	6	6	0	0	2	0	78	1
truck	3	18	2	4	0	1	3	0	8	61

Base VGG11:

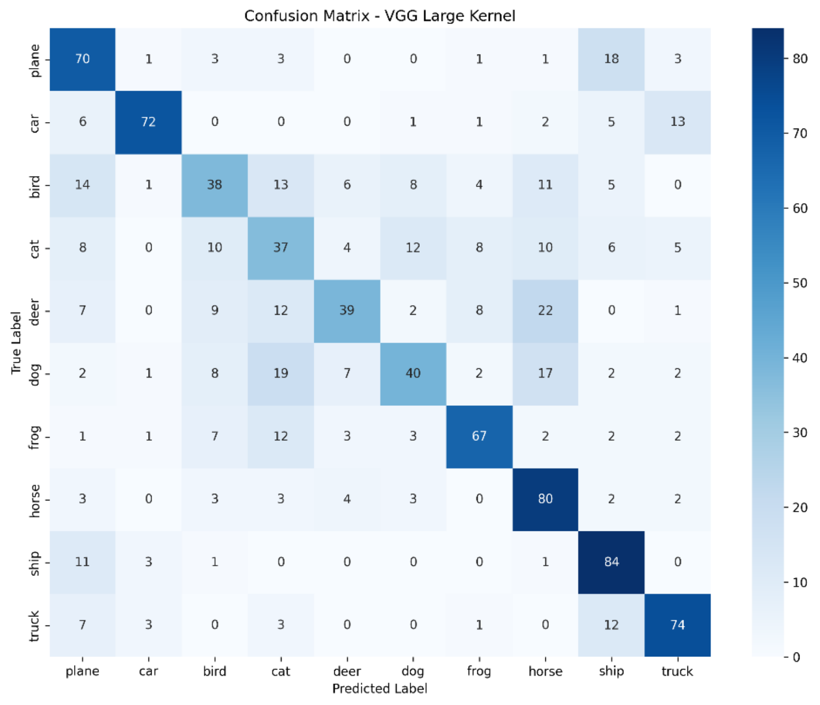
- Best-recognized: Car (83% accuracy).

- Worst-recognized: Dog (31% accuracy).
- Cat vs Dog confusion: High misclassification due to similar textures, shapes, and backgrounds.



Shallow VGG:

- Deer achieved the highest accuracy (83%), though several other classes were misclassified as deer. Perhaps the model was biased towards Deer class.



Large-Kernel VGG:

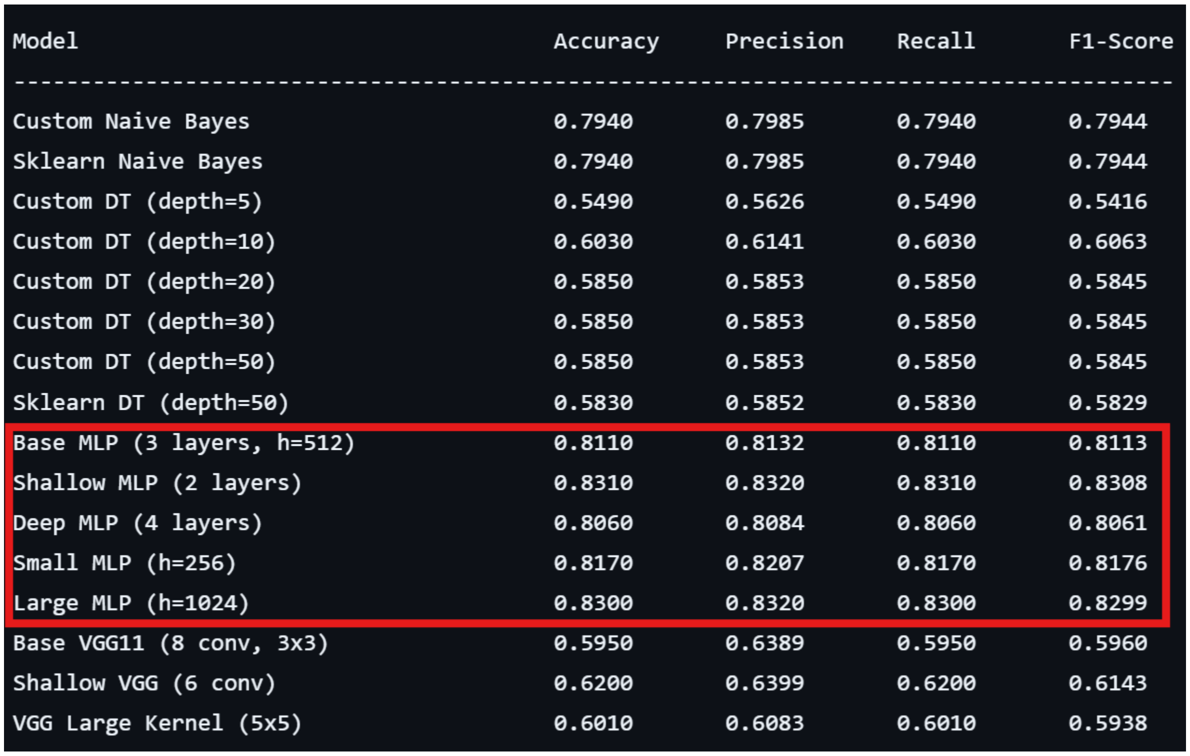
- Animal classes performed significantly worse than object classes (Plane, Car, Ship, Truck) → Objects have more rigid, consistent shapes, while animals show wide variation in pose, breed, color, and shape.

6.5) Summary

- The Shallow VGG variant delivered the best performance, indicating that moderately deep networks may generalize better than very deep ones on CIFAR-10.
- Increasing kernel size offered limited improvements, showing that receptive fields must match the scale of the input.
- Overall, VGG-style CNNs perform well on small images, but both depth and kernel design must be carefully balanced to avoid overfitting and preserve spatial detail.

Project Summary

Looking back at the results, the MLP turned out to be the most reliable classifier overall. Despite using compressed PCA inputs, the shallow MLP was able to learn stable patterns and produced the highest accuracy among all models; Compared to the other models, it handled the dataset's patterns more reliably and produced the most consistent results.



```
Model                                   Accuracy     Precision    Recall       F1-Score
------------------------------------------------------------------------------------------
Custom Naive Bayes                      0.7940       0.7985       0.7940       0.7944
Sklearn Naive Bayes                     0.7940       0.7985       0.7940       0.7944
Custom DT (depth=5)                     0.5490       0.5626       0.5490       0.5416
Custom DT (depth=10)                    0.6030       0.6141       0.6030       0.6063
Custom DT (depth=20)                    0.5850       0.5853       0.5850       0.5845
Custom DT (depth=30)                    0.5850       0.5853       0.5850       0.5845
Custom DT (depth=50)                    0.5850       0.5853       0.5850       0.5845
Sklearn DT (depth=50)                   0.5830       0.5852       0.5830       0.5829
Base MLP (3 layers, h=512)              0.8110       0.8132       0.8110       0.8113
Shallow MLP (2 layers)                  0.8310       0.8320       0.8310       0.8308
Deep MLP (4 layers)                     0.8060       0.8084       0.8060       0.8061
Small MLP (h=256)                       0.8170       0.8207       0.8170       0.8176
Large MLP (h=1024)                      0.8300       0.8320       0.8300       0.8299
Base VGG11 (8 conv, 3x3)                0.5950       0.6389       0.5950       0.5960
Shallow VGG (6 conv)                    0.6200       0.6399       0.6200       0.6143
VGG Large Kernel (5x5)                  0.6010       0.6083       0.6010       0.5938
```