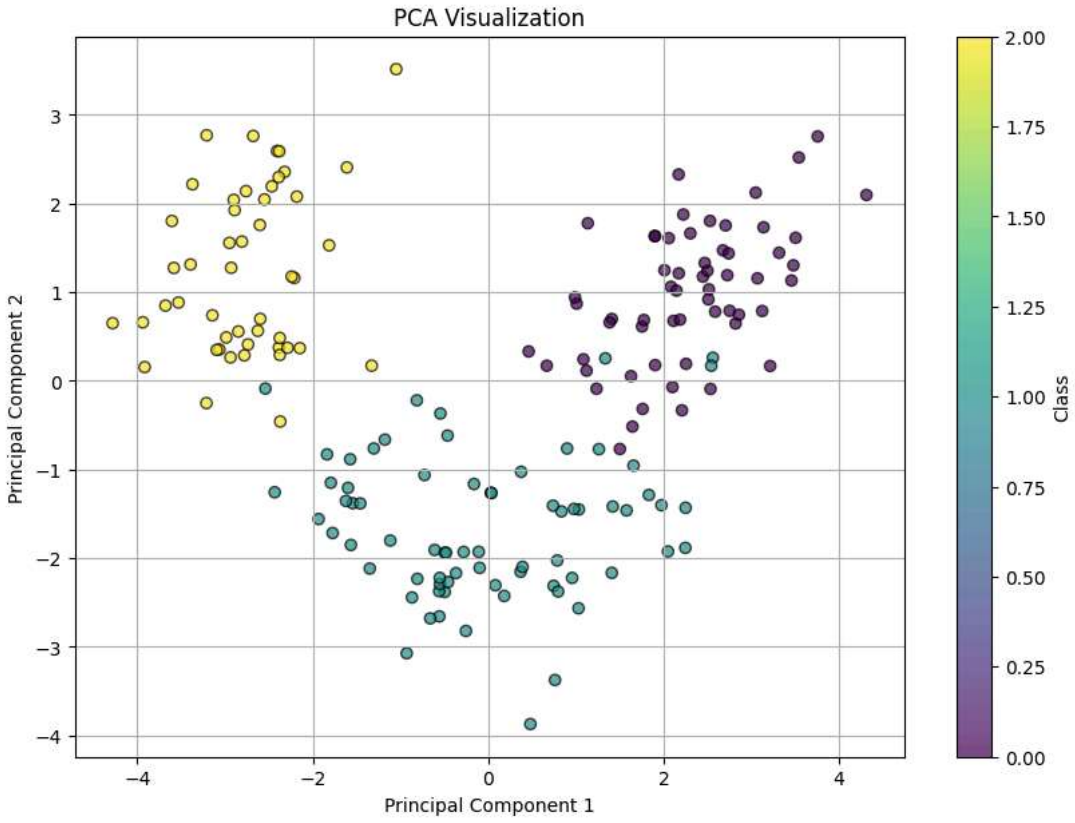
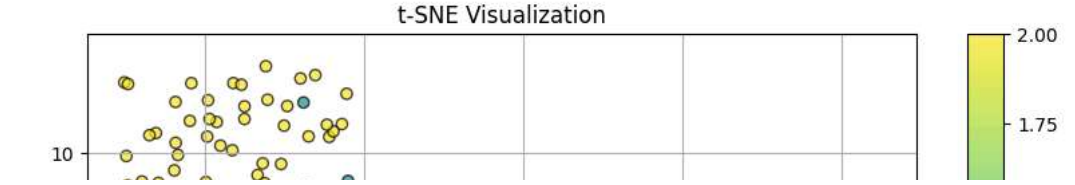
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Loading the Wine dataset
wine_data = load_wine()
X = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)  # Features
y = wine_data.target  # Target classes
X
y

# Check for missing values and descriptive statistics
print("Missing values per column:\n", X.isnull().sum())
print("\nDescriptive statistics:\n", X.describe())

# Normalize the data to ensure all features are on the same scale
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
X_normalized

# Dimensionality reduction using PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X_normalized)
pca_result

# Visualizing PCA results
plt.figure(figsize=(10, 7))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=y, cmap='viridis', alpha=0.7, edgecolor='k')
plt.title('PCA Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Class')
plt.grid(True)
plt.show()

# Dimensionality reduction using t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=0)
tsne_result = tsne.fit_transform(X_normalized)

# Visualizng t-SNE results
plt.figure(figsize=(10, 7))
plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=y, cmap='viridis', alpha=0.7, edgecolor='k')
plt.title('t-SNE Visualization')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(label='Class')
plt.grid(True)
plt.show()

# Import necessary libraries for second part of the project
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.3, random_state=42)

# Define a function to evaluate and display results for each model
def evaluate_model(model, model_name):
    # Train the model
    model.fit(X_train, y_train)
    # Predict on the test set
    y_pred = model.predict(X_test)
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Results for {model_name}:\n")
```

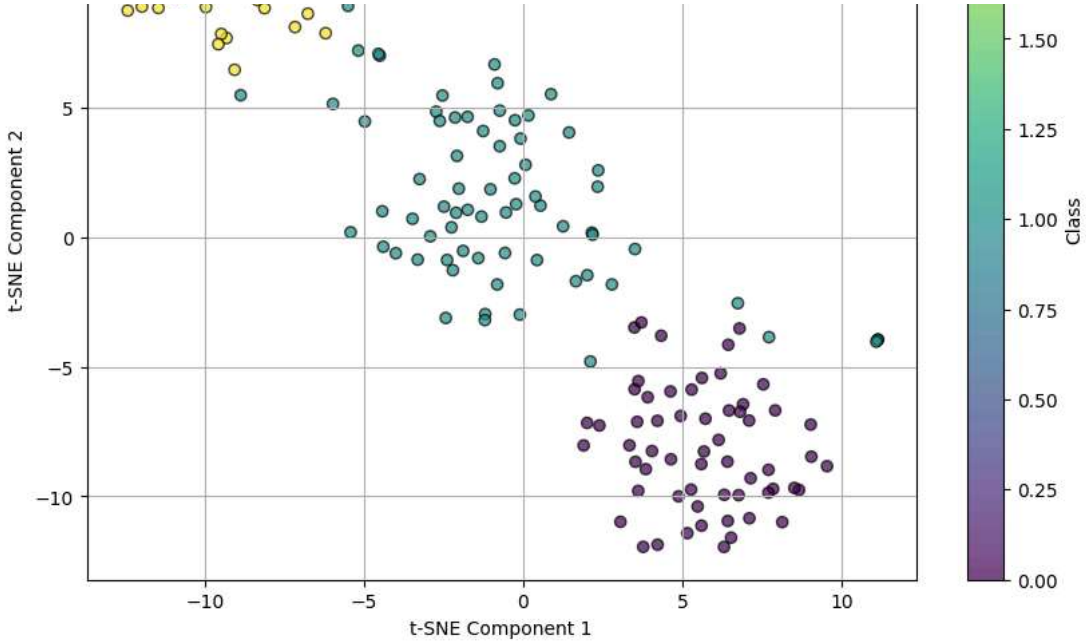
```
    print(f"Accuracy: {accuracy:.2f}\n")
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("------------------------------\n")

# 1. Distances
# k-Nearest Neighbors (kNN)
knn = KNeighborsClassifier(n_neighbors=5)
evaluate_model(knn, "k-Nearest Neighbors (kNN)")

# Nearest Centroid Classifier
from sklearn.neighbors import NearestCentroid
centroid = NearestCentroid()
evaluate_model(centroid, "Centroid Classifier")

# 2. Probabilistic Models
# Gaussian Naive Bayes (GNB)
gnb = GaussianNB()
evaluate_model(gnb, "Gaussian Naive Bayes (GNB)")

# Linear Discriminant Analysis (LDA)
lda = LinearDiscriminantAnalysis()
evaluate_model(lda, "Linear Discriminant Analysis (LDA)")

# 3. Rule-Based Models
# Decision Trees (DTs)
decision_tree = DecisionTreeClassifier(random_state=42)
evaluate_model(decision_tree, "Decision Tree")

# Bagging with kNN
bagging_knn = BaggingClassifier(estimator=KNeighborsClassifier(), n_estimators=10, random_state=42)
evaluate_model(bagging_knn, "Bagging with kNN")

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
evaluate_model(random_forest, "Random Forest")



# 4. Hyperplane-Based Models
# Support Vector Machines (SVM)
# Linear Kernel
svm_linear = SVC(kernel='linear', random_state=42)
evaluate_model(svm_linear, "SVM (Linear Kernel)")

# Polynomial Kernel
svm_poly = SVC(kernel='poly', degree=3, random_state=42)
evaluate_model(svm_poly, "SVM (Polynomial Kernel)")

# RBF Kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
evaluate_model(svm_rbf, "SVM (RBF Kernel)")

# Perceptron
from sklearn.linear_model import Perceptron
perceptron = Perceptron(max_iter=1000, random_state=42)
evaluate_model(perceptron, "Perceptron")


# Hyperparameter Tuning Example: kNN
knn_params = {'n_neighbors': [3, 5, 7, 10]}
grid_search = GridSearchCV(KNeighborsClassifier(), knn_params, cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters for kNN:", grid_search.best_params_)
evaluate_model(grid_search.best_estimator_, "Tuned k-Nearest Neighbors (kNN)")

# Import necessary libraries for the third part
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report, confusion_matrix

# Load the Wine dataset
wine_data = load_wine()
X = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)
y = wine_data.target
```

```
# One-hot encode the target variable
encoder = OneHotEncoder(sparse_output=False)  # Corrected argument
y_encoded = encoder.fit_transform(y.reshape(-1, 1))

# Normalize the data
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Split the dataset into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(X_normalized, y_encoded, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Define a function to create and compile the model
def create_model(input_dim, hidden_layers=[64, 32], dropout_rate=0.2, learning_rate=0.001):
    model = Sequential()
    model.add(Dense(hidden_layers[0], input_dim=input_dim, activation='relu'))
    for units in hidden_layers[1:]:
        model.add(Dense(units, activation='relu'))
        model.add(Dropout(dropout_rate))
    model.add(Dense(3, activation='softmax'))  # Output layer for 3 classes
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Experiment 1: Baseline Model
print("\n--- Experiment 1: Baseline Model ---")
baseline_model = create_model(input_dim=X_train.shape[1], hidden_layers=[64, 32])
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
history = baseline_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=16, callbacks=[early_stopping], ver
baseline_results = baseline_model.evaluate(X_test, y_test, verbose=0)
print(f"Baseline Test Accuracy: {baseline_results[1]:.4f}")

# Experiment 2: Deeper Network
print("\n--- Experiment 2: Deeper Network ---")
deeper_model = create_model(input_dim=X_train.shape[1], hidden_layers=[128, 64, 32], dropout_rate=0.3)
history_deeper = deeper_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=16, callbacks=[early_stopping]
deeper_results = deeper_model.evaluate(X_test, y_test, verbose=0)
print(f"Deeper Network Test Accuracy: {deeper_results[1]:.4f}")

# Experiment 3: Lighter Network
print("\n--- Experiment 3: Lighter Network ---")
lighter_model = create_model(input_dim=X_train.shape[1], hidden_layers=[32, 16], dropout_rate=0.5)
history_lighter = lighter_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=16, callbacks=[early_stoppin
lighter_results = lighter_model.evaluate(X_test, y_test, verbose=0)
print(f"Lighter Network Test Accuracy: {lighter_results[1]:.4f}")

# Compare all configurations
print("\n--- Comparison of Models ---")
print(f"Baseline Model Accuracy: {baseline_results[1]:.4f}")
print(f"Deeper Network Accuracy: {deeper_results[1]:.4f}")
print(f"Lighter Network Accuracy: {lighter_results[1]:.4f}")

# Generate classification report for the best model
y_pred = np.argmax(baseline_model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)
print("\n--- Classification Report for Baseline Model ---")
print(classification_report(y_true, y_pred))

# Generate confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_true, y_pred))
```

```
Missing values per column:
 alcohol                         0
malic_acid                       0
ash                              0
alcalinity_of_ash                0
magnesium                        0
total_phenols                    0
flavanoids                       0
nonflavanoid_phenols             0
proanthocyanins                  0
color_intensity                  0
hue                              0
od280/od315_of_diluted_wines     0
proline                          0
dtype: int64

Descriptive statistics:
          alcohol  malic_acid         ash  alcalinity_of_ash   magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     11.030000    0.740000    1.360000          10.600000   70.000000
25%     12.362500    1.602500    2.210000          17.200000   88.000000
50%     13.050000    1.865000    2.360000          19.500000   98.000000
75%     13.677500    3.082500    2.557500          21.500000  107.000000
max     14.830000    5.800000    3.230000          30.000000  162.000000

       total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
count     178.000000  178.000000            178.000000       178.000000
mean        2.295112    2.029270              0.361854         1.590899
std         0.625851    0.998859              0.124453         0.572359
min         0.980000    0.340000              0.130000         0.410000
25%         1.742500    1.205000              0.270000         1.250000
50%         2.355000    2.135000              0.340000         1.555000
75%         2.800000    2.875000              0.437500         1.950000
max         3.880000    5.080000              0.660000         3.580000

       color_intensity         hue  od280/od315_of_diluted_wines      proline
count       178.000000  178.000000                    178.000000   178.000000
mean          5.058090    0.957449                      2.611685   746.893258
std           2.318286    0.228572                      0.709990   314.907474
min           1.280000    0.480000                      1.270000   278.000000
25%           3.220000    0.782500                      1.937500   500.500000
50%           4.690000    0.965000                      2.780000   673.500000
75%           6.200000    1.120000                      3.170000   985.000000
max          13.000000    1.710000                      4.000000  1680.000000
```








```
Results for k-Nearest Neighbors (kNN):

Accuracy: 0.96

Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      0.90      0.95        21
           2       0.93      1.00      0.97        14

    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54

Confusion Matrix:
 [[19  0  0]
 [ 1 19  1]
 [ 0  0 14]]
------------------------------

Results for Centroid Classifier:

Accuracy: 0.98

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.95      0.98        21
           2       0.93      1.00      0.97        14

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54

Confusion Matrix:
 [[19  0  0]
 [ 0 20  1]
 [ 0  0 14]]
------------------------------

Results for Gaussian Naive Bayes (GNB):

Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54

Confusion Matrix:
 [[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
------------------------------
```

```
Results for Linear Discriminant Analysis (LDA):

Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54

Confusion Matrix:
 [[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
------------------------------

Results for Decision Tree:

Accuracy: 0.96

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        19
           1       0.95      1.00      0.98        21
           2       1.00      0.93      0.96        14

    accuracy                           0.96        54
   macro avg       0.97      0.96      0.96        54
weighted avg       0.96      0.96      0.96        54

Confusion Matrix:
 [[18  1  0]
 [ 0 21  0]
 [ 1  0 13]]
------------------------------

Results for Bagging with kNN:

Accuracy: 0.94

Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      0.86      0.92        21
           2       0.88      1.00      0.93        14

    accuracy                           0.94        54
   macro avg       0.94      0.95      0.94        54
weighted avg       0.95      0.94      0.94        54

Confusion Matrix:
 [[19  0  0]
 [ 1 18  2]
 [ 0  0 14]]
------------------------------

Results for Random Forest:

Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54

Confusion Matrix:
 [[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
------------------------------

Results for SVM (Linear Kernel):

Accuracy: 0.98

Classification Report:
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00        19
           1       1.00      0.95      0.98        21
           2       0.93      1.00      0.97        14

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54

Confusion Matrix:
 [[19  0  0]
 [ 0 20  1]
 [ 0  0 14]]
------------------------------

Results for SVM (Polynomial Kernel):

Accuracy: 0.96

Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.89      0.94        19
           1       0.91      1.00      0.95        21
           2       1.00      1.00      1.00        14

    accuracy                           0.96        54
   macro avg       0.97      0.96      0.97        54
weighted avg       0.97      0.96      0.96        54

Confusion Matrix:
 [[17  2  0]
 [ 0 21  0]
 [ 0  0 14]]
------------------------------

Results for SVM (RBF Kernel):

Accuracy: 0.98

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.95      1.00      0.98        21
           2       1.00      0.93      0.96        14

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54

Confusion Matrix:
 [[19  0  0]
 [ 0 21  0]
 [ 0  1 13]]
------------------------------

Results for Perceptron:

Accuracy: 0.98

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.95      0.98        21
           2       0.93      1.00      0.97        14

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54

Confusion Matrix:
 [[19  0  0]
 [ 0 20  1]
 [ 0  0 14]]
------------------------------

Best parameters for kNN: {'n_neighbors': 5}
Results for Tuned k-Nearest Neighbors (kNN):

Accuracy: 0.96

Classification Report:
               precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      0.90      0.95        21
           2       0.93      1.00      0.97        14

    accuracy                           0.96        54
```

```
    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54

Confusion Matrix:
 [[19  0  0]
 [ 1 19  1]
 [ 0  0 14]]
----------------------------


--- Experiment 1: Baseline Model ---
Epoch 1/100
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.2304 - loss: 1.2847 - val_accuracy: 0.4444 - val_loss: 1.1135
Epoch 2/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4745 - loss: 0.9680 - val_accuracy: 0.7037 - val_loss: 0.9067
Epoch 3/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6565 - loss: 0.8249 - val_accuracy: 0.9259 - val_loss: 0.7333
Epoch 4/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8350 - loss: 0.6979 - val_accuracy: 0.9630 - val_loss: 0.5930
Epoch 5/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8936 - loss: 0.5773 - val_accuracy: 0.9630 - val_loss: 0.4784
Epoch 6/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9650 - loss: 0.4367 - val_accuracy: 0.9630 - val_loss: 0.3837
Epoch 7/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9773 - loss: 0.3637 - val_accuracy: 1.0000 - val_loss: 0.3054
Epoch 8/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9666 - loss: 0.2992 - val_accuracy: 1.0000 - val_loss: 0.2439
Epoch 9/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9743 - loss: 0.2461 - val_accuracy: 1.0000 - val_loss: 0.1975
Epoch 10/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9853 - loss: 0.2089 - val_accuracy: 1.0000 - val_loss: 0.1628
Epoch 11/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9673 - loss: 0.1791 - val_accuracy: 1.0000 - val_loss: 0.1372
Epoch 12/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9832 - loss: 0.1367 - val_accuracy: 1.0000 - val_loss: 0.1164
Epoch 13/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9832 - loss: 0.1240 - val_accuracy: 1.0000 - val_loss: 0.1004
Epoch 14/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9802 - loss: 0.1082 - val_accuracy: 1.0000 - val_loss: 0.0871
Epoch 15/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.1046 - val_accuracy: 1.0000 - val_loss: 0.0765
Epoch 16/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9947 - loss: 0.1041 - val_accuracy: 1.0000 - val_loss: 0.0688
Epoch 17/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9961 - loss: 0.0603 - val_accuracy: 1.0000 - val_loss: 0.0618
Epoch 18/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9947 - loss: 0.0550 - val_accuracy: 1.0000 - val_loss: 0.0553
Epoch 19/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0685 - val_accuracy: 1.0000 - val_loss: 0.0504
Epoch 20/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9929 - loss: 0.0459 - val_accuracy: 1.0000 - val_loss: 0.0458
Epoch 21/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0424 - val_accuracy: 1.0000 - val_loss: 0.0418
Epoch 22/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9906 - loss: 0.0525 - val_accuracy: 1.0000 - val_loss: 0.0394
Epoch 23/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0385 - val_accuracy: 1.0000 - val_loss: 0.0366
Epoch 24/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0373 - val_accuracy: 1.0000 - val_loss: 0.0342
Epoch 25/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9763 - loss: 0.0555 - val_accuracy: 1.0000 - val_loss: 0.0323
Epoch 26/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0347 - val_accuracy: 1.0000 - val_loss: 0.0317
Epoch 27/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0212 - val_accuracy: 1.0000 - val_loss: 0.0309
Epoch 28/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0266 - val_accuracy: 1.0000 - val_loss: 0.0292
Epoch 29/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0228 - val_accuracy: 1.0000 - val_loss: 0.0272
Epoch 30/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0218 - val_accuracy: 1.0000 - val_loss: 0.0261
Epoch 31/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9906 - loss: 0.0279 - val_accuracy: 1.0000 - val_loss: 0.0248
Epoch 32/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9871 - loss: 0.0357 - val_accuracy: 1.0000 - val_loss: 0.0229
Epoch 33/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0198 - val_accuracy: 1.0000 - val_loss: 0.0221
Epoch 34/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0154 - val_accuracy: 1.0000 - val_loss: 0.0219
Epoch 35/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0126 - val_accuracy: 1.0000 - val_loss: 0.0220
Epoch 36/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0140 - val_accuracy: 1.0000 - val_loss: 0.0217
Epoch 37/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0222 - val_accuracy: 1.0000 - val_loss: 0.0215
Epoch 38/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0128 - val_accuracy: 1.0000 - val_loss: 0.0210
Epoch 39/100
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0186 - val_accuracy: 1.0000 - val_loss: 0.0198
Epoch 40/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9802 - loss: 0.0328 - val_accuracy: 1.0000 - val_loss: 0.0184
Epoch 41/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0118 - val_accuracy: 1.0000 - val_loss: 0.0180
Epoch 42/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0087 - val_accuracy: 1.0000 - val_loss: 0.0183
Epoch 43/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0186 - val_accuracy: 1.0000 - val_loss: 0.0187
Epoch 44/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0107 - val_accuracy: 1.0000 - val_loss: 0.0194
Epoch 45/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0095 - val_accuracy: 1.0000 - val_loss: 0.0196
Epoch 46/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0102 - val_accuracy: 1.0000 - val_loss: 0.0190
Epoch 47/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0117 - val_accuracy: 1.0000 - val_loss: 0.0187
Epoch 48/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0076 - val_accuracy: 1.0000 - val_loss: 0.0184
Epoch 49/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0094 - val_accuracy: 1.0000 - val_loss: 0.0174
Epoch 50/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0069 - val_accuracy: 1.0000 - val_loss: 0.0171
Epoch 51/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0054 - val_accuracy: 1.0000 - val_loss: 0.0169
Epoch 52/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0110 - val_accuracy: 1.0000 - val_loss: 0.0163
Epoch 53/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0128 - val_accuracy: 1.0000 - val_loss: 0.0158
Epoch 54/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0063 - val_accuracy: 1.0000 - val_loss: 0.0150
Epoch 55/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0072 - val_accuracy: 1.0000 - val_loss: 0.0155
Epoch 56/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0043 - val_accuracy: 1.0000 - val_loss: 0.0157
Epoch 57/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0083 - val_accuracy: 1.0000 - val_loss: 0.0160
Epoch 58/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0074 - val_accuracy: 1.0000 - val_loss: 0.0158
Epoch 59/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 1.0000 - val_loss: 0.0144
Epoch 60/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 1.0000 - val_loss: 0.0142
Epoch 61/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0056 - val_accuracy: 1.0000 - val_loss: 0.0139
Epoch 62/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 1.0000 - val_loss: 0.0138
Epoch 63/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0050 - val_accuracy: 1.0000 - val_loss: 0.0140
Epoch 64/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0051 - val_accuracy: 1.0000 - val_loss: 0.0143
Epoch 65/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0053 - val_accuracy: 1.0000 - val_loss: 0.0144
Epoch 66/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 1.0000 - val_loss: 0.0143
Epoch 67/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 1.0000 - val_loss: 0.0141
Epoch 68/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 1.0000 - val_loss: 0.0138
Epoch 69/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 1.0000 - val_loss: 0.0134
Epoch 70/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0042 - val_accuracy: 1.0000 - val_loss: 0.0133
Epoch 71/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 1.0000 - val_loss: 0.0130
Epoch 72/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0039 - val_accuracy: 1.0000 - val_loss: 0.0130
Epoch 73/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0070 - val_accuracy: 1.0000 - val_loss: 0.0133
Epoch 74/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 1.0000 - val_loss: 0.0135
Epoch 75/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 1.0000 - val_loss: 0.0137
Epoch 76/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 1.0000 - val_loss: 0.0134
Epoch 77/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 1.0000 - val_loss: 0.0133
Epoch 78/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 0.0133
Epoch 79/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 0.0134
Epoch 80/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 1.0000 - val_loss: 0.0131
Epoch 81/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0024 - val_accuracy: 1.0000 - val_loss: 0.0129
Epoch 82/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 1.0000 - val_loss: 0.0121
Epoch 83/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 1.0000 - val_loss: 0.0120
Epoch 84/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 0.0121
```

```
Epoch 85/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 1.0000 - val_loss: 0.0122
Epoch 86/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 1.0000 - val_loss: 0.0119
Epoch 87/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0079 - val_accuracy: 1.0000 - val_loss: 0.0113
Epoch 88/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 1.0000 - val_loss: 0.0110
Epoch 89/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 1.0000 - val_loss: 0.0109
Epoch 90/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 0.0110
Epoch 91/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 1.0000 - val_loss: 0.0113
Epoch 92/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 0.0113
Epoch 93/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 1.0000 - val_loss: 0.0111
Epoch 94/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 0.0110
Epoch 95/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 1.0000 - val_loss: 0.0113
Epoch 96/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 0.0115
Epoch 97/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 0.0113
Epoch 98/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 1.0000 - val_loss: 0.0110
Epoch 99/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 0.0107
Epoch 100/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 1.0000 - val_loss: 0.0109
Baseline Test Accuracy: 1.0000

--- Experiment 2: Deeper Network ---
Epoch 1/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.3915 - loss: 1.0672 - val_accuracy: 0.8148 - val_loss: 0.8059
Epoch 2/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7289 - loss: 0.7895 - val_accuracy: 1.0000 - val_loss: 0.6198
Epoch 3/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7992 - loss: 0.6695 - val_accuracy: 1.0000 - val_loss: 0.4507
Epoch 4/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8608 - loss: 0.5231 - val_accuracy: 1.0000 - val_loss: 0.3117
Epoch 5/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9315 - loss: 0.4197 - val_accuracy: 1.0000 - val_loss: 0.2017
Epoch 6/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9277 - loss: 0.2765 - val_accuracy: 1.0000 - val_loss: 0.1343
Epoch 7/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9529 - loss: 0.1965 - val_accuracy: 1.0000 - val_loss: 0.0960
Epoch 8/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9676 - loss: 0.1743 - val_accuracy: 1.0000 - val_loss: 0.0725
Epoch 9/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9738 - loss: 0.1356 - val_accuracy: 1.0000 - val_loss: 0.0543
Epoch 10/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9802 - loss: 0.0920 - val_accuracy: 1.0000 - val_loss: 0.0430
Deeper Network Test Accuracy: 0.8148

--- Experiment 3: Lighter Network ---
Epoch 1/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.2123 - loss: 1.7183 - val_accuracy: 0.0370 - val_loss: 1.5579
Epoch 2/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2246 - loss: 1.8354 - val_accuracy: 0.1111 - val_loss: 1.3539
Epoch 3/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.2991 - loss: 1.3862 - val_accuracy: 0.2593 - val_loss: 1.1993
Epoch 4/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.3631 - loss: 1.3060 - val_accuracy: 0.3333 - val_loss: 1.0865
Epoch 5/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3649 - loss: 1.2096 - val_accuracy: 0.4444 - val_loss: 0.9993
Epoch 6/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4890 - loss: 1.0203 - val_accuracy: 0.4444 - val_loss: 0.9319
Epoch 7/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6368 - loss: 0.9329 - val_accuracy: 0.5185 - val_loss: 0.8744
Epoch 8/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5233 - loss: 0.9812 - val_accuracy: 0.5926 - val_loss: 0.8306
Epoch 9/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6223 - loss: 0.8652 - val_accuracy: 0.6296 - val_loss: 0.7916
Epoch 10/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6204 - loss: 0.8301 - val_accuracy: 0.7037 - val_loss: 0.7572
Lighter Network Test Accuracy: 0.1111

--- Comparison of Models ---
Baseline Model Accuracy: 1.0000
Deeper Network Accuracy: 0.8148
Lighter Network Accuracy: 0.1111
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step

--- Classification Report for Baseline Model ---
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
```

```
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00         8

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27


Confusion Matrix:
[[ 9  0  0]
 [ 0 10  0]
 [ 0  0  8]]
```