AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
International Credit Hours Programs (ICHEP)
Mechatronics and Automation

Computational Intelligence (CSE473)

Lab 6 Report

Name	ID	Section
Mahmoud Elsayd Abdelqader Labib Eldwakhly	21P0017	1

Submitted to Dr Hossam Hassan & Eng. Ali Ahmed

Fall 2025

1. Introduction

The goal of this assignment was to build two different "smart" systems (classifiers) that can look at a specific shape of data points—called the "Double Moon"—and correctly label them into four distinct categories.

We used two famous machine learning methods to do this:

1. MCSVM (Multi-Class Support Vector Machine).
2. MLNN (Multi-Layer Neural Network).

The main challenge is that the "Moon" shapes are curved, so we cannot just draw a straight line to separate them. The computer needs to learn how to draw curved boundaries.

2. The Dataset

First, we created the data. We generated 2,000 points shaped like two interlocking moons.

- The Original Shape: Two moons (Top and Bottom).
- The 4 Classes: To make it harder (and match the assignment), we split the moons in half:
    - Class 0 (Dark Blue): Top Moon, Left side.
    - Class 1 (Red): Top Moon, Right side.
    - Class 2 (Yellow): Bottom Moon, Left side.
    - Class 3 (Light Blue): Bottom Moon, Right side.

Why this is tricky: The computer has to figure out not just "Top vs. Bottom," but also "Left vs. Right" while following the curve of the moon.

3. Method 1: The MCSVM Classifier

What is it? Think of an SVM (Support Vector Machine) as a system that tries to draw the widest possible street between different groups of points.

How we set it up:

- The "Kernel" Trick: Since our data is curved, a standard straight line won't work. We used something called the RBF Kernel. This is a mathematical trick that allows the SVM to draw flexible, curvy boundaries around the moons.
- Visualizing "Learning": SVMs usually solve the problem in one go (unlike neural networks). To satisfy the assignment requirement of "visualizing loss," we forced the SVM to learn step-by-step (increasing the iterations slowly). This showed us that the error rate dropped quickly as the model was allowed more time to think.

The Result: The SVM did an excellent job. It drew smooth, curved lines exactly where the colors changed. It handled the gap between the two moons perfectly.

4. Method 2: The MLNN Classifier

What is it? A Neural Network (MLNN) works a bit like a human brain. It uses layers of "neurons" to pass information along and learn complex patterns.

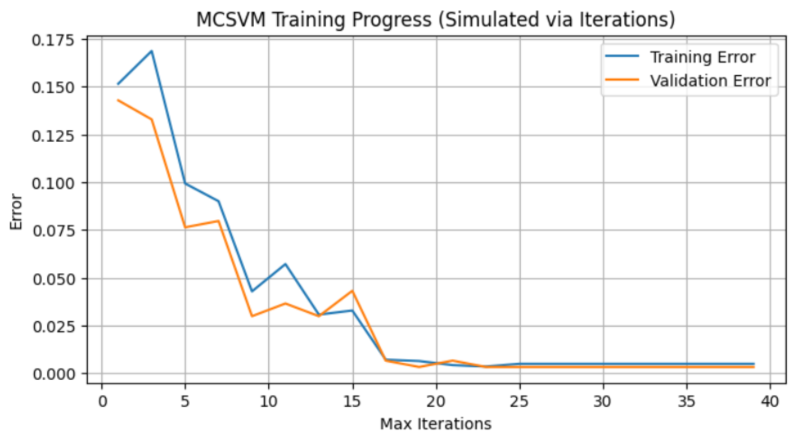
How we set it up:

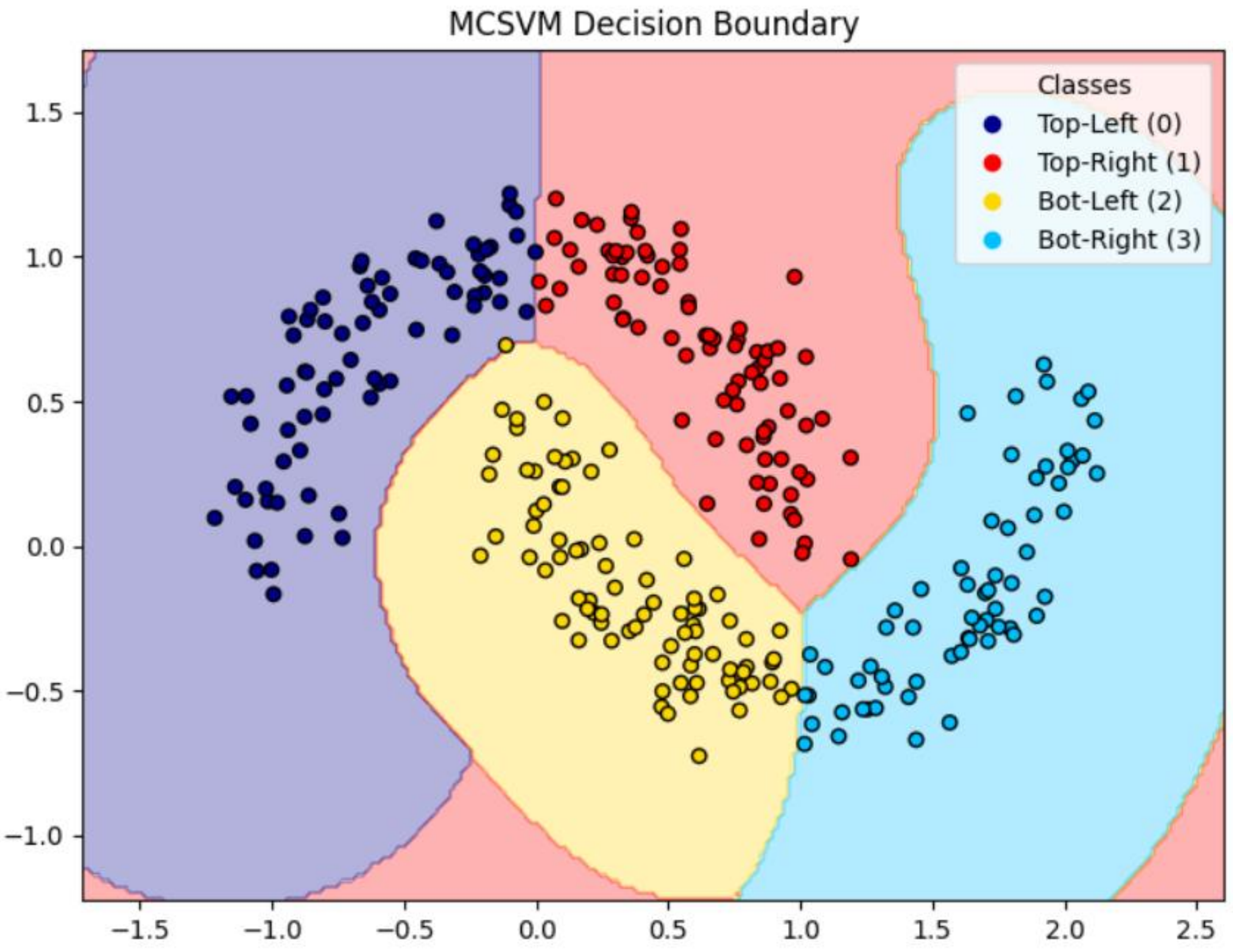
- **The Structure:** We built a network with:
  - **Input Layer:** Reads the X and Y coordinates.
  - **Hidden Layers:** Two layers of 64 neurons each. These do the "heavy lifting" to figure out the curves.
  - **Output Layer:** 4 neurons (one for each color/class).
- **Training:** We trained it for 50 "epochs" (rounds of practice).
- **Scaling:** Neural networks are sensitive to raw numbers, so we "scaled" the data (made all numbers small and centered) before feeding it in. This made training much faster and more accurate.

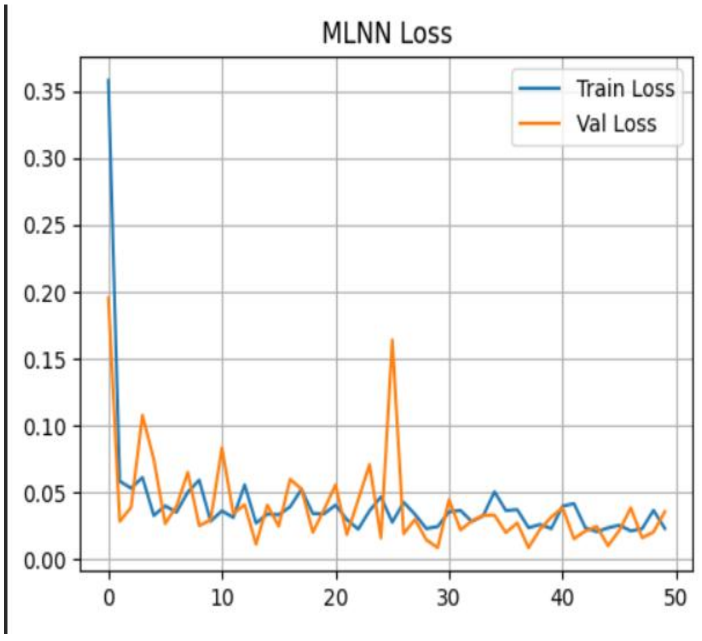
The Result: The Neural Network also succeeded. The "Loss" graph showed the error dropping steadily over time, and the "Accuracy" graph went up. The final decision boundary looked very similar to the SVM, effectively separating all four colors.

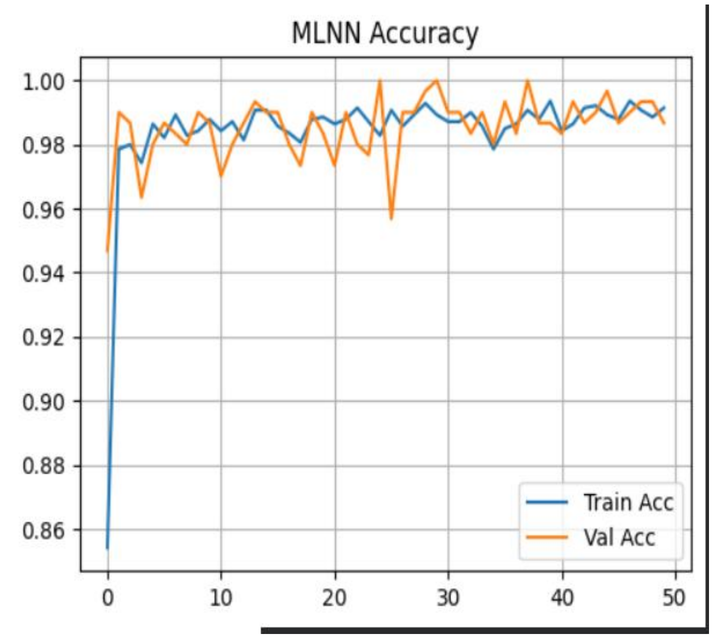
Feature	MCSVM (Support Vector Machine)	MLNN (Neural Network)
Setup	Easier to set up. Requires less data preparation.	Requires more setup (defining layers, epochs, batch size).
Speed	Very fast for this amount of data.	Slightly slower to train, but still fast.
Accuracy	**High Accuracy.** The boundaries were very smooth.	**High Accuracy.** The boundaries were flexible but slightly more "wobbly" than the SVM.
Data Prep	Worked well on raw data.	**Required "Scaling"** to work properly.

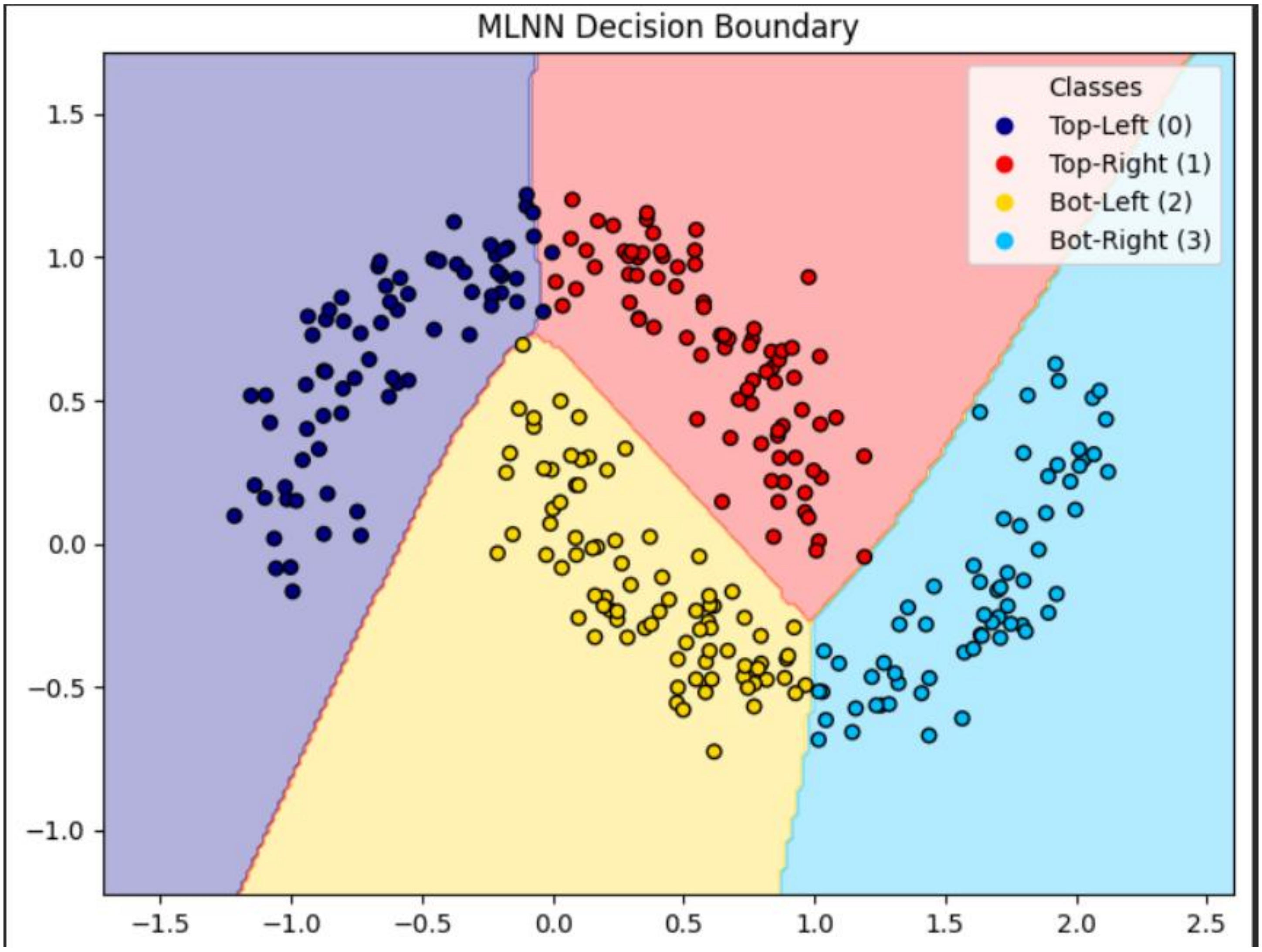
5. Results

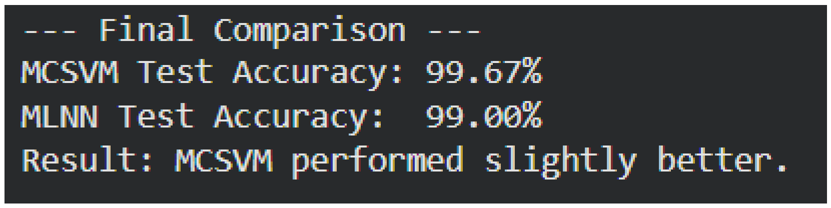



MCSVM Decision Boundary
Classes
Top-Left (0)
Top-Right (1)
Bot-Left (2)
Bot-Right (3)


MLNN Loss
Train Loss
Val Loss


MLNN Accuracy
Train Acc
Val Acc





```
--- Final Comparison ---
MCSVM Test Accuracy: 99.67%
MLNN Test Accuracy:  99.00%
Result: MCSVM performed slightly better.
```

```python
# Mahmoud Elsayd Eldwakhly
# ID : 21P0017
# Lab 6

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# ==========================================
# 1. VISUALIZATION SETUP
# ==========================================
# Define the exact solid colors seen in assignment:
# 0: Dark Blue, 1: Red, 2: Yellow, 3: Light Blue
custom_colors = ['#00008B', '#FF0000', '#FFD700', '#00BFFF']
my_cmap = ListedColormap(custom_colors)

def plot_decision_boundary(model, X, y, title="Decision Boundary", is_keras=False, scaler=None):
    """
    Plots the decision boundary using the specific assignment colors.
    """
    # Define bounds for the meshgrid
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Predict Class for every point on the meshgrid
    grid_points = np.c_[xx.ravel(), yy.ravel()]

    if is_keras and scaler:
        grid_points_scaled = scaler.transform(grid_points)
        Z = model.predict(grid_points_scaled, verbose=0)
        Z = np.argmax(Z, axis=1) # Convert probabilities to class labels
    else:
        Z = model.predict(grid_points)

    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(8, 6))
    # Filled contour for background regions (using custom solid colors)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=my_cmap)

    # Scatter plot for data points (with black edges for visibility)
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, s=30, edgecolor='k', cmap=my_cmap)

    # Manual Legend
    handles, _ = scatter.legend_elements()
    legend_labels = ['Top-Left (0)', 'Top-Right (1)', 'Bot-Left (2)', 'Bot-Right (3)']
    plt.legend(handles, legend_labels, title="Classes")
    plt.title(title)
    plt.show()

# ==========================================
# 2. DATASET CREATION
# ==========================================
def create_four_class_moons(n_samples=2000, noise=0.1):
    X, y_binary = make_moons(n_samples=n_samples, noise=noise, random_state=42)
    y_four_class = np.zeros(n_samples, dtype=int)

    for i in range(n_samples):
        x_val = X[i, 0]
        original_label = y_binary[i]

        if original_label == 0: # Top Moon
```

```
            if x_val < 0.0: y_four_class[i] = 0 # Dark Blue
            else:           y_four_class[i] = 1 # Red
        else: # Bottom Moon
            if x_val < 1.0: y_four_class[i] = 2 # Yellow
            else:           y_four_class[i] = 3 # Light Blue
    return X, y_four_class

print("Generating Data...")
X, y = create_four_class_moons(n_samples=2000, noise=0.12)

# Split: Train, Validation, Test
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.1765, random_state=42)

# ==========================================
# 3. MCSVM CLASSIFIER (Task 2)
# ==========================================
print("\n--- Training MCSVM ---")

# A. Simulate Training Curve (Iterative Limit)
svm_train_errors, svm_val_errors = [], []
iterations = range(1, 40, 2)

for max_i in iterations:
    # Train with limited iterations
    svm = SVC(kernel='rbf', C=10, gamma='scale', max_iter=max_i, random_state=42)
    svm.fit(X_train, y_train)
    # Record Error (1 - Accuracy)
    svm_train_errors.append(1 - svm.score(X_train, y_train))
    svm_val_errors.append(1 - svm.score(X_val, y_val))

# Plot SVM Learning Curve
plt.figure(figsize=(8, 4))
plt.plot(iterations, svm_train_errors, label='Training Error')
plt.plot(iterations, svm_val_errors, label='Validation Error')
plt.title('MCSVM Training Progress (Simulated via Iterations)')
plt.xlabel('Max Iterations')
plt.ylabel('Error')
plt.legend()
plt.grid(True)
plt.show()

# B. Final SVM Model & Boundary
final_svm = SVC(kernel='rbf', C=20, gamma=2, random_state=42)
final_svm.fit(X_train, y_train)
plot_decision_boundary(final_svm, X_test, y_test, title="MCSVM Decision Boundary")

# ==========================================
# 4. MLNN CLASSIFIER (Task 3)
# ==========================================
print("\n--- Training MLNN (TensorFlow) ---")

# Scale data (Standardization is best for Neural Networks)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Build Model
model = Sequential([
    Dense(64, input_shape=(2,), activation='relu'),
    Dense(64, activation='relu'),
    Dense(4, activation='softmax') # 4 Output neurons for 4 classes
])

model.compile(optimizer=Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train
history = model.fit(X_train_scaled, y_train,
                    validation_data=(X_val_scaled, y_val),
                    epochs=50, batch_size=32, verbose=0)
```

```python
# A. Plot Loss & Accuracy
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(history.history['loss'], label='Train Loss')
ax1.plot(history.history['val_loss'], label='Val Loss')
ax1.set_title('MLNN Loss')
ax1.legend()
ax1.grid(True)

ax2.plot(history.history['accuracy'], label='Train Acc')
ax2.plot(history.history['val_accuracy'], label='Val Acc')
ax2.set_title('MLNN Accuracy')
ax2.legend()
ax2.grid(True)
plt.show()

# B. MLNN Decision Boundary
plot_decision_boundary(model, X_test, y_test, title="MLNN Decision Boundary", is_keras=True, scaler=scaler)

# ==========================================
# 5. COMPARISON (Task 4)
# ==========================================
print("\n--- Final Comparison ---")
svm_acc = accuracy_score(y_test, final_svm.predict(X_test))
loss, mlnn_acc = model.evaluate(X_test_scaled, y_test, verbose=0)

print(f"MCSVM Test Accuracy: {svm_acc*100:.2f}%")
print(f"MLNN Test Accuracy:  {mlnn_acc*100:.2f}%")

if svm_acc > mlnn_acc:
    print("Result: MCSVM performed slightly better.")
else:
    print("Result: MLNN performed slightly better.")
```