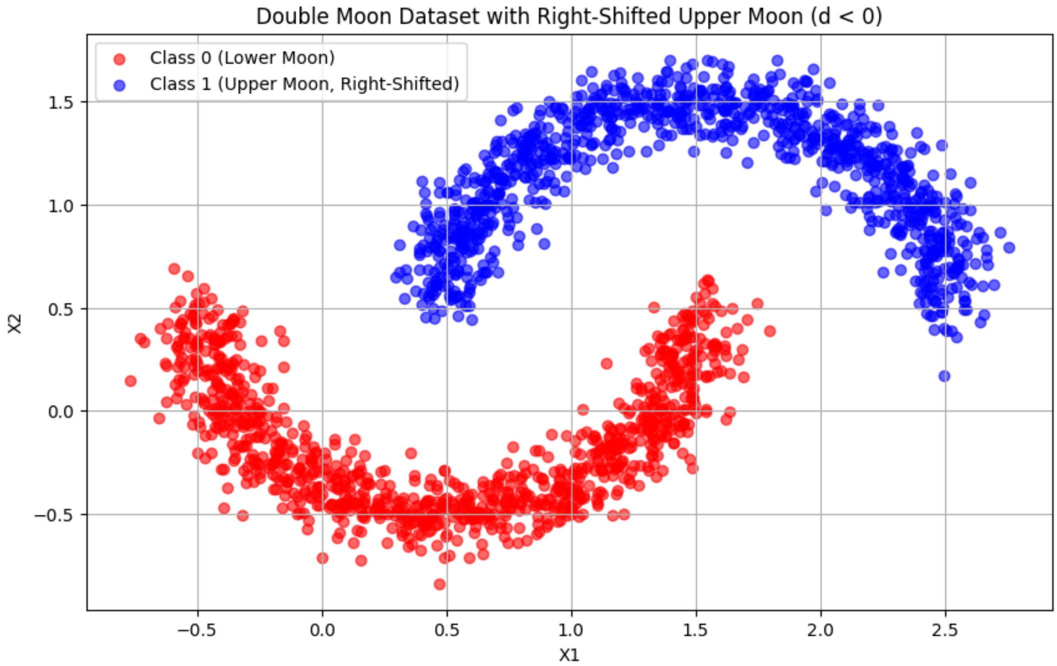
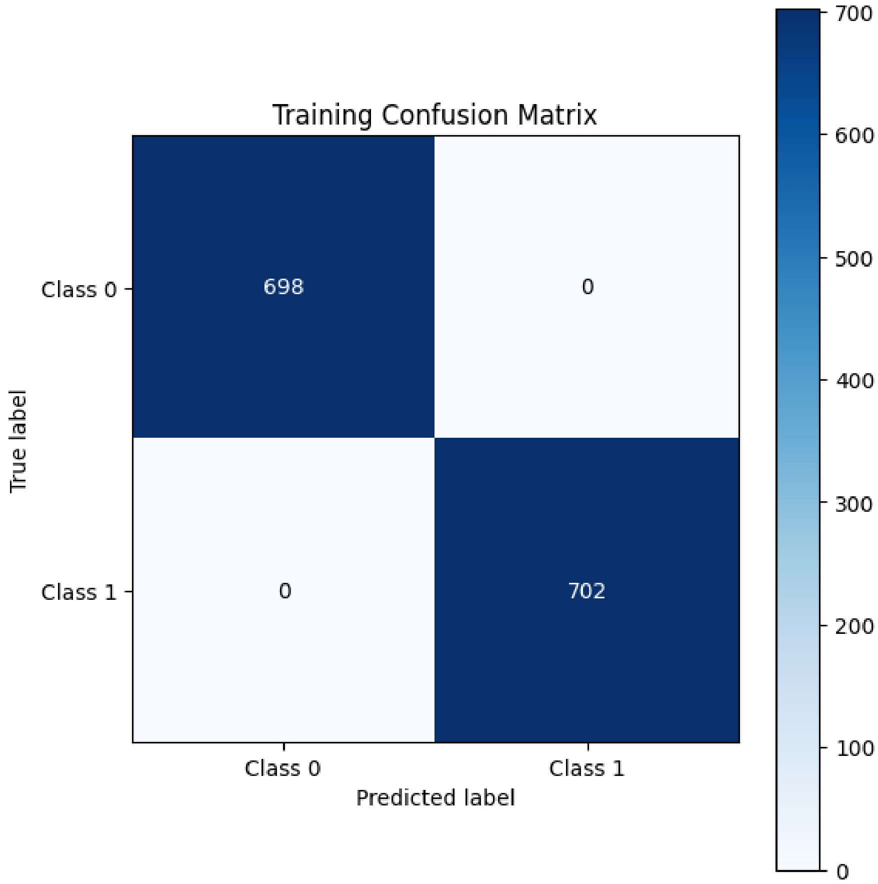
NAME: SHAILESH KUMAR

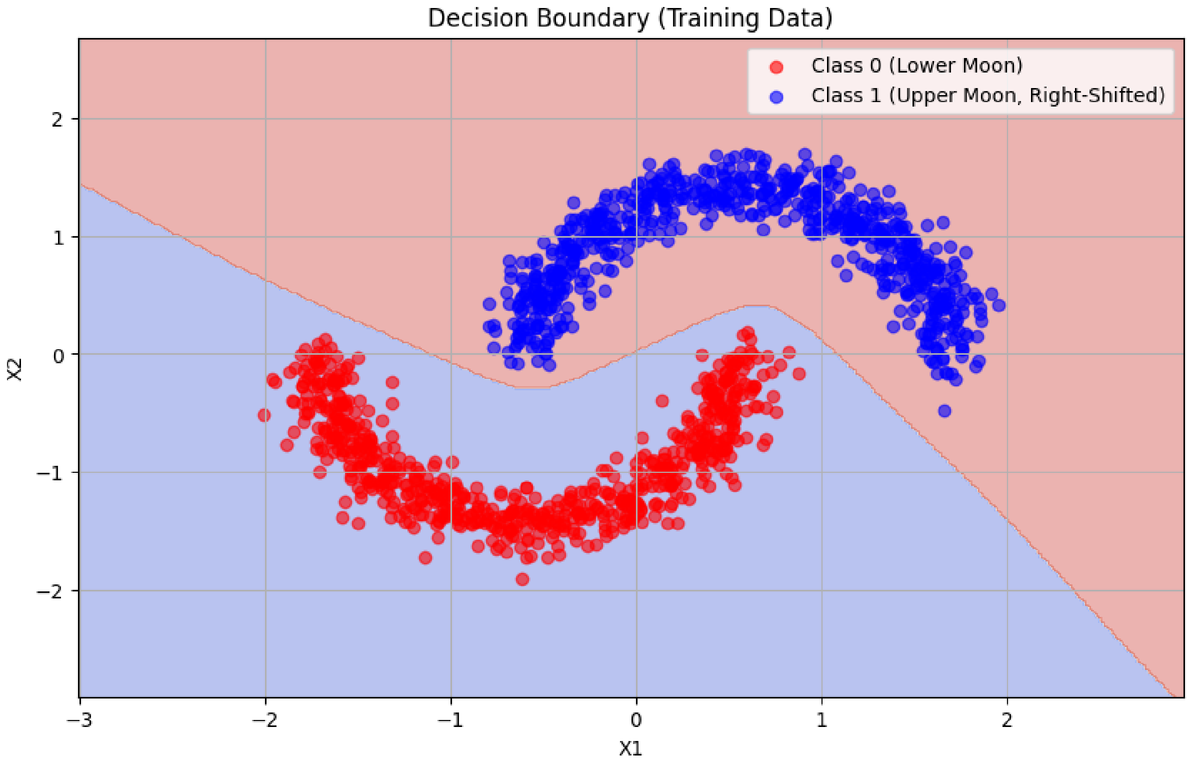
ROLL NO: 224EC6013

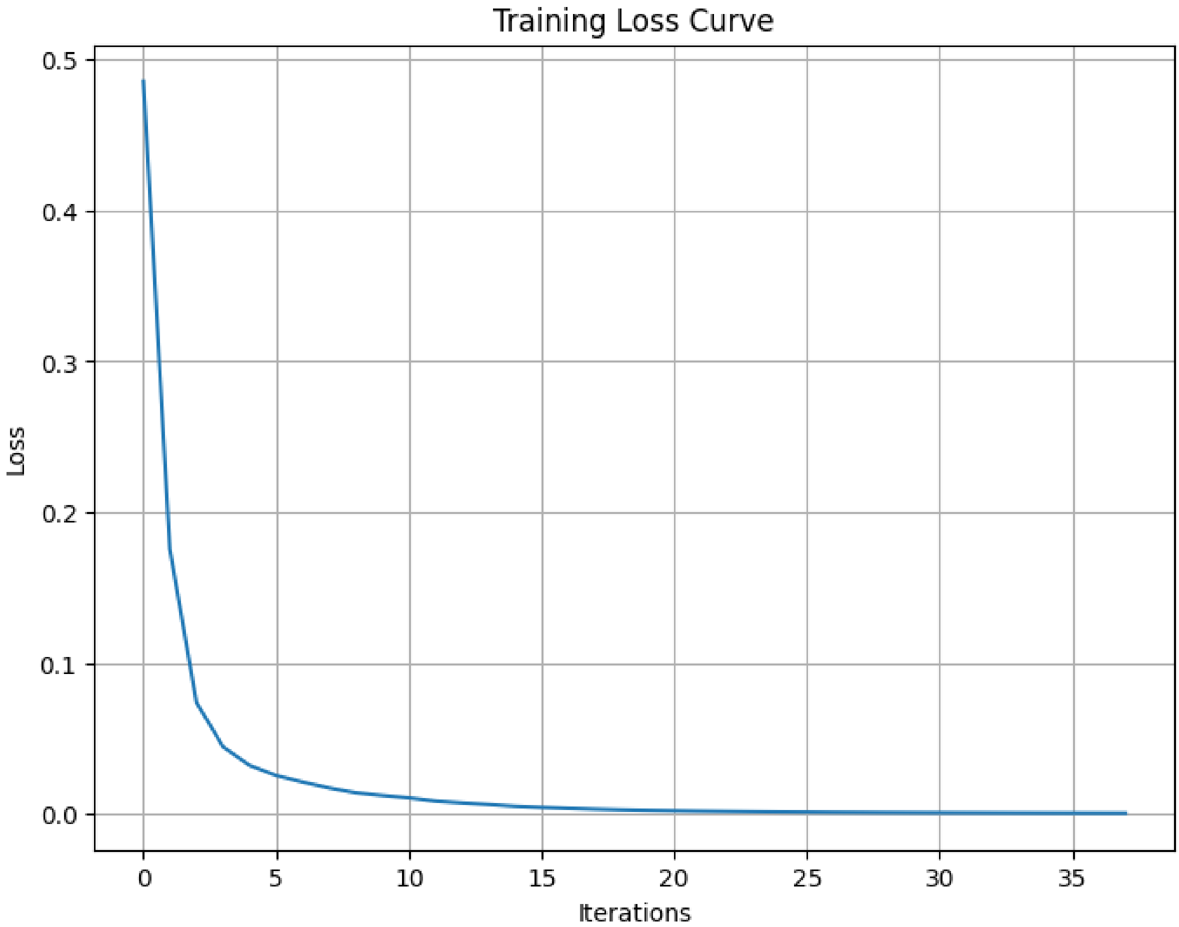
SIGNAL AND IMAGE PROCESSING



SS




Decision Boundary (Training Data)
Class 0 (Lower Moon)
Class 1 (Upper Moon, Right-Shifted)
2
1
0
−1
−2
X2
−3
−2
−1
0
1
2
X1


Training Loss Curve
0.5
0.4
0.3
0.2
0.1
0.0
Loss
0
5
10
15
20
25
30
35
Iterations

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler

# 1. Generate Double Moon Dataset with Right-Shifted Upper Moon
def generate_shifted_double_moon(n_samples=1000, radius=1.0, width=0.5, d=-0.5,
                                 shift=1.0, noise=0.1):
    """
    Generate double moon dataset with upper moon shifted to the right

    Parameters:
    n_samples: Total number of samples (half per moon)
    radius: Radius of the moons
    width: Width of the moons
    d: Vertical distance between moons (negative for intersection)
    shift: Horizontal shift for the upper moon (right shift)
    noise: Gaussian noise added to the data

    Returns:
    X: Array of features (x, y coordinates)
    y: Array of labels (0 for lower moon, 1 for upper moon)
    """
    # Generate upper moon (shifted right)
    theta = np.linspace(0, np.pi, n_samples//2)
    upper_x = radius * np.cos(theta) + radius/2 + shift  # Added shift here
    upper_y = radius * np.sin(theta) - d

    # Generate lower moon (original position)
    lower_x = radius * np.cos(theta) + radius/2
    lower_y = -radius * np.sin(theta) + width

    # Add noise
    upper_x += np.random.normal(0, noise, n_samples//2)
    upper_y += np.random.normal(0, noise, n_samples//2)
    lower_x += np.random.normal(0, noise, n_samples//2)
    lower_y += np.random.normal(0, noise, n_samples//2)

    # Combine data
    X = np.vstack([np.column_stack((lower_x, lower_y)),
                   np.column_stack((upper_x, upper_y))])
    y = np.hstack([np.zeros(n_samples//2), np.ones(n_samples//2)])

    return X, y

# 2. Generate and visualize the data
```

```
X, y = generate_shifted_double_moon(n_samples=2000, d=-0.5, shift=1.0, noise=0.1)

plt.figure(figsize=(10, 6))
plt.scatter(X[y==0, 0], X[y==0, 1], color='red', label='Class 0 (Lower Moon)', alpha=0.6)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', label='Class 1 (Upper Moon, Right-Shifted)',
alpha=0.6)
plt.title("Double Moon Dataset with Right-Shifted Upper Moon (d < 0)")
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend()
plt.grid(True)
plt.show()

# 3. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 4. Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 5. Create and train MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu',
                    solver='adam', alpha=0.0001, batch_size=32,
                    learning_rate='adaptive', learning_rate_init=0.001,
                    max_iter=500, random_state=42, verbose=True)

mlp.fit(X_train, y_train)

# 6. Evaluate the model
train_pred = mlp.predict(X_train)
test_pred = mlp.predict(X_test)

train_acc = accuracy_score(y_train, train_pred)
test_acc = accuracy_score(y_test, test_pred)

print(f"\nTraining Accuracy: {train_acc:.4f}")
print(f"Testing Accuracy: {test_acc:.4f}")

# 7. Plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, ['Class 0', 'Class 1'])
    plt.yticks(tick_marks, ['Class 0', 'Class 1'])

    thresh = cm.max() / 2.
```

```
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

plot_confusion_matrix(y_train, train_pred, "Training Confusion Matrix")
plot_confusion_matrix(y_test, test_pred, "Testing Confusion Matrix")
plt.show()

# 8. Plot decision boundary
def plot_decision_boundary(X, y, model, title):
    # Create a mesh grid
    h = 0.02  # step size
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # Predict for each point in the grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, Z, alpha=0.4, cmap='coolwarm')
    plt.scatter(X[y==0, 0], X[y==0, 1], color='red', label='Class 0 (Lower Moon)', alpha=0.6)
    plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', label='Class 1 (Upper Moon, Right-Shifted)',
alpha=0.6)
    plt.title(title)
    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.legend()
    plt.grid(True)

plot_decision_boundary(X_train, y_train, mlp, "Decision Boundary (Training Data)")
plot_decision_boundary(X_test, y_test, mlp, "Decision Boundary (Testing Data)")
plt.show()

# 9. Plot training loss curve
plt.figure(figsize=(8, 6))
plt.plot(mlp.loss_curve_)
plt.title("Training Loss Curve")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```