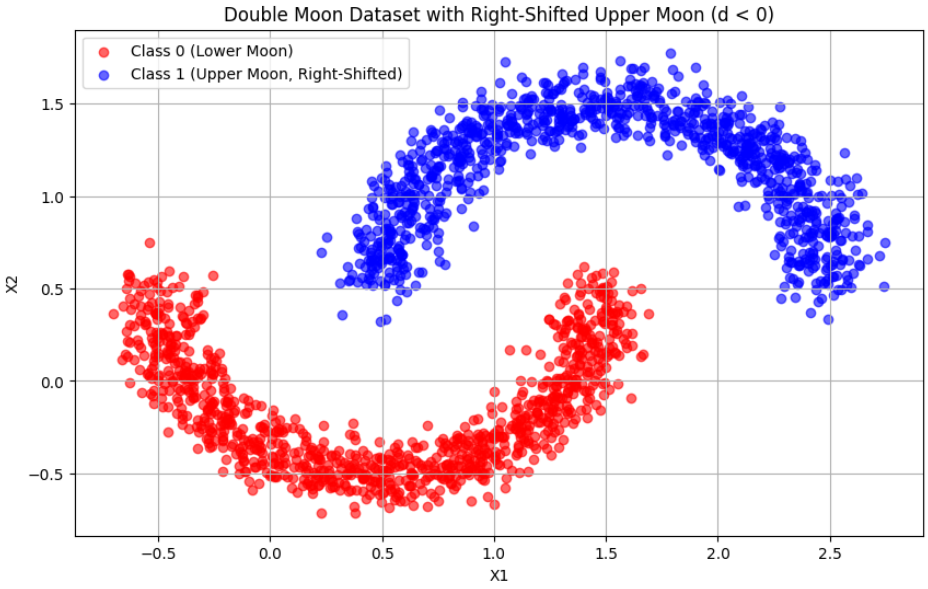
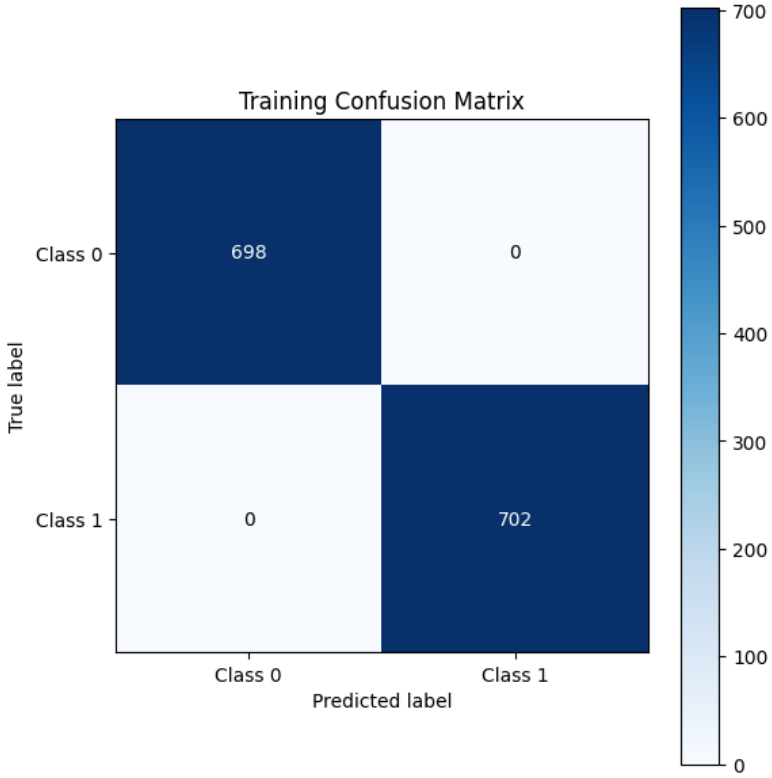
NAME: SHAILESH KUMAR

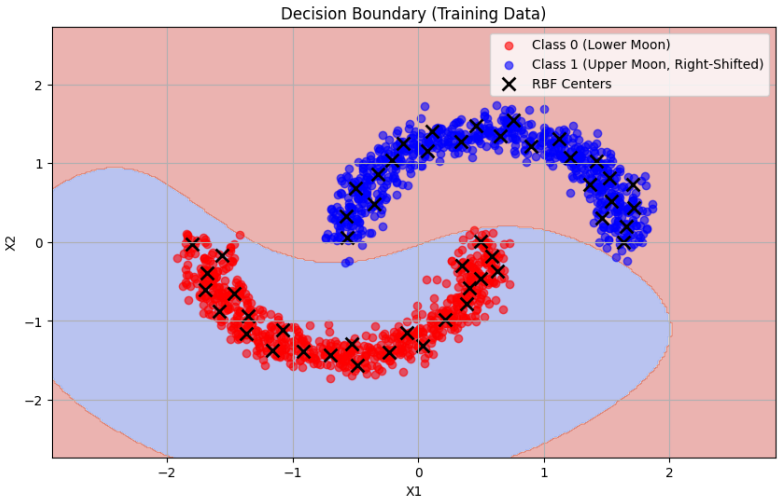
ROLL NO: 224EC6013

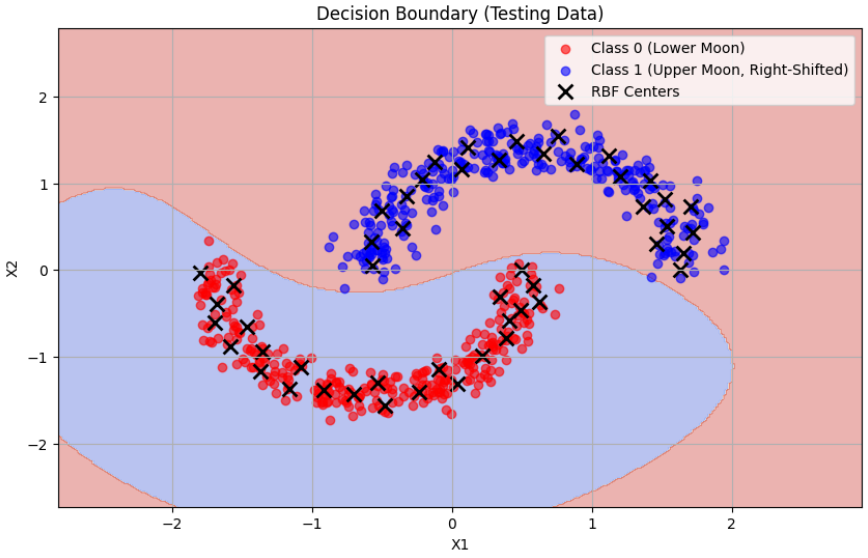
SIGNAL AND IMAGE PROCESSING

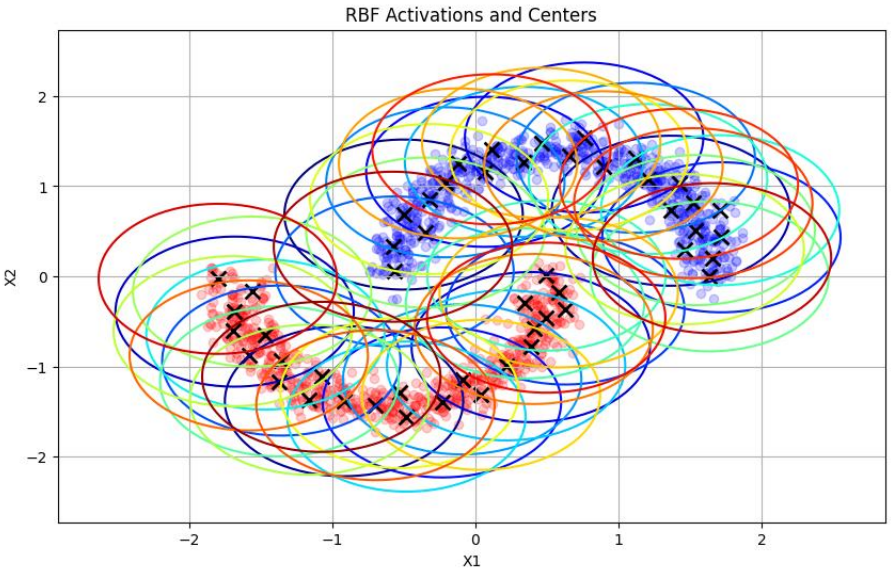
Assignment No: 3

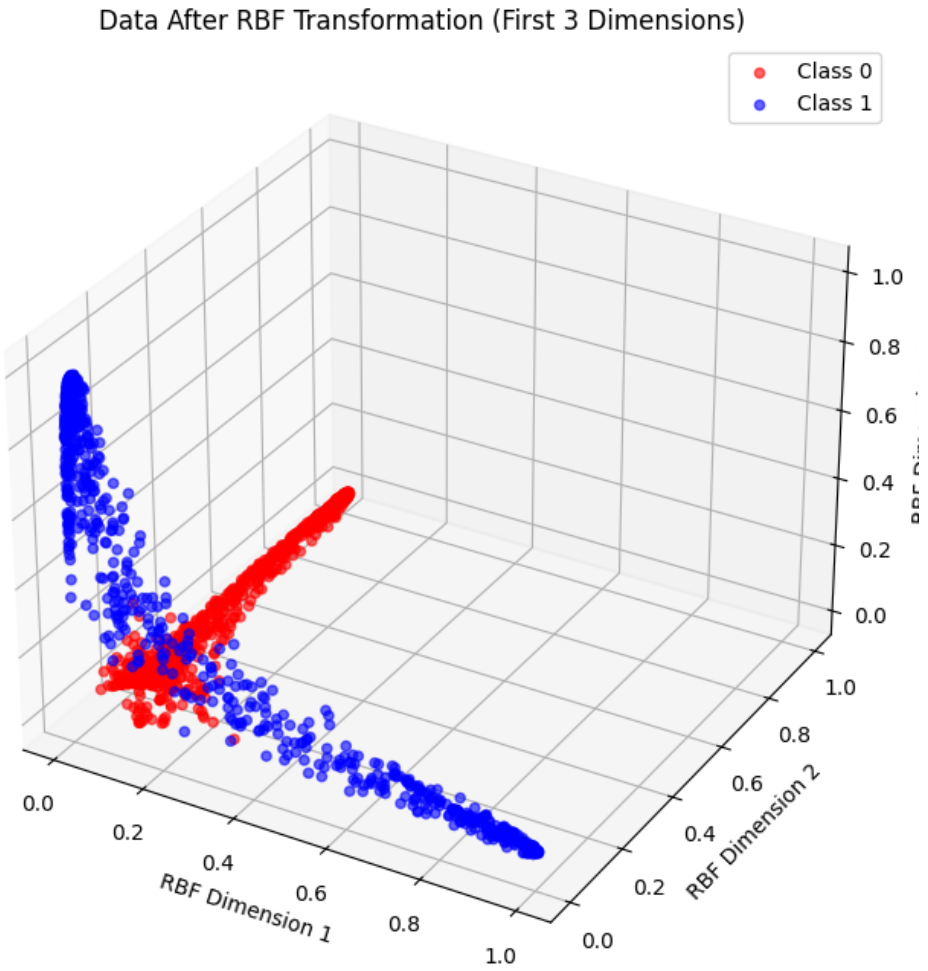


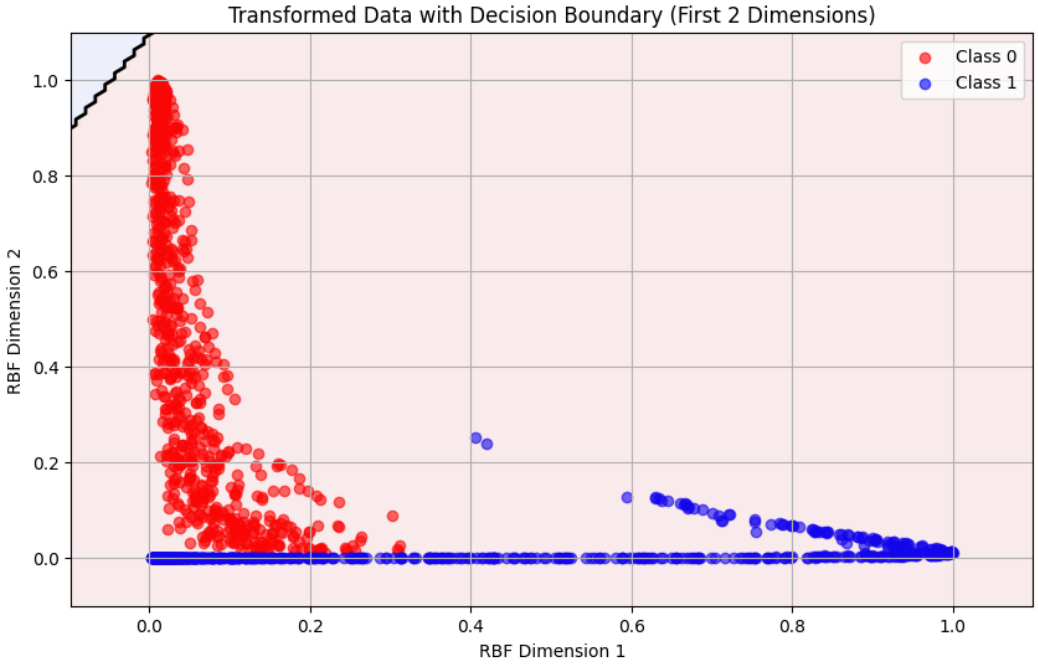



Decision Boundary (Training Data)
Class 0 (Lower Moon)
Class 1 (Upper Moon, Right-Shifted)
RBF Centers
X1
X2


Decision Boundary (Testing Data)
Class 0 (Lower Moon)
Class 1 (Upper Moon, Right-Shifted)
RBF Centers
X1
X2


RBF Activations and Centers
X1
X2


Data After RBF Transformation (First 3 Dimensions)
Class 0
Class 1
RBF Dimension 1
RBF Dimension 2
0.0
0.2
0.4
0.6
0.8
1.0


Transformed Data with Decision Boundary (First 2 Dimensions)
Class 0
Class 1
RBF Dimension 1
RBF Dimension 2
0.0
0.2
0.4
0.6
0.8
1.0

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.spatial.distance import cdist
from mpl_toolkits.mplot3d import Axes3D

class RBFNetwork(BaseEstimator, ClassifierMixin):
    def __init__(self, n_rbf=20, sigma=1.0, lr=0.01, epochs=1000):
        self.n_rbf = n_rbf       # Number of RBF neurons
        self.sigma = sigma       # Width of RBFs
        self.lr = lr             # Learning rate
        self.epochs = epochs     # Training epochs
        self.centers = None      # RBF centers
        self.weights = None      # Output layer weights
        self.bias = None         # Output layer bias

    def _rbf(self, X, center):
        return np.exp(-self.sigma * np.sum((X - center)**2, axis=1))

    def fit(self, X, y):
        # Step 1: Find RBF centers using K-means clustering
        kmeans = KMeans(n_clusters=self.n_rbf, random_state=42)
        kmeans.fit(X)
        self.centers = kmeans.cluster_centers_

        # Step 2: Calculate RBF activations
        rbf_activations = np.zeros((len(X), self.n_rbf))
        for i, center in enumerate(self.centers):
            rbf_activations[:, i] = self._rbf(X, center)

        # Step 3: Train output layer weights (using gradient descent)
        n_samples, n_features = rbf_activations.shape
        self.weights = np.random.randn(n_features)
        self.bias = np.random.randn(1)

        # Convert y to {-1, 1} for perceptron learning
        y_ = np.where(y == 0, -1, 1)

        for _ in range(self.epochs):
            # Forward pass
            outputs = np.dot(rbf_activations, self.weights) + self.bias
            predictions = np.where(outputs >= 0, 1, -1)

            # Compute error
            errors = y_ - predictions

            # Update weights
```

```
            self.weights += self.lr * np.dot(rbf_activations.T, errors)
            self.bias += self.lr * np.sum(errors)

        return self

    def predict(self, X):
        rbf_activations = np.zeros((len(X), self.n_rbf))
        for i, center in enumerate(self.centers):
            rbf_activations[:, i] = self._rbf(X, center)

        outputs = np.dot(rbf_activations, self.weights) + self.bias
        return np.where(outputs >= 0, 1, 0)

    def predict_from_activations(self, rbf_activations):
        outputs = np.dot(rbf_activations, self.weights) + self.bias
        return np.where(outputs >= 0, 1, 0)

# 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 upper moon (shifted right)
    theta = np.linspace(0, np.pi, n_samples//2)
    upper_x = radius * np.cos(theta) + radius/2 + shift
    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 RBF network
rbf_net = RBFNetwork(n_rbf=50, sigma=1.0, lr=0.01, epochs=1000)
rbf_net.fit(X_train, y_train)

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

    # Plot RBF centers
    if hasattr(model, 'centers'):
        plt.scatter(model.centers[:, 0], model.centers[:, 1],
                    color='black', marker='x', s=100, linewidths=2,
                    label='RBF Centers')

    plt.title(title)
    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.legend()
    plt.grid(True)

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

# 9. Plot RBF activations (for visualization)
def plot_rbf_activations(model, X, y, title):
    if not hasattr(model, 'centers'):
        return

    plt.figure(figsize=(10, 6))

    # Create a grid for visualization
    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.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))
```

```
    # Plot each RBF activation
    for i, center in enumerate(model.centers):
        grid_points = np.c_[xx.ravel(), yy.ravel()]
        activations = model._rbf(grid_points, center)
        activations = activations.reshape(xx.shape)

        # Plot contour of this RBF
        cs = plt.contour(xx, yy, activations, levels=[0.5],
                         colors=[plt.cm.jet(i/len(model.centers))])

    plt.scatter(X[y==0, 0], X[y==0, 1], color='red', alpha=0.2)
    plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', alpha=0.2)
    plt.scatter(model.centers[:, 0], model.centers[:, 1],
                color='black', marker='x', s=100, linewidths=2)
    plt.title(title)
    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.grid(True)

plot_rbf_activations(rbf_net, X_train, y_train, "RBF Activations and Centers")
plt.show()

# 10. Plot the data after RBF transformation (first 3 dimensions for visualization)
def plot_transformed_data(X, y, model, title):
    if not hasattr(model, 'centers'):
        return

    # Transform the data using RBF activations
    rbf_activations = np.zeros((len(X), model.n_rbf))
    for i, center in enumerate(model.centers):
        rbf_activations[:, i] = model._rbf(X, center)

    # For visualization, we'll plot the first 3 dimensions
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Plot class 0
    ax.scatter(rbf_activations[y==0, 0],
               rbf_activations[y==0, 1],
               rbf_activations[y==0, 2],
               color='red', label='Class 0', alpha=0.6)

    # Plot class 1
    ax.scatter(rbf_activations[y==1, 0],
               rbf_activations[y==1, 1],
               rbf_activations[y==1, 2],
               color='blue', label='Class 1', alpha=0.6)

    ax.set_title(title)
    ax.set_xlabel("RBF Dimension 1")
    ax.set_ylabel("RBF Dimension 2")
```

```
    ax.set_zlabel("RBF Dimension 3")
    ax.legend()
    plt.show()

# Plot transformed training data
plot_transformed_data(X_train, y_train, rbf_net, "Data After RBF Transformation (First 3
Dimensions)")

# 11. Plot the transformed data with the separating hyperplane (first 2 dimensions)
def plot_transformed_decision_boundary(X, y, model, title):
    if not hasattr(model, 'centers'):
        return

    # Transform the data using RBF activations
    rbf_activations = np.zeros((len(X), model.n_rbf))
    for i, center in enumerate(model.centers):
        rbf_activations[:, i] = model._rbf(X, center)

    # We'll visualize just the first 2 dimensions
    dim1, dim2 = 0, 1  # Can change these indices to view different dimensions

    plt.figure(figsize=(10, 6))

    # Plot the transformed data points
    plt.scatter(rbf_activations[y==0, dim1], rbf_activations[y==0, dim2],
                color='red', label='Class 0', alpha=0.6)
    plt.scatter(rbf_activations[y==1, dim1], rbf_activations[y==1, dim2],
                color='blue', label='Class 1', alpha=0.6)

    # Plot the decision boundary (works well for 2D visualization)
    if model.weights is not None:
        # Create a grid for decision boundary
        x_min, x_max = rbf_activations[:, dim1].min() - 0.1, rbf_activations[:, dim1].max() + 0.1
        y_min, y_max = rbf_activations[:, dim2].min() - 0.1, rbf_activations[:, dim2].max() + 0.1
        xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                             np.linspace(y_min, y_max, 100))

        # For the decision boundary, we need to consider all dimensions but we'll
        # set other dimensions to their mean values for this 2D visualization
        grid_points = np.zeros((xx.ravel().shape[0], model.n_rbf))
        grid_points[:, dim1] = xx.ravel()
        grid_points[:, dim2] = yy.ravel()

        # Set other dimensions to their mean values
        for dim in range(model.n_rbf):
            if dim not in [dim1, dim2]:
                grid_points[:, dim] = np.mean(rbf_activations[:, dim])

        # Make predictions
        Z = model.predict_from_activations(grid_points)
        Z = Z.reshape(xx.shape)
```

```
        # Plot decision boundary
        plt.contour(xx, yy, Z, levels=[0.5], colors='black', linewidths=2)
        plt.contourf(xx, yy, Z, alpha=0.1, cmap='coolwarm')

    plt.title(title)
    plt.xlabel(f"RBF Dimension {dim1+1}")
    plt.ylabel(f"RBF Dimension {dim2+1}")
    plt.legend()
    plt.grid(True)
    plt.show()

# Plot transformed data with decision boundary
plot_transformed_decision_boundary(X_train, y_train, rbf_net,
                                   "Transformed Data with Decision Boundary (First 2 Dimensions)")
```