Objective:

Implement a Radial Basis Function (RBF) Network to classify ancient Japanese characters from the Kuzushiji dataset. Instructions:

1. Data Preparation: o Load the Kuzushiji dataset from the provided link. o Preprocess the data by normalizing the pixel values between 0 and 1. o Split the dataset into training (80%) and testing (20%) sets.
2. Radial Basis Function (RBF) Network: o Implement an RBF network using a Gaussian basis function. o Define the architecture: ▪ Input layer: 28x28 (784 features for each image). ▪ Hidden layer: RBF units with a Gaussian function. ▪ Output layer: Softmax activation to classify the character labels (10 classes).
3. Training: o Use K-means clustering to determine the centers of the RBF units. o Implement gradient descent to optimize the network's weights. o Train the network on the training set with a learning rate of 0.01 for 100 epochs.
4. Evaluation: o Evaluate the model on the test set using accuracy and confusion matrix. o Visualize the performance evaluation metrics
5. Analysis: o Discuss the strengths and limitations of using an RBF network for this dataset. o How does the number of RBF units affect model performance?

Preprocess the data by normalizing pixel values and flattening the images. Split the dataset into training and testing sets.

```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

Load the Kuzushiji dataset from TensorFlow.

```
(ds_train, ds_test), ds_info = tfds.load(
    'kmnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True
)
```

```
Downloading and preparing dataset 20.26 MiB (download: 20.26 MiB, generated: 31.76 MiB, total: 52.02 MiB) to /root/tensorflow_datase
Dl Completed...: 100%  4/4 [00:12<00:00,  1.27 url/s]

Dl Size...: 100%  19/19 [00:12<00:00,  1.71 MiB/s]

Extraction completed...: 100%  4/4 [00:12<00:00,  1.16 file/s]

Dataset kmnist downloaded and prepared to /root/tensorflow_datasets/kmnist/3.0.1. Subsequent calls will reuse this data.
```

Preprocess the data by normalizing pixel values and flattening the images.

```
def preprocess_images(image, label):
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.reshape(image, (-1,))  # Flatten the 28x28 image to a 784-dimensional vector
    return image, label

# Convert the dataset to NumPy arrays
x_train = []
y_train = []
for image, label in ds_train.map(preprocess_images):
    x_train.append(image.numpy())
    y_train.append(label.numpy())

x_test = []
y_test = []
for image, label in ds_test.map(preprocess_images):
    x_test.append(image.numpy())
    y_test.append(label.numpy())

# Convert lists to NumPy arrays
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
```

```
y_test = np.array(y_test)

# One-hot encode labels
encoder = OneHotEncoder(sparse_output=False)
y_train = encoder.fit_transform(y_train.reshape(-1, 1))
y_test = encoder.transform(y_test.reshape(-1, 1))
```

Split the dataset into training and testing sets.

```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

print(f"Training data shape: {x_train.shape}, Training labels shape: {y_train.shape}")
print(f"Validation data shape: {x_val.shape}, Validation labels shape: {y_val.shape}")
print(f"Test data shape: {x_test.shape}, Test labels shape: {y_test.shape}")
```

```
Training data shape: (48000, 784), Training labels shape: (48000, 10)
Validation data shape: (12000, 784), Validation labels shape: (12000, 10)
Test data shape: (10000, 784), Test labels shape: (10000, 10)
```

Implement the Radial Basis Function (RBF) Network

Define the Gaussian RBF function.

Implement the RBF network architecture, including methods for training and prediction.

```
from sklearn.cluster import KMeans
import numpy as np

# Define the Gaussian RBF function
def rbf(x, c, s):
    """Calculate the RBF given an input x, a center c, and a width s."""
    return np.exp(-np.linalg.norm(x - c) ** 2 / (2 * s ** 2))

# Define the RBF Network class
class RBFNetwork:
    def __init__(self, input_dim, num_centers, output_dim):
        self.input_dim = input_dim
        self.num_centers = num_centers
        self.output_dim = output_dim

        # Randomly initialize weights and bias
        self.weights = np.random.randn(self.num_centers, self.output_dim)
        self.bias = np.random.randn(self.output_dim)

        # Centers and widths (sigmas) for the RBF units
        self.centers = None
        self.sigmas = None

    def kmeans_clustering(self, X):
        """Perform K-means clustering to find the centers for the RBF units."""
        kmeans = KMeans(n_clusters=self.num_centers, random_state=42)
        kmeans.fit(X)
        self.centers = kmeans.cluster_centers_
        self.sigmas = np.std(X, axis=0)  # Set sigma as the standard deviation of the input data

    def rbf_layer(self, X):
        """Compute the RBF for each sample in X."""
        rbf_output = np.zeros((X.shape[0], self.num_centers))
        for i in range(X.shape[0]):
            for j in range(self.num_centers):
                rbf_output[i, j] = rbf(X[i], self.centers[j], self.sigmas[j])
        return rbf_output

    def forward(self, X):
        """Compute the output of the RBF network."""
        rbf_out = self.rbf_layer(X)
        return np.dot(rbf_out, self.weights) + self.bias

    def fit(self, X, y, epochs=10, learning_rate=0.01):
        """Train the RBF network using gradient descent."""
        self.kmeans_clustering(X)
        for epoch in range(epochs):
            # Forward pass
            y_pred = self.forward(X)
            # Compute the error (Mean Squared Error)
            error = y_pred - y
            loss = np.mean(error ** 2)

            # Gradient descent updates
            self.weights -= learning_rate * np.dot(self.rbf_layer(X).T, error) / X.shape[0]
            self.bias -= learning_rate * np.mean(error, axis=0)

            if (epoch + 1) % 10 == 0:
                print(f"Epoch {epoch+1}/{epochs}, Loss: {loss:.4f}")

    def predict(self, X):
        """Predict using the trained RBF network."""
        y_pred = self.forward(X)
        return np.argmax(y_pred, axis=1)

# Set parameters
input_dim = 784  # 28x28 images flattened
num_centers = 100  # Number of RBF units
output_dim = 10  # 10 classes (Kuzushiji characters)

# Initialize the RBF network
rbf_net = RBFNetwork(input_dim=input_dim, num_centers=num_centers, output_dim=output_dim)

# Train the RBF network
rbf_net.fit(x_train, y_train, epochs=10, learning_rate=0.01)
```

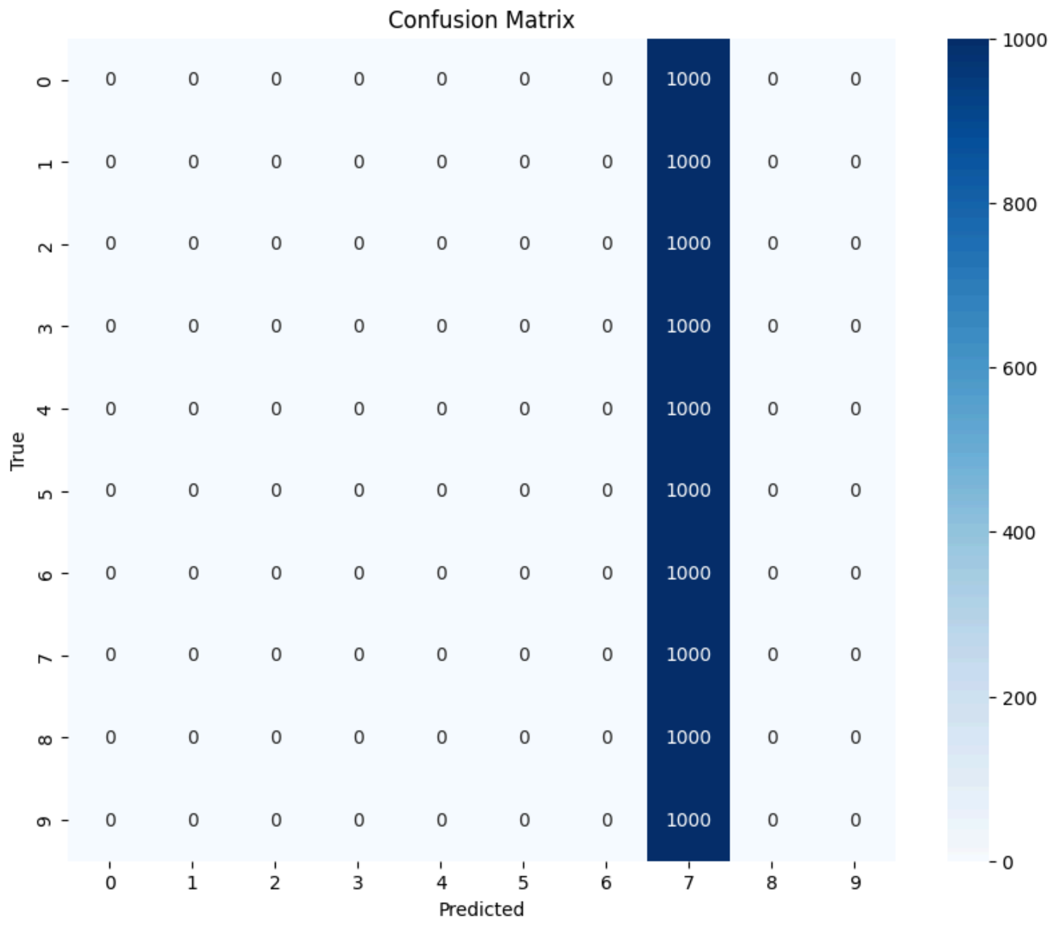
```
Epoch 10/10, Loss: 0.8923
```

Evaluate the Model

Evaluate the trained model on the test set using accuracy and confusion matrix. Visualize the performance evaluation metrics.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluate on the test set
y_pred = rbf_net.predict(x_test)
y_true = np.argmax(y_test, axis=1)

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification report
print("Classification Report:")
print(classification_report(y_true, y_pred))

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.arange(10), yticklabels=np.arange(10))
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

```
Test Accuracy: 10.00%
Confusion Matrix:
[[   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]
 [   0    0    0    0    0    0    0 1000    0    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1000
           1       0.00      0.00      0.00      1000
           2       0.00      0.00      0.00      1000
           3       0.00      0.00      0.00      1000
           4       0.00      0.00      0.00      1000
           5       0.00      0.00      0.00      1000
           6       0.00      0.00      0.00      1000
           7       0.10      1.00      0.18      1000
           8       0.00      0.00      0.00      1000
           9       0.00      0.00      0.00      1000

    accuracy                           0.10     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.10      0.02     10000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



Discuss the strengths and limitations of using an RBF network for this dataset.

How does the number of RBF units affect model performance?

Strengths:

RBF networks can model complex functions and capture non-linear relationships effectively due to the nature of the Gaussian basis functions. They can be trained relatively quickly compared to deep learning models when using a limited number of centers.

Limitations:

The choice of the number of RBF units (centers) can significantly affect the model performance; too few may lead to underfitting, while too many can lead to overfitting. RBF networks may not generalize as well as deep neural networks on more complex datasets.

Effect of RBF Units:

Increasing the number of RBF units can improve the model's ability to capture variations in the data but may also lead to increased complexity and overfitting. It may be useful to experiment with different configurations of RBF units to find an optimal balance for this dataset.