1. Data Preparation

```
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Kuzushiji-MNIST dataset
ds_train, ds_test = tfds.load('kmnist', split=['train', 'test'],
as_supervised=True)

def preprocess(images, labels):
    images = tf.cast(images, tf.float32) / 255.0  # Normalize pixel
values between 0 and 1
    labels = tf.cast(labels, tf.int32)
    return images, labels

# Apply the preprocessing function to the dataset
ds_train = ds_train.map(preprocess).batch(1024)
ds_test = ds_test.map(preprocess).batch(1024)

# Convert dataset to NumPy arrays
X_train = []
y_train = []
for image_batch, label_batch in ds_train:
    X_train.append(image_batch.numpy())
    y_train.append(label_batch.numpy())

X_train = np.concatenate(X_train, axis=0).reshape(-1, 28*28)
y_train = np.concatenate(y_train, axis=0)

X_test = []
y_test = []
for image_batch, label_batch in ds_test:
    X_test.append(image_batch.numpy())
    y_test.append(label_batch.numpy())

X_test = np.concatenate(X_test, axis=0).reshape(-1, 28*28)
y_test = np.concatenate(y_test, axis=0)

# Normalize pixel values between 0 and 1 using MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
/Users/riyashah/Library/Python/3.9/lib/python/site-packages/tqdm/
auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
2024-10-18 13:11:05.734244: W
external/local_tsl/tsl/platform/cloud/google_auth_provider.cc:184] All
attempts to get a Google authentication bearer token failed, returning
an empty token. Retrieving token from files failed with "NOT_FOUND:
Could not locate the credentials file.". Retrieving token from GCE
failed with "FAILED_PRECONDITION: Error executing an HTTP request:
libcurl code 6 meaning 'Couldn't resolve host name', error details:
Could not resolve host: metadata.google.internal".

Downloading and preparing dataset 20.26 MiB (download: 20.26 MiB,
generated: 31.76 MiB, total: 52.02 MiB) to
/Users/riyashah/tensorflow_datasets/kmnist/3.0.1...

Dl Completed...: 0 url [00:00, ? url/s]
Dl Completed...:   0%|          | 0/1 [00:00<?, ? url/s]
Dl Completed...:   0%|          | 0/2 [00:00<?, ? url/s]
Dl Completed...:   0%|          | 0/3 [00:00<?, ? url/s]
Dl Completed...:   0%|          | 0/4 [00:00<?, ? url/s]
Dl Completed...:   0%|          | 0/4 [00:01<?, ? url/s]
Dl Completed...:   0%|          | 0/4 [00:01<?, ? url/s]
Dl Completed...:  25%|██        | 1/4 [00:01<00:03,  1.07s/ url]
Dl Completed...:  25%|██        | 1/4 [00:01<00:03,  1.07s/ url]
Dl Completed...:  25%|██        | 1/4 [00:01<00:03,  1.07s/ url]
pleted...:  25%|██        | 1/4 [00:01<00:03,  1.07s/ url]
Dl Completed...:  25%|██        | 1/4 [00:01<00:03,  1.07s/ url]
Dl Completed...:  50%|████      | 2/4 [00:01<00:01,  1.84 url/s]
Dl Completed...:  50%|████      | 2/4 [00:01<00:01,  1.84 url/s]
pleted...:  50%|████      | 2/4 [00:01<00:01,  1.84 url/s]
Dl Completed...:  50%|████      | 2/4 [00:02<00:01,  1.84 url/s]
Dl Completed...:  50%|████      | 2/4 [00:02<00:01,  1.84 url/s]
Dl Completed...:  50%|████      | 2/4 [00:02<00:01,  1.84 url/s]
Dl Completed...:  50%|████      | 2/4 [00:02<00:01,  1.84 url/s]
Dl Completed...:  50%|████      | 2/4 [00:02<00:01,  1.84 url/s]
Dl Completed...:  75%|██████    | 3/4 [00:03<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:03<00:01,  1.14s/ url]
pleted...:  75%|██████    | 3/4 [00:03<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:03<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:03<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:04<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:05<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:06<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:06<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:07<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:08<00:01,  1.14s/ url]
Dl Completed...:  75%|██████    | 3/4 [00:09<00:01,  1.14s/ url]
```

```
Dl Completed...:  75%|███████   | 3/4 [00:10<00:01,  1.14s/ url]
Dl Completed...:  75%|███████   | 3/4 [00:11<00:01,  1.14s/ url]
Dl Completed...:  75%|███████   | 3/4 [00:12<00:01,  1.14s/ url]
Dl Completed...:  75%|███████   | 3/4 [00:13<00:01,  1.14s/ url]
Dl Completed...:  75%|███████   | 3/4 [00:14<00:01,  1.14s/ url]
Dl Completed...: 100%|████████| 4/4 [00:14<00:00,  5.12s/ url]
Dl Completed...: 100%|████████| 4/4 [00:14<00:00,  5.12s/ url]
pleted...: 100%|████████| 4/4 [00:14<00:00,  5.12s/ url]
Extraction completed...: 100%|████████| 4/4 [00:14<00:00,  3.62s/
file]
Dl Size...: 100%|████████| 19/19 [00:14<00:00,  1.31 MiB/s]
Dl Completed...: 100%|████████| 4/4 [00:14<00:00,  3.62s/ url]


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

2024-10-18 13:11:32.480460: I
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
2024-10-18 13:11:32.748094: I
tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
```

2. Radial Basis Function (RBF) Network Implementation

```python
def gaussian_rbf(x, center, sigma):
    distance = np.linalg.norm(x - center)
    return np.exp(- (distance ** 2) / (2 * (sigma ** 2)))

class RBFNetwork:
    def __init__(self, k, input_dim, output_dim, sigma=1.0):
        self.k = k  # Number of RBF units
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.sigma = sigma

        # Initialize the RBF centers
        self.centers = np.random.randn(k, input_dim)

        # Initialize weights between RBF units and output layer
        self.weights = np.random.randn(k, output_dim)

    def fit(self, X, y, epochs=100, learning_rate=0.01):
        # Use KMeans to find centers for the RBF units
        kmeans = KMeans(n_clusters=self.k, random_state=42)
        kmeans.fit(X)
        self.centers = kmeans.cluster_centers_
```

```
        # Convert y to one-hot encoding
        y_one_hot = np.zeros((y.size, self.output_dim))
        y_one_hot[np.arange(y.size), y] = 1

        # Gradient descent to optimize weights
        for epoch in range(epochs):
            for i in range(X.shape[0]):
                # Compute the activation for each RBF unit
                rbf_outputs = np.array([gaussian_rbf(X[i], center,
self.sigma) for center in self.centers])

                # Compute output (softmax layer)
                output = self._softmax(np.dot(rbf_outputs,
self.weights))

                # Compute error
                error = y_one_hot[i] - output

                # Gradient descent weight update
                self.weights += learning_rate * np.outer(rbf_outputs,
error)

            # Optionally print loss or accuracy at intervals

    def predict(self, X):
        y_pred = []
        for i in range(X.shape[0]):
            rbf_outputs = np.array([gaussian_rbf(X[i], center,
self.sigma) for center in self.centers])
            output = self._softmax(np.dot(rbf_outputs, self.weights))
            y_pred.append(np.argmax(output))
        return np.array(y_pred)

    def _softmax(self, x):
        exp_x = np.exp(x - np.max(x))
        return exp_x / exp_x.sum(axis=0)
```

3. Training the RBF Network

```
# Define the number of RBF units (can be tuned)
k = 50  # Number of RBF units

# Initialize the RBF network
rbf_net = RBFNetwork(k=k, input_dim=X_train.shape[1], output_dim=10,
sigma=1.0)

# Train the network
rbf_net.fit(X_train, y_train, epochs=100, learning_rate=0.01)
```

4. Evaluation

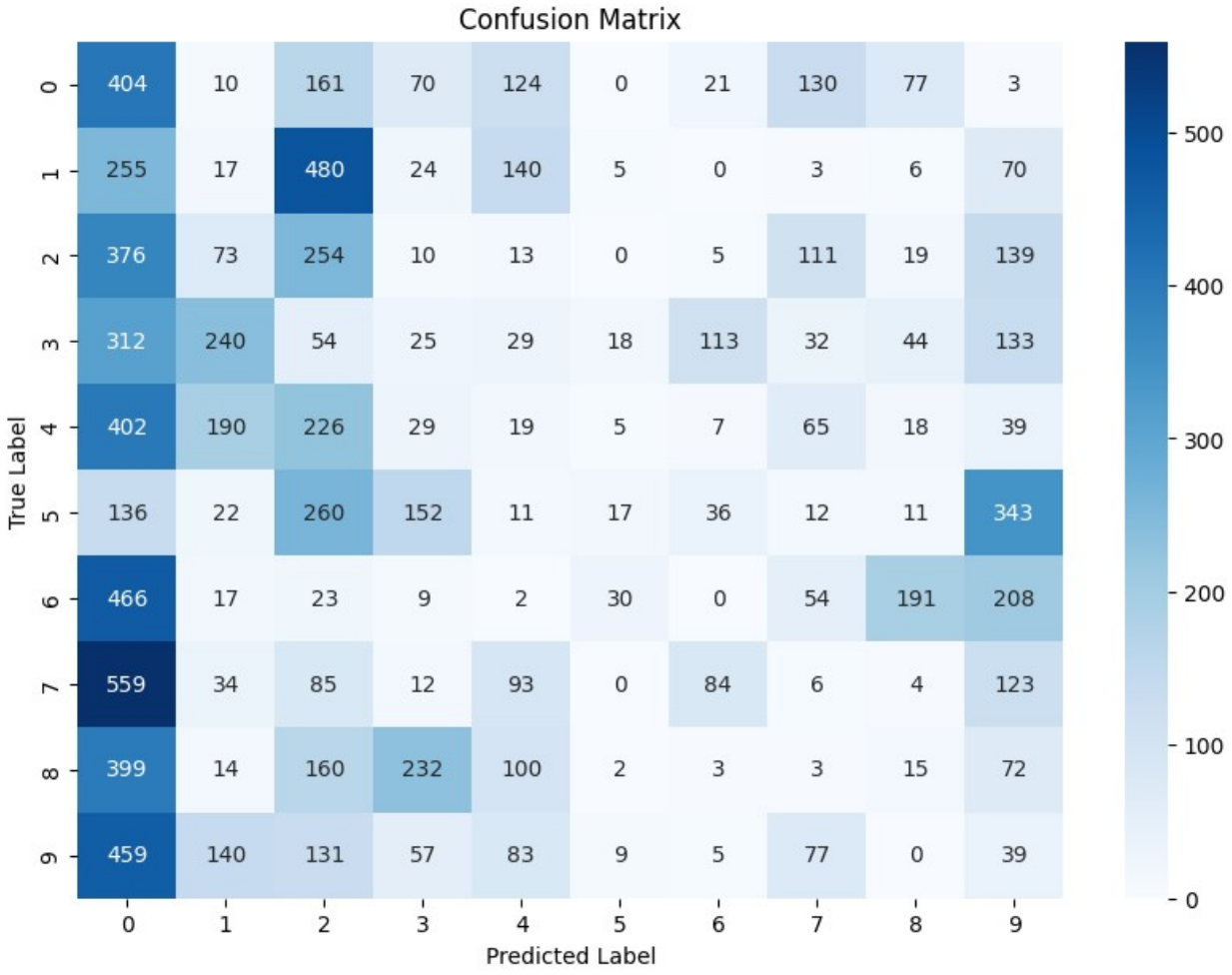
```python
# Make predictions on the test set
y_pred = rbf_net.predict(X_test)

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

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 7.96%
```



5. Analysis

Strengths of the RBF Network:

The use of localized Gaussian basis functions makes RBF networks good for handling classification tasks where data clusters are spatially distinct. The K-means clustering approach for determining RBF centers is a natural way to capture the structure of the dataset.

Limitations:

RBF networks can be computationally expensive, especially with a high number of RBF units. Finding the optimal number of RBF units and the right sigma can be challenging. Performance is sensitive to the choice of hyperparameters like the number of RBF units and the learning rate.

Effect of the number of RBF units:

Too few RBF units can result in underfitting, while too many can lead to overfitting. You can experiment with different values of k (RBF units) to observe the impact on performance.