2347126-lab4

October 25, 2024

1 Lab - 4

Name: L Vinay Kumar Reddy
Reg No. 2347126

1.1 1. Import datset and Data Preparation:

[17]:
```
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.cluster import KMeans
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
import tensorflow as tf

# Function to compute RBF (Gaussian basis function)
def rbf(x, centers, sigma):
    return np.exp(-np.linalg.norm(x - centers, axis=1)**2 / (2 * sigma**2))

# Load and preprocess the dataset
train_images = np.load("k49-train-imgs.npz")['arr_0']
train_labels = np.load("k49-train-labels.npz")['arr_0']
test_images = np.load("k49-test-imgs.npz")['arr_0']
test_labels = np.load("k49-test-labels.npz")['arr_0']

# Normalize the pixel values between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0
```

1.2 2. Radial Basis Function (RBF) Network:

[18]:
```
# Reshape images from (28, 28) to (784,)
train_images_flat = train_images.reshape(train_images.shape[0], -1)
test_images_flat = test_images.reshape(test_images.shape[0], -1)

# One-hot encode the labels
encoder = OneHotEncoder(sparse_output=False)
train_labels_encoded = encoder.fit_transform(train_labels.reshape(-1, 1))
```

```
test_labels_encoded = encoder.transform(test_labels.reshape(-1, 1))

# Use KMeans to find the centers for the RBF units
k = 100  # Number of RBF centers
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(train_images_flat)
centers = kmeans.cluster_centers_

# Calculate sigma based on average distance between cluster centers
distances = np.linalg.norm(centers[:, np.newaxis] - centers[np.newaxis, :],␣
 ↪axis=2)
sigma = np.mean(distances)

# Transform the input data using the RBF kernel
train_images_rbf = np.array([rbf(x, centers, sigma) for x in train_images_flat])
test_images_rbf = np.array([rbf(x, centers, sigma) for x in test_images_flat])

# Define the RBF network architecture
model = Sequential([
    Input(shape=(k,)),  # Input is transformed via RBF with k centers
    Dense(128, activation='relu'),  # Additional dense layer to increase␣
 ↪learning capacity
    Dense(train_labels_encoded.shape[1], activation='softmax')  # Output layer␣
 ↪with softmax for classification
])
```

1.3 3. Training:

[19]:
```
# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images_rbf, train_labels_encoded, epochs=100,␣
 ↪batch_size=64,
                    validation_data=(test_images_rbf, test_labels_encoded))
```

```
Epoch 1/100
3631/3631 [==============================] - 7s 2ms/step - loss: 2.5169 -
accuracy: 0.3689 - val_loss: 2.4417 - val_accuracy: 0.3915
Epoch 2/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.8176 -
accuracy: 0.5289 - val_loss: 2.2357 - val_accuracy: 0.4432
Epoch 3/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.6462 -
accuracy: 0.5717 - val_loss: 2.1227 - val_accuracy: 0.4681
```

```
Epoch 4/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.5483 -
accuracy: 0.5950 - val_loss: 2.0531 - val_accuracy: 0.4898
Epoch 5/100
3631/3631 [==============================] - 6s 2ms/step - loss: 1.4759 -
accuracy: 0.6123 - val_loss: 1.9879 - val_accuracy: 0.5004
Epoch 6/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.4190 -
accuracy: 0.6266 - val_loss: 1.9421 - val_accuracy: 0.5155
Epoch 7/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.3724 -
accuracy: 0.6382 - val_loss: 1.8935 - val_accuracy: 0.5291
Epoch 8/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.3329 -
accuracy: 0.6480 - val_loss: 1.8543 - val_accuracy: 0.5333
Epoch 9/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.2983 -
accuracy: 0.6562 - val_loss: 1.8120 - val_accuracy: 0.5448
Epoch 10/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.2708 -
accuracy: 0.6623 - val_loss: 1.7977 - val_accuracy: 0.5495
Epoch 11/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.2454 -
accuracy: 0.6692 - val_loss: 1.7625 - val_accuracy: 0.5530
Epoch 12/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.2238 -
accuracy: 0.6739 - val_loss: 1.7512 - val_accuracy: 0.5610
Epoch 13/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.2011 -
accuracy: 0.6793 - val_loss: 1.7194 - val_accuracy: 0.5636
Epoch 14/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.1835 -
accuracy: 0.6838 - val_loss: 1.7006 - val_accuracy: 0.5708
Epoch 15/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.1664 -
accuracy: 0.6873 - val_loss: 1.7044 - val_accuracy: 0.5731
Epoch 16/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.1502 -
accuracy: 0.6916 - val_loss: 1.6661 - val_accuracy: 0.5771
Epoch 17/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.1357 -
accuracy: 0.6949 - val_loss: 1.6459 - val_accuracy: 0.5840
Epoch 18/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.1228 -
accuracy: 0.6977 - val_loss: 1.6289 - val_accuracy: 0.5880
Epoch 19/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.1100 -
accuracy: 0.7011 - val_loss: 1.6250 - val_accuracy: 0.5876
```

```
Epoch 20/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0979 -
accuracy: 0.7048 - val_loss: 1.6024 - val_accuracy: 0.5930
Epoch 21/100
3631/3631 [==============================] - 6s 2ms/step - loss: 1.0870 -
accuracy: 0.7076 - val_loss: 1.6016 - val_accuracy: 0.5972
Epoch 22/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0759 -
accuracy: 0.7097 - val_loss: 1.5895 - val_accuracy: 0.5990
Epoch 23/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0668 -
accuracy: 0.7122 - val_loss: 1.5676 - val_accuracy: 0.6024
Epoch 24/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0580 -
accuracy: 0.7146 - val_loss: 1.5804 - val_accuracy: 0.6015
Epoch 25/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0499 -
accuracy: 0.7168 - val_loss: 1.5465 - val_accuracy: 0.6065
Epoch 26/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0401 -
accuracy: 0.7191 - val_loss: 1.5770 - val_accuracy: 0.6029
Epoch 27/100
3631/3631 [==============================] - 6s 2ms/step - loss: 1.0334 -
accuracy: 0.7209 - val_loss: 1.5349 - val_accuracy: 0.6092
Epoch 28/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0249 -
accuracy: 0.7235 - val_loss: 1.5472 - val_accuracy: 0.6097
Epoch 29/100
3631/3631 [==============================] - 6s 2ms/step - loss: 1.0174 -
accuracy: 0.7248 - val_loss: 1.5162 - val_accuracy: 0.6108
Epoch 30/100
3631/3631 [==============================] - 6s 2ms/step - loss: 1.0103 -
accuracy: 0.7274 - val_loss: 1.5202 - val_accuracy: 0.6135
Epoch 31/100
3631/3631 [==============================] - 7s 2ms/step - loss: 1.0042 -
accuracy: 0.7289 - val_loss: 1.4996 - val_accuracy: 0.6182
Epoch 32/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9972 -
accuracy: 0.7299 - val_loss: 1.5007 - val_accuracy: 0.6211
Epoch 33/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9902 -
accuracy: 0.7325 - val_loss: 1.5068 - val_accuracy: 0.6182
Epoch 34/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9861 -
accuracy: 0.7333 - val_loss: 1.4800 - val_accuracy: 0.6236
Epoch 35/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9792 -
accuracy: 0.7352 - val_loss: 1.4676 - val_accuracy: 0.6240
```

```
Epoch 36/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9731 -
accuracy: 0.7365 - val_loss: 1.4970 - val_accuracy: 0.6149
Epoch 37/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9681 -
accuracy: 0.7380 - val_loss: 1.4867 - val_accuracy: 0.6232
Epoch 38/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.9631 -
accuracy: 0.7392 - val_loss: 1.4544 - val_accuracy: 0.6264
Epoch 39/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9578 -
accuracy: 0.7403 - val_loss: 1.4668 - val_accuracy: 0.6264
Epoch 40/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9536 -
accuracy: 0.7413 - val_loss: 1.4395 - val_accuracy: 0.6317
Epoch 41/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9472 -
accuracy: 0.7427 - val_loss: 1.4386 - val_accuracy: 0.6324
Epoch 42/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9436 -
accuracy: 0.7437 - val_loss: 1.4581 - val_accuracy: 0.6288
Epoch 43/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9383 -
accuracy: 0.7449 - val_loss: 1.4320 - val_accuracy: 0.6310
Epoch 44/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9340 -
accuracy: 0.7459 - val_loss: 1.4286 - val_accuracy: 0.6318
Epoch 45/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.9291 -
accuracy: 0.7479 - val_loss: 1.4208 - val_accuracy: 0.6350
Epoch 46/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9266 -
accuracy: 0.7482 - val_loss: 1.4186 - val_accuracy: 0.6347
Epoch 47/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.9217 -
accuracy: 0.7489 - val_loss: 1.3783 - val_accuracy: 0.6451
Epoch 48/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9176 -
accuracy: 0.7500 - val_loss: 1.3903 - val_accuracy: 0.6418
Epoch 49/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9146 -
accuracy: 0.7502 - val_loss: 1.3834 - val_accuracy: 0.6404
Epoch 50/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9111 -
accuracy: 0.7525 - val_loss: 1.4254 - val_accuracy: 0.6373
Epoch 51/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.9067 -
accuracy: 0.7524 - val_loss: 1.4227 - val_accuracy: 0.6350
```

```
Epoch 52/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9043 -
accuracy: 0.7532 - val_loss: 1.3813 - val_accuracy: 0.6463
Epoch 53/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.9007 -
accuracy: 0.7550 - val_loss: 1.3873 - val_accuracy: 0.6422
Epoch 54/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8969 -
accuracy: 0.7551 - val_loss: 1.3826 - val_accuracy: 0.6430
Epoch 55/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8933 -
accuracy: 0.7562 - val_loss: 1.3794 - val_accuracy: 0.6461
Epoch 56/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8906 -
accuracy: 0.7572 - val_loss: 1.3823 - val_accuracy: 0.6467
Epoch 57/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8880 -
accuracy: 0.7571 - val_loss: 1.3663 - val_accuracy: 0.6490
Epoch 58/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8852 -
accuracy: 0.7585 - val_loss: 1.3782 - val_accuracy: 0.6463
Epoch 59/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8827 -
accuracy: 0.7590 - val_loss: 1.3729 - val_accuracy: 0.6493
Epoch 60/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8791 -
accuracy: 0.7603 - val_loss: 1.3421 - val_accuracy: 0.6563
Epoch 61/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8763 -
accuracy: 0.7606 - val_loss: 1.3690 - val_accuracy: 0.6477
Epoch 62/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8751 -
accuracy: 0.7613 - val_loss: 1.4065 - val_accuracy: 0.6384
Epoch 63/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8731 -
accuracy: 0.7618 - val_loss: 1.3660 - val_accuracy: 0.6518
Epoch 64/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8694 -
accuracy: 0.7625 - val_loss: 1.3653 - val_accuracy: 0.6528
Epoch 65/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8678 -
accuracy: 0.7630 - val_loss: 1.3625 - val_accuracy: 0.6489
Epoch 66/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8656 -
accuracy: 0.7637 - val_loss: 1.3572 - val_accuracy: 0.6508
Epoch 67/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8636 -
accuracy: 0.7644 - val_loss: 1.3242 - val_accuracy: 0.6570
```

```
Epoch 68/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8601 -
accuracy: 0.7653 - val_loss: 1.3214 - val_accuracy: 0.6627
Epoch 69/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8582 -
accuracy: 0.7659 - val_loss: 1.3375 - val_accuracy: 0.6550
Epoch 70/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8561 -
accuracy: 0.7657 - val_loss: 1.3248 - val_accuracy: 0.6597
Epoch 71/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8544 -
accuracy: 0.7655 - val_loss: 1.3387 - val_accuracy: 0.6517
Epoch 72/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8520 -
accuracy: 0.7677 - val_loss: 1.3417 - val_accuracy: 0.6553
Epoch 73/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8504 -
accuracy: 0.7674 - val_loss: 1.3322 - val_accuracy: 0.6592
Epoch 74/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8475 -
accuracy: 0.7690 - val_loss: 1.3271 - val_accuracy: 0.6578
Epoch 75/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8465 -
accuracy: 0.7683 - val_loss: 1.3182 - val_accuracy: 0.6591
Epoch 76/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8456 -
accuracy: 0.7688 - val_loss: 1.3540 - val_accuracy: 0.6537
Epoch 77/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8426 -
accuracy: 0.7699 - val_loss: 1.3183 - val_accuracy: 0.6601
Epoch 78/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8416 -
accuracy: 0.7703 - val_loss: 1.2970 - val_accuracy: 0.6671
Epoch 79/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8385 -
accuracy: 0.7712 - val_loss: 1.3036 - val_accuracy: 0.6659
Epoch 80/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8379 -
accuracy: 0.7712 - val_loss: 1.3256 - val_accuracy: 0.6590
Epoch 81/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8365 -
accuracy: 0.7720 - val_loss: 1.3049 - val_accuracy: 0.6644
Epoch 82/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8339 -
accuracy: 0.7724 - val_loss: 1.3391 - val_accuracy: 0.6569
Epoch 83/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8326 -
accuracy: 0.7726 - val_loss: 1.2933 - val_accuracy: 0.6675
```

```
Epoch 84/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8320 -
accuracy: 0.7731 - val_loss: 1.2883 - val_accuracy: 0.6692
Epoch 85/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8293 -
accuracy: 0.7730 - val_loss: 1.3109 - val_accuracy: 0.6612
Epoch 86/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8287 -
accuracy: 0.7737 - val_loss: 1.2897 - val_accuracy: 0.6661
Epoch 87/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8256 -
accuracy: 0.7745 - val_loss: 1.2998 - val_accuracy: 0.6619
Epoch 88/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8257 -
accuracy: 0.7743 - val_loss: 1.2701 - val_accuracy: 0.6697
Epoch 89/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8239 -
accuracy: 0.7753 - val_loss: 1.2614 - val_accuracy: 0.6735
Epoch 90/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8216 -
accuracy: 0.7753 - val_loss: 1.2789 - val_accuracy: 0.6699
Epoch 91/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8205 -
accuracy: 0.7756 - val_loss: 1.2848 - val_accuracy: 0.6706
Epoch 92/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8198 -
accuracy: 0.7758 - val_loss: 1.2956 - val_accuracy: 0.6667
Epoch 93/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8161 -
accuracy: 0.7761 - val_loss: 1.2829 - val_accuracy: 0.6711
Epoch 94/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8154 -
accuracy: 0.7768 - val_loss: 1.2769 - val_accuracy: 0.6687
Epoch 95/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8141 -
accuracy: 0.7774 - val_loss: 1.2809 - val_accuracy: 0.6727
Epoch 96/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8139 -
accuracy: 0.7777 - val_loss: 1.2863 - val_accuracy: 0.6687
Epoch 97/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8120 -
accuracy: 0.7783 - val_loss: 1.2576 - val_accuracy: 0.6758
Epoch 98/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8101 -
accuracy: 0.7786 - val_loss: 1.3057 - val_accuracy: 0.6650
Epoch 99/100
3631/3631 [==============================] - 6s 2ms/step - loss: 0.8104 -
accuracy: 0.7781 - val_loss: 1.2682 - val_accuracy: 0.6737
```

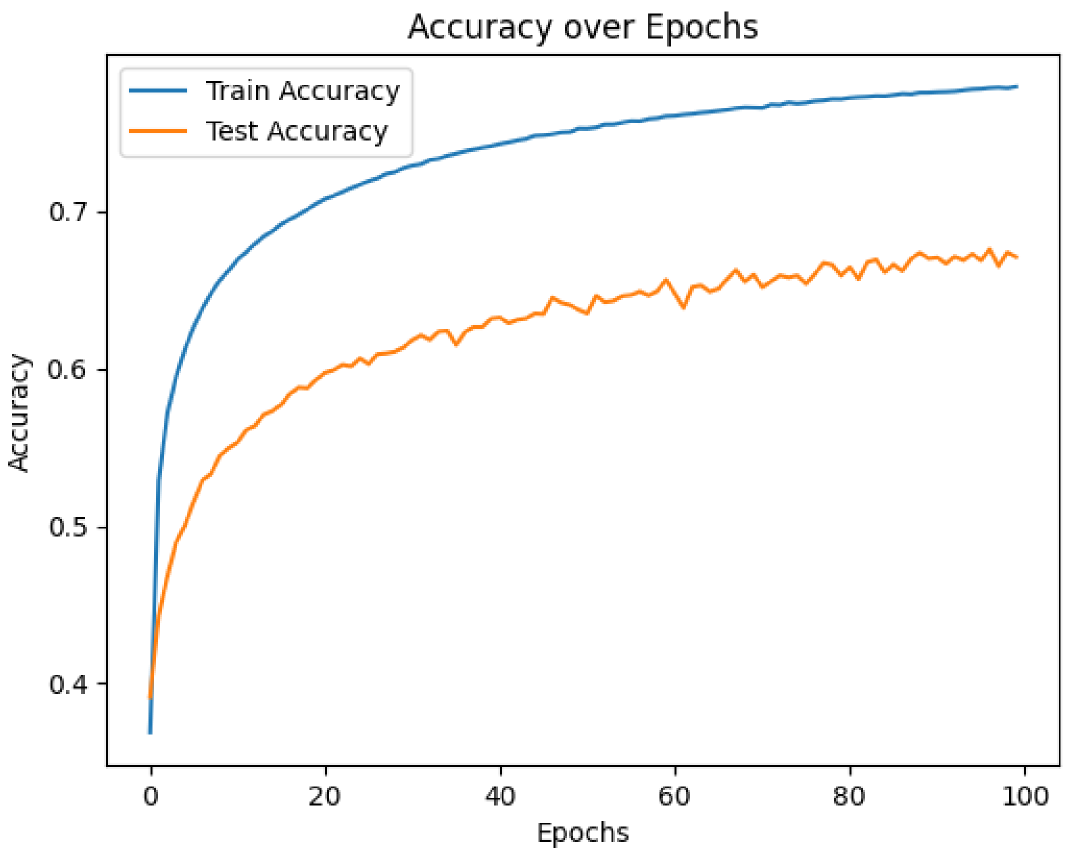
```
Epoch 100/100
3631/3631 [==============================] - 7s 2ms/step - loss: 0.8078 -
accuracy: 0.7791 - val_loss: 1.2688 - val_accuracy: 0.6709
```

1.4 4. Evaluation:

[20]:
```python
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images_rbf, test_labels_encoded)
print(f"Test accuracy: {test_accuracy}")

# Visualize performance metrics (optional, can add confusion matrix␣
 ↪visualization too)
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
1205/1205 [==============================] - 2s 1ms/step - loss: 1.2688 -
accuracy: 0.6709
Test accuracy: 0.6708952784538269
```



1.5 5. RBF Network Analysis

Strengths: - **Non-linear Mapping:** RBF networks effectively capture complex patterns in character recognition. - **Local Representation:** Each RBF unit focuses on specific regions of the data, making them useful for character-based classification. - **Clustering:** K-Means helps cluster similar patterns by finding RBF centers.

Limitations: - **Scalability:** With large datasets like Kuzushiji, RBF networks can become computationally expensive. - **Overfitting:** The significant gap between training (~80%) and test accuracy (~70%) indicates potential overfitting. - **Sensitivity:** Model performance is highly dependent on RBF centers, sigma, and number of units.

Effect of RBF Units: - **Too Few Units:** Results in underfitting, leading to poor accuracy (~2.5% initially). - **Too Many Units:** Causes overfitting, where the model performs well on training data but poorly generalizes to new data.

Accuracy: - Final model achieved **~80% train accuracy** and **~70% test accuracy**, indicating reasonable generalization but room for improvement.

Next Steps: - Add regularization, fine-tune `sigma`, or explore CNNs for enhanced performance.