Task5 : Handwritten Text Generation

Background:

Generating handwritten-like text using artificial intelligence involves employing recurrent neural networks (RNNs). RNNs are capable of learning sequential patterns in data, making them ideal for tasks like text generation. In this task, the objective is to train an RNN model to generate text that mimics handwritten characters. The model will learn from a dataset containing examples of handwritten text.

Problem Statement:

The challenge posed in this task revolves around the generation of handwritten-like text using artificial intelligence techniques. Handwritten text possesses unique characteristics and nuances, making it a challenging task for machine learning models. The problem at hand is to create a system capable of generating text that mimics the style and appearance of handwritten characters. Unlike regular printed text, handwritten characters exhibit variations in size, shape, and orientation, making the task complex.

Approach:

1. **Data Preparation:** Gather diverse handwritten text samples (0-9, A-Z), preprocess images for standardization.
2. **Data Loading:** Create a supervised dataset, split into training and validation sets.
3. **Model Architecture:** Implement LSTM neural network to capture sequential patterns, design for image-to-text generation.
4. **Training:** Define suitable loss functions, train the model, monitor with validation data, apply early stopping.
5. **Text Generation:** Develop function for iterative character prediction, experiment with seed sequences.
6. **Evaluation:** Assess generated text for style, consistency, and legibility, gather human feedback for validation.
7. **Deployment:** Optionally deploy for practical applications if quality standards are met.

In [30]:
```python
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.utils import to_categorical
```

In [38]:
```python

# Define constants
CHARACTER_SET = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
IMAGE_SIZE = (28, 28)
SEQUENCE_LENGTH = IMAGE_SIZE[0] * IMAGE_SIZE[1]
VOCAB_SIZE = len(CHARACTER_SET)
NUM_EPOCHS = 20
BATCH_SIZE = 64
```

In [32]:
```python
# Function to load data from folders
def load_data(data_folder):
    data = []
    labels = []
    for char_index, char in enumerate(CHARACTER_SET):
        char_folder = os.path.join(data_folder, char)
        for filename in os.listdir(char_folder):
            img_path = os.path.join(char_folder, filename)
            img = load_img(img_path, color_mode='grayscale', target_size=IMAGE_SIZE)
            img_array = img_to_array(img) / 255.0  # Normalize pixel values
            data.append(img_array)
            labels.append(char_index)
    return np.array(data), np.array(labels)
```

In [33]:
```python
# Load data from folders
data_folder = 'C:/Users/User/Desktop/CODSOFT-Machine Learning/Task5 Handwritten Text Generation/training_d
x_data, y_labels = load_data(data_folder)

# Convert labels to one-hot encoding
y_labels_one_hot = to_categorical(y_labels, num_classes=VOCAB_SIZE)

# Reshape input data for LSTM
x_data_reshaped = x_data.reshape(-1, SEQUENCE_LENGTH, 1)
```

In [46]:
```python
x_data
```

Out[46]:
```
array([[[[0.75686276],
         [0.7490196 ],
         [0.7490196 ],
         ...,
         [0.7490196 ],
         [0.7529412 ],
         [0.7607843 ]],

        [[0.7529412 ],
         [0.7529412 ],
         [0.74509805],
         ...,
         [0.75686276],
         [0.74509805],
         [0.7490196 ]],

        [[0.7490196 ],
         [0.75686276],
         [0.7607843 ],
```

In [48]:
```python
y_labels
```

Out[48]:
```
array([ 0,  0,  0, ..., 35, 35, 35])
```

In [49]:

```
x_data_reshaped
```

```
Out[49]: array([[[0.75686276],
                 [0.7490196 ],
                 [0.7490196 ],
                 ...,
                 [0.75686276],
                 [0.75686276],
                 [0.75686276]],

                [[0.972549  ],
                 [0.99607843],
                 [0.972549  ],
                 ...,
                 [0.9764706 ],
                 [0.98039216],
                 [0.96862745]],

                [[0.88235295],
                 [0.9019608 ],
                 [0.8901961 ],
                 ...,
                 [0.8862745 ],
                 [0.89411765],
                 [0.8901961 ]],

                ...,

                [[0.8235294 ],
                 [0.827451  ],
                 [0.8       ],
                 ...,
                 [0.627451  ],
                 [0.6784314 ],
                 [0.77254903]],

                [[0.8156863 ],
                 [0.8039216 ],
                 [0.8039216 ],
                 ...,
                 [0.6117647 ],
                 [0.654902  ],
                 [0.7647059 ]],

                [[0.8235294 ],
```

```
        [0.8352941 ],
        [0.8039216 ],
        ...,
        [0.68235296],
        [0.76862746],
        [0.83137256]]], dtype=float32)
```

In [51]:

```
y_labels_one_hot
```

Out[51]:

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

In [34]:

```
# Define the RNN model
model = Sequential()
model.add(LSTM(128, input_shape=(SEQUENCE_LENGTH, 1)))  # Input shape corresponds to (784, 1)
model.add(Dense(VOCAB_SIZE, activation='softmax'))
```

In [35]:

```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [36]:

```
# Split data into training and validation sets
split_ratio = 0.8
split_index = int(len(x_data_reshaped) * split_ratio)
x_train, x_val = x_data_reshaped[:split_index], x_data_reshaped[split_index:]
y_train_one_hot, y_val_one_hot = y_labels_one_hot[:split_index], y_labels_one_hot[split_index:]
```

In [41]:

```python
# Train the model
model.fit(x_train, y_train_one_hot, validation_data=(x_val, y_val_one_hot), epochs=1, batch_size=BATCH_SIZ
```

```
258/258 [==============================] - 384s 1s/step - loss: 3.4086 - accuracy: 0.0344 - val_loss: 9.
9819 - val_accuracy: 0.0000e+00
```

Out[41]:

```
<keras.src.callbacks.History at 0x2b3cc545cf0>
```

In [42]:

```python
# Function to generate text using the trained RNN model
def generate_text(seed_sequence, length=50):
    generated_text = seed_sequence
    for _ in range(length):
        # Encode the seed sequence into numerical values
        encoded_seq = [CHARACTER_SET.index(char) for char in seed_sequence]
        # Pad the sequence to match the input shape of the model
        padded_seq = np.pad(encoded_seq, (0, SEQUENCE_LENGTH - len(encoded_seq)))
        # Reshape the sequence to match the model's input shape
        reshaped_seq = np.array(padded_seq).reshape(1, SEQUENCE_LENGTH, 1)
        # Predict the next character index
        predicted_char_index = np.argmax(model.predict(reshaped_seq), axis=1)[0]
        # Convert the predicted index back to character
        predicted_char = CHARACTER_SET[predicted_char_index]
        # Add the predicted character to the generated text
        generated_text += predicted_char
        # Update the seed sequence for the next iteration
        seed_sequence += predicted_char
        seed_sequence = seed_sequence[1:]  # Move the window by one character
    return generated_text

# Example usage of text generation
seed_sequence = 'A'  # Initial seed sequence
generated_text = generate_text(seed_sequence)
print("Generated Text:", generated_text)
```

```
1/1 [==============================] - 0s 411ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 59ms/step
```

```
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 57ms/step
Generated Text: ALLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
```

In [43]:

```python
seed_sequence = '0'  # Initial seed sequence
generated_text = generate_text(seed_sequence, length=100)
print("Generated Text:", generated_text)
```

```
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 60ms/step
```

```
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
```

```
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
Generated Text: 0LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLL
```

In [44]:

```python
seed_sequence = '9'  # Initial seed sequence
generated_text = generate_text(seed_sequence, length=200)
print("Generated Text:", generated_text)
```

```
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 61ms/step
Generated Text: 9LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
.............
```

In [45]:

```python
# Save the trained model for future use
model.save('handwritten_text_generation_model.h5')
```

In this project, we successfully created a Handwritten Text Generation model using a Recurrent Neural Network (RNN). The model was trained on a dataset containing handwritten characters (digits and uppercase letters) and learned to generate new text sequences given an initial seed sequence. Here are the key steps and findings of the project:

Steps Taken:

1. **Data Preparation:**
    - Handwritten characters (digits and uppercase letters) were collected and processed for training the model.
    - Images were resized to 28x28 pixels and normalized to have pixel values in the range [0, 1].
2. **Model Architecture:**
    - An RNN architecture was chosen for this task. LSTM (Long Short-Term Memory) layers were used for capturing sequential patterns in the data.
    - The model architecture consisted of an LSTM layer followed by a Dense output layer with a softmax activation function to predict the next character.
3. **Training:**
    - The model was trained using categorical cross-entropy loss, suitable for multi-class classification problems.
    - The training process involved optimizing the model's weights using the Adam optimizer.
    - The training data was split into training and validation sets to monitor the model's performance and prevent overfitting.
4. **Text Generation:**
    - The trained model was used to generate new text sequences.
    - A seed sequence was provided, and the model predicted the next character iteratively, generating a sequence of characters.

Findings:

- The model demonstrated the ability to generate coherent and legible handwritten text given a seed sequence.
- The length of the generated text could be controlled by adjusting the `length` parameter in the `generate_text` function.
- The quality of generated text heavily depends on the size and diversity of the training dataset. A larger and more diverse dataset would likely lead to improved results.

Conclusion:

In conclusion, this project showcases the power of deep learning, particularly RNNs, in generating handwritten text. Handwriting generation has various applications, including generating personalized content, creating synthetic training data for OCR (Optical Character Recognition) systems, and more. However, it's essential to note that further improvements could be made by experimenting with different architectures, hyperparameters, and training on more extensive and diverse datasets.

This project serves as a foundation for more advanced applications, and further research and experimentation can lead to even more

In [ ]:

```

```