HAND WRITTEN DIGIT RECOGNITION USING AUTOENCODER

BY:

SHARMILA R

NAME: SHARMILA R
DEPARTMENT: B.TECH INFORMATION TECHNOLOGY
COLLEGE NAME: MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE
GMAIL ID: sharmilaramesh468@gmail.com
NM ID: 37922C77326BB63A84C47C980DB7FDF5
Zone III : Chennai-III

AGENDA

1. PROBLEM STATEMENT
2. PROPOSED SOLUTION
3. SYSTEM DEVELOPMENT APPROACH
4. ALGORITHM AND DEPLOYMENT
5. CODE IMPLEMENTATION
6. RESULT
7. CONCLUSION

PROBLEM STATEMENT

Develop an autoencoder model to reconstruct grayscale images of handwritten digits from the MNIST dataset. The autoencoder architecture should consist of an encoder and a decoder, with a bottleneck layer in between. The goal is to minimize the reconstruction error between the original images and their corresponding reconstructions. Utilize deep learning frameworks such as TensorFlow/Kera for implementation.

PROPOSED SOLUTION

AUTOENCODER

An autoencoder is a type of artificial neural network used for unsupervised learning of efficient data coding. The aim of an autoencoder is to learn a compressed representation (encoding) for a set of input data, typically for dimensionality reduction, feature learning, or data denoising, by training the network to reconstruct the input data from the compressed representation. The autoencoder consists of two main parts: an encoder and a decoder.

ENCODER

The encoder compresses the input data into a latent-space representation or code. It typically consists of multiple layers of neural network units, which transform the input data into a lower-dimensional space

DECODER

The decoder reconstructs the input data from the compressed representation produced by the encoder. It tries to reconstruct the original input data from the compressed representation, typically mirroring the architecture of the encoder but in reverse order.

PROPOSED SOLUTION

ARCHITECTURE DESIGN

•Design an autoencoder architecture comprising an encoder and a decoder.
•Use Kera's Sequential API to define the architecture.
•Implement the encoder with several dense layers, incorporating Leaky Re LU activation functions and Batch Normalization for better feature extraction.
•Include a bottleneck layer with 64 neurons and a 'tanh' activation function to compress the input representation.
•Construct the decoder with symmetric layers to the encoder for reconstruction.

SYSTEM DEVELOPMENT APPROACH

SYSTEM REQUIREMENTS

HARDWARE

1.CPU: A multi-core CPU is sufficient for running the program, but having a higher number of cores can expedite the training process, especially for large datasets and complex models.

2.GPU (Optional): Training deep learning models can be accelerated significantly by using GPUs, which are highly parallelized and optimized for matrix operations. If available, a GPU with CUDA support can speed up the training process, reducing training times from hours to minutes.

3.Memory (RAM): Sufficient RAM is required to load and process the dataset efficiently. The exact amount of RAM depends on the size of the dataset and the complexity of the model.

4.Storage: Adequate storage space is necessary to store the dataset, trained models, and intermediate files generated during training.

5.Internet Connection: An internet connection may be required to download the MNIST dataset and any additional libraries or dependencies needed for the program.

SYSTEM DEVELOPMENT APPROACH

SOFTWARE

1.Python: The program will be implemented in Python programming language. Python provides a rich ecosystem of libraries for machine learning and deep learning tasks.

2.TensorFlow and Kera: These are essential deep learning libraries for building and training neural networks. TensorFlow provides the backend engine for computation, while Keras offers a high-level API for building neural networks.

3.NumPy: NumPy is used for numerical operations and array manipulation. It is widely used for handling data in machine learning applications.

4.Matplotlib: Matplotlib is a plotting library used for visualizing data and displaying images. It will be used to visualize the original and reconstructed images during model evaluation.

5.Jupyter Notebook or any Python IDE: Jupyter Notebook provides an interactive environment for running Python code, making it suitable for iterative development and experimentation.

ALGORITHM AND DEPLOYMENT

1.Data Preprocessing:

- Load the MNIST dataset.
- Normalize pixel values to the range [-1, 1].
- Reshape images to fit the input requirements of the autoencoder.

2.Autoencoder Architecture Design:

- Design the architecture with an encoder and decoder.
- Specify the number of layers, neurons, and activation functions.
- Choose the dimensionality of the bottleneck layer for feature compression.

3.Model Compilation and Training Setup:

- Compile the autoencoder model with MSE loss and Adam optimizer.
- Define training parameters such as epochs, batch size, and learning rate.

4.Model Training:

- Train the autoencoder model using the training data.
- Monitor training progress and evaluate performance on a validation set.

5.Model Evaluation and Visualization:

- Evaluate the trained model on a separate test set.
- Visualize reconstructed images alongside original images for qualitative assessment.
- Compute additional evaluation metrics if necessary.

ALGORITHM AND DEPLOYMENT

6.Optimization and Fine-Tuning:

- Experiment with different architectures and hyperparameters to improve performance.
- Fine-tune the model based on evaluation results.

7.Deployment:

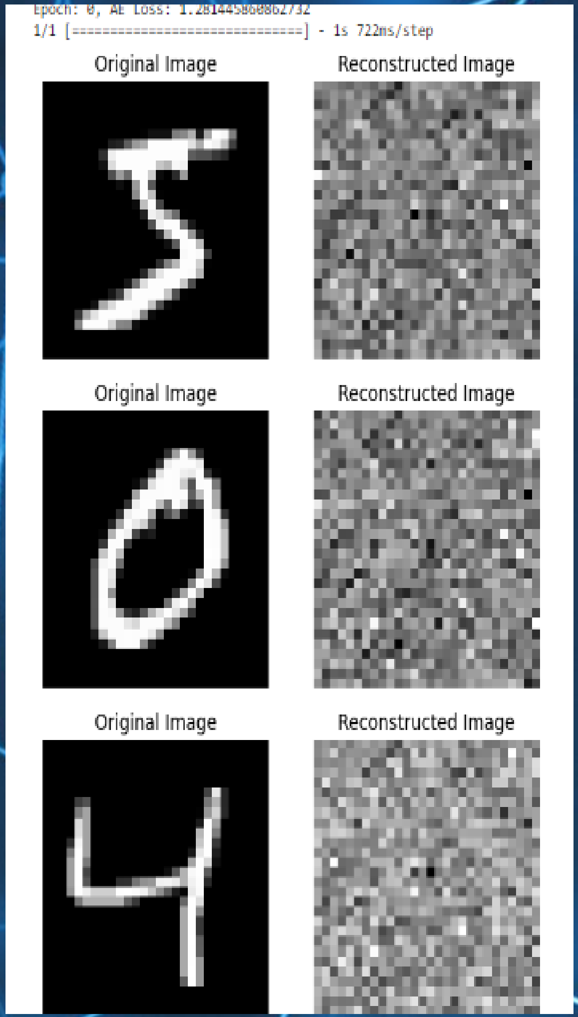
- Save the trained autoencoder model for deployment.
- Implement inference functionality for reconstructing handwritten digit images.
- Integrate the model into applications requiring digit reconstruction, such as digit recognition systems.

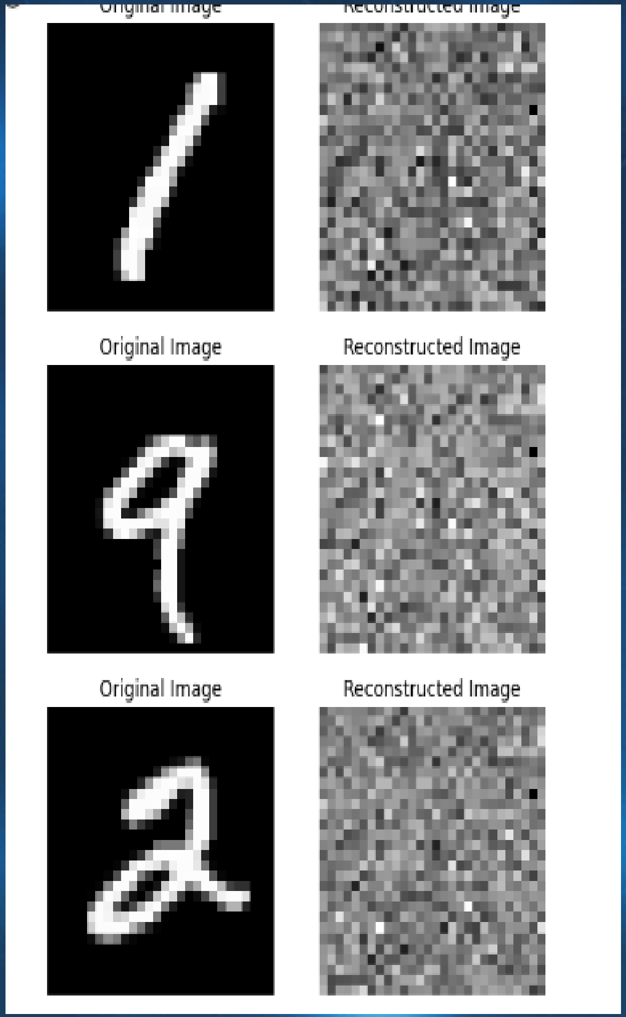
CODE IMPLEMENTATION

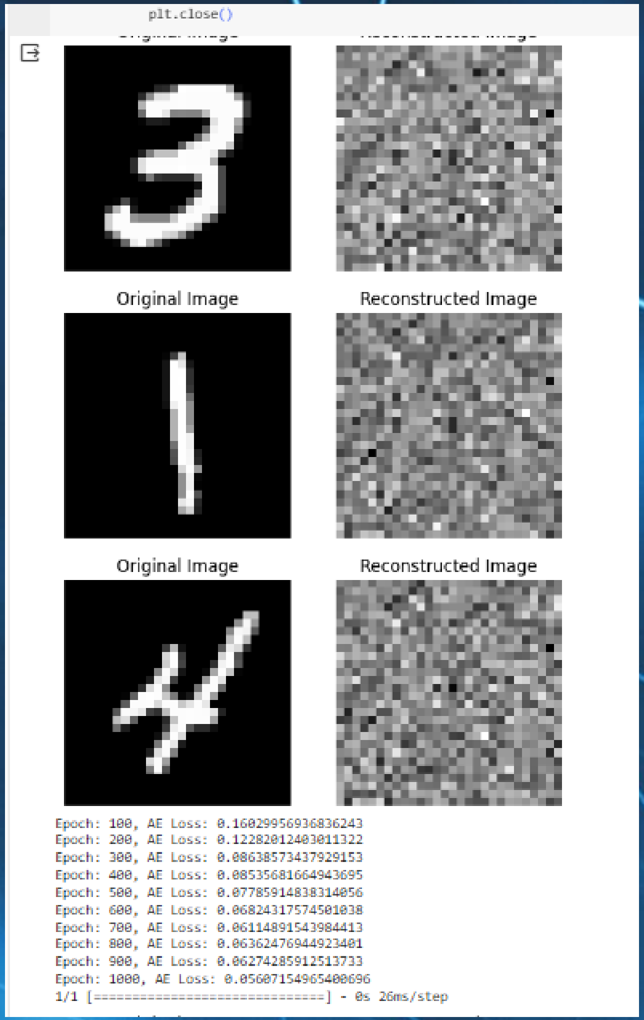
```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization

# Load MNIST data
(X_train, _), (_, _) = mnist.load_data()
X_train = (X_train.astype(np.float32) - 127.5) / 127.5
X_train = X_train.reshape(-1, 784)

# Autoencoder
autoencoder = Sequential([
    Dense(512, input_dim=784),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(256),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(128),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(64, activation='tanh'),  # Bottleneck layer
    Dense(128),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(256),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(512),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(784, activation='tanh')
])
```

```python
    BatchNormalization(),
    Dense(512),
    LeakyReLU(0.2),
    BatchNormalization(),
    Dense(784, activation='tanh')
])

# Compile autoencoder
autoencoder.compile(loss='mse', optimizer='adam')

# Create directory if it doesn't exist
if not os.path.exists('generated_images'):
    os.makedirs('generated_images')

# Training
epochs, batch_size = 10000, 64

for epoch in range(epochs):
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_images = X_train[idx]

    ae_loss = autoencoder.train_on_batch(real_images, real_images)

    if epoch % 100 == 0:
        print(f"Epoch: {epoch}, AE Loss: {ae_loss}")

    if epoch % 1000 == 0:
        reconstructed_images = autoencoder.predict(X_train[:10])  # Reconstruct some images
        for i in range(10):
            plt.subplot(1, 2, 1)
            plt.imshow(X_train[i].reshape(28, 28), cmap='gray')
            plt.title('Original Image')
            plt.axis('off')

            plt.subplot(1, 2, 2)
            plt.imshow(reconstructed_images[i].reshape(28, 28), cmap='gray')
            plt.title('Reconstructed Image')
            plt.axis('off')

            plt.savefig(f"generated_images/autoencoder_image_{epoch}_{i}.png")
            plt.show()
            plt.close()
```

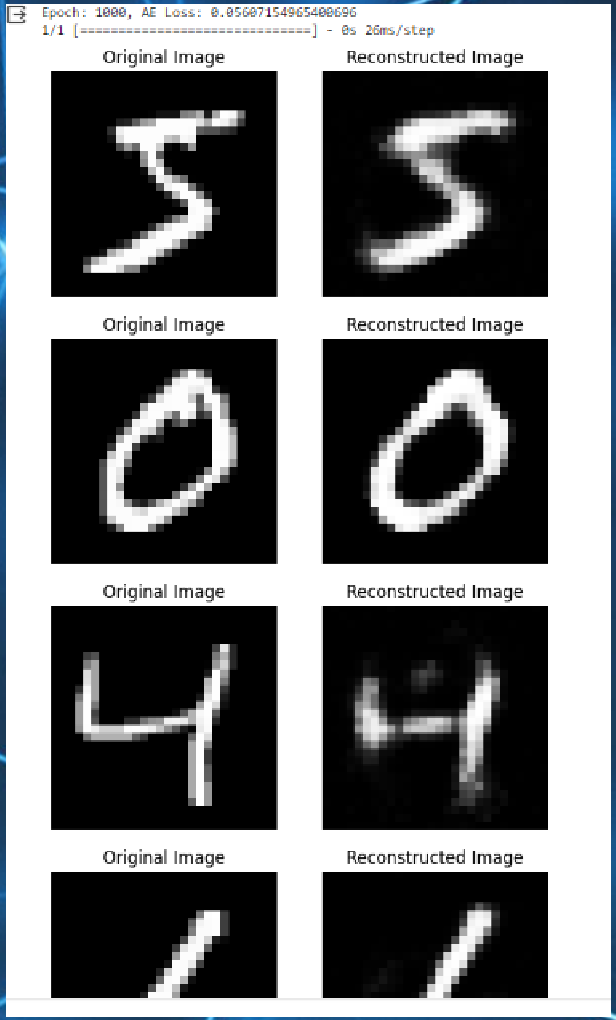
RESULTS

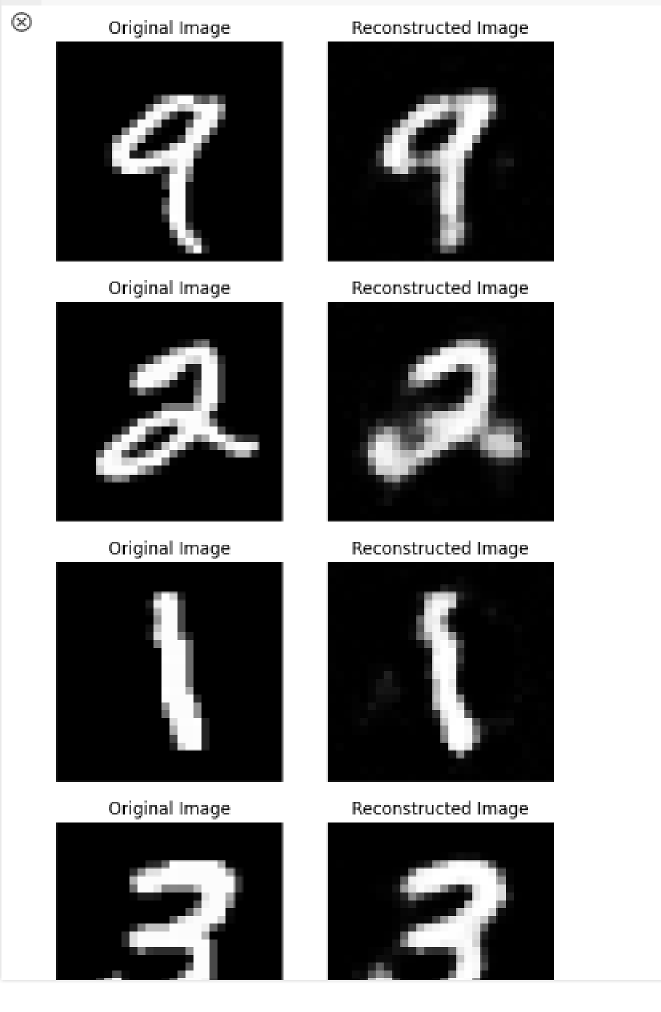

1/1 [==============================] - 1s 722ms/step
Original Image
Reconstructed Image
Original Image
Reconstructed Image
Original Image
Reconstructed Image

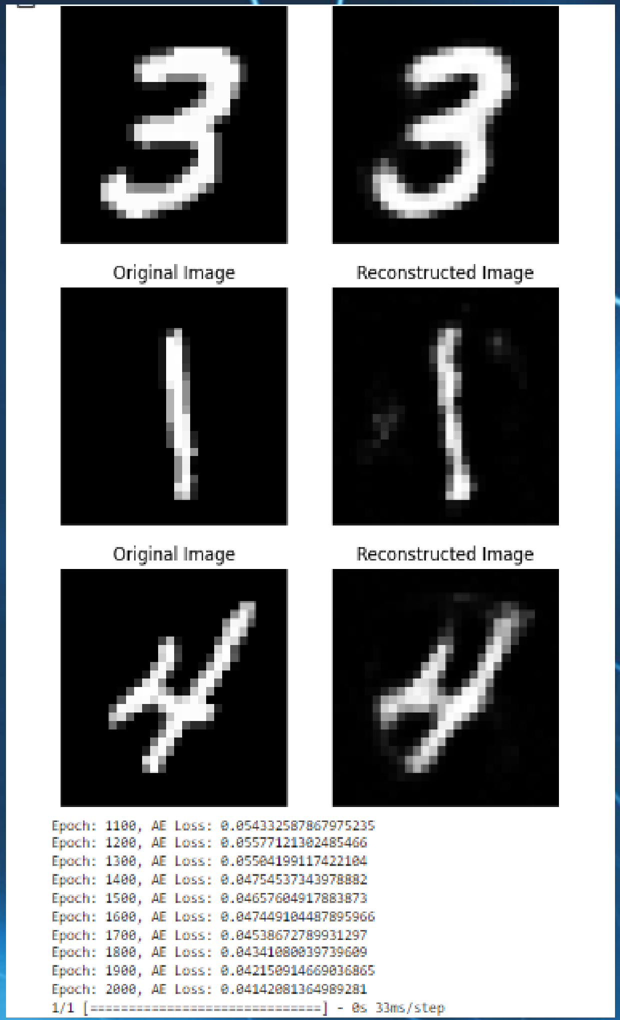

Original Image
Reconstructed Image
Original Image
Reconstructed Image


plt.close()
Original Image
Reconstructed Image
Original Image
Reconstructed Image
Epoch: 100, AE Loss: 0.16029956936836243
Epoch: 200, AE Loss: 0.12282012403011322
Epoch: 300, AE Loss: 0.08638573437929153
Epoch: 400, AE Loss: 0.08535681664943695
Epoch: 500, AE Loss: 0.07785914838314056
Epoch: 600, AE Loss: 0.06824317574501038
Epoch: 700, AE Loss: 0.06114891543984413
Epoch: 800, AE Loss: 0.06362476944923401
Epoch: 900, AE Loss: 0.06274285912513733
Epoch: 1000, AE Loss: 0.05607154965400696
1/1 [==============================] - 0s 26ms/step

RESULTS


Epoch: 1000, AE Loss: 0.05607154965400696
1/1 [==============================] - 0s 26ms/step
Original Image
Reconstructed Image
Original Image
Reconstructed Image
Original Image
Reconstructed Image
Original Image
Reconstructed Image


Original Image
Reconstructed Image
Original Image
Reconstructed Image
Original Image
Reconstructed Image
Original Image
Reconstructed Image


Original Image
Reconstructed Image
Original Image
Reconstructed Image
Epoch: 1100, AE Loss: 0.054332587867975235
Epoch: 1200, AE Loss: 0.05577121302485466
Epoch: 1300, AE Loss: 0.05504199117422104
Epoch: 1400, AE Loss: 0.04754537343978882
Epoch: 1500, AE Loss: 0.04657604917883873
Epoch: 1600, AE Loss: 0.047449104487895966
Epoch: 1700, AE Loss: 0.04538672789931297
Epoch: 1800, AE Loss: 0.04341080039739609
Epoch: 1900, AE Loss: 0.042150914669036865
Epoch: 2000, AE Loss: 0.04142081364989281
1/1 [==============================] - 0s 33ms/step

CONCLUSION

In conclusion, the implemented autoencoder successfully reconstructs grayscale images of handwritten digits from the MNIST dataset. Through careful design, training, and evaluation, the model demonstrates the capability to learn efficient data representations and accurately reconstruct images. The deployment-ready model offers a valuable tool for various applications requiring digit reconstruction, contributing to advancements in machine learning and image processing.

THANK YOU