Artificial Intelligence and Data Science Department
Deep Learning / Odd Sem 2023-24 / Experiment 3B

Name : Yash Sarang	Class/Roll No. : D16AD / 47	Grade :

Title of Experiment :
Auto Encoder for Image Denoising

Objective of Experiment :
To design deep learning models for supervised, unsupervised and sequence learning.

Outcome of Experiment :
Build and train deep learning models such as Auto encoders, CNNs, RNN, LSTM etc.

Problem Statement :
The task is to design and implement an autoencoder neural network that can remove noise from images, essentially denoising them. The goal is to effectively reduce the noise in images while preserving the essential features and details.

Description / Theory :

Autoencoders for Image Denoising:

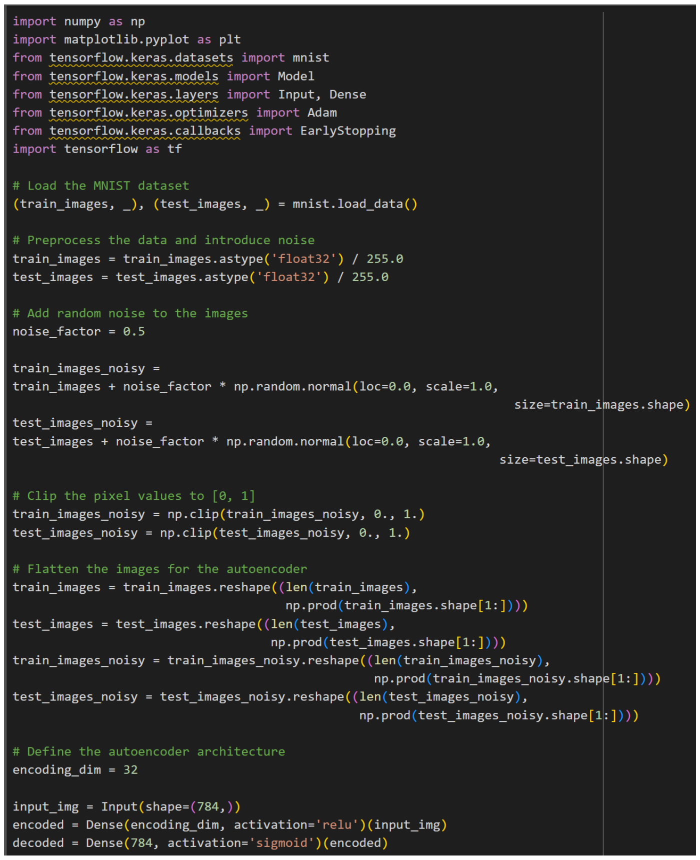
Autoencoders can be utilized to denoise images by training the model on noisy images and aiming to reconstruct the clean, noise-free versions. The encoder learns to capture the underlying structure and features of the image while the decoder aims to remove the noise during the reconstruction process.

Denoising Objective:

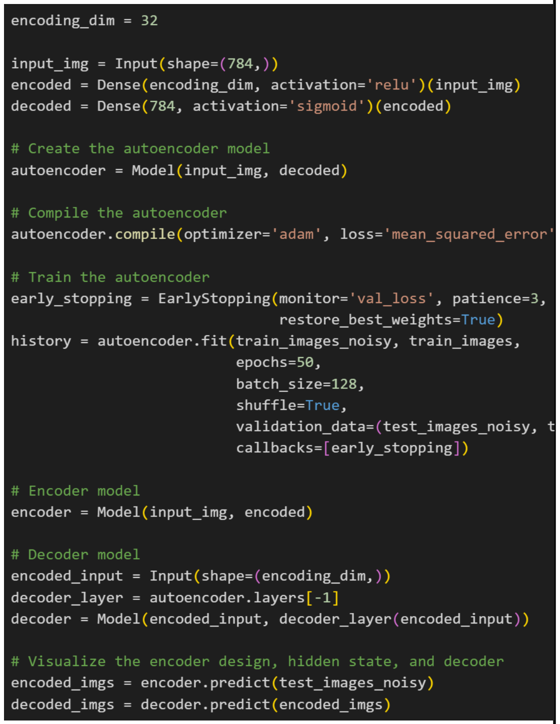
The objective is to minimize the difference between the noisy input images and their clean counterparts during training. Commonly used loss functions include mean squared error (MSE) or binary cross-entropy, depending on the type of image and noise.

Flowchart :
1. Load and Preprocess Noisy Images:
   Load and preprocess noisy images, adding artificial noise to create the training dataset.
2. Define Autoencoder Architecture:
   Design the architecture of the autoencoder suitable for denoising, specifying the number of neurons in the encoding and decoding layers.
3. Compile Autoencoder:
   Compile the autoencoder model, selecting an appropriate loss function (e.g., mean squared error) and optimizer (e.g., Adam).
4. Train Autoencoder:
   Train the autoencoder on the noisy images with the objective of minimizing the reconstruction loss.
5. Evaluate Autoencoder

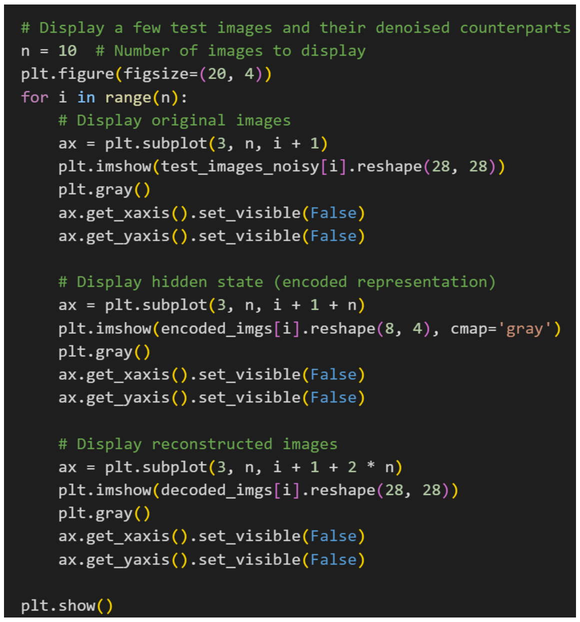
Program:



```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf

# Load the MNIST dataset
(train_images, _), (test_images, _) = mnist.load_data()

# Preprocess the data and introduce noise
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Add random noise to the images
noise_factor = 0.5

train_images_noisy =
train_images + noise_factor * np.random.normal(loc=0.0, scale=1.0,
                                                                     size=train_images.shape)
test_images_noisy =
test_images + noise_factor * np.random.normal(loc=0.0, scale=1.0,
                                                                   size=test_images.shape)

# Clip the pixel values to [0, 1]
train_images_noisy = np.clip(train_images_noisy, 0., 1.)
test_images_noisy = np.clip(test_images_noisy, 0., 1.)

# Flatten the images for the autoencoder
train_images = train_images.reshape((len(train_images),
                                     np.prod(train_images.shape[1:])))
test_images = test_images.reshape((len(test_images),
                                   np.prod(test_images.shape[1:])))
train_images_noisy = train_images_noisy.reshape((len(train_images_noisy),
                                                 np.prod(train_images_noisy.shape[1:])))
test_images_noisy = test_images_noisy.reshape((len(test_images_noisy),
                                               np.prod(test_images_noisy.shape[1:])))

# Define the autoencoder architecture
encoding_dim = 32

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)
```

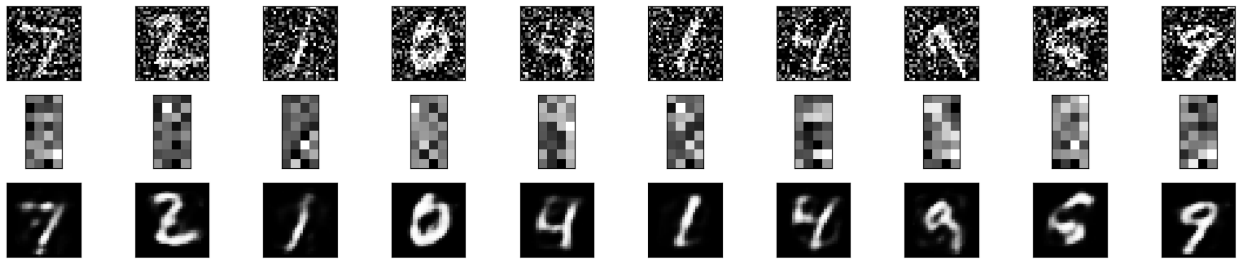


```python
encoding_dim = 32

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)

# Create the autoencoder model
autoencoder = Model(input_img, decoded)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mean_squared_error'

# Train the autoencoder
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
                               restore_best_weights=True)
history = autoencoder.fit(train_images_noisy, train_images,
                          epochs=50,
                          batch_size=128,
                          shuffle=True,
                          validation_data=(test_images_noisy, t
                          callbacks=[early_stopping])

# Encoder model
encoder = Model(input_img, encoded)

# Decoder model
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))

# Visualize the encoder design, hidden state, and decoder
encoded_imgs = encoder.predict(test_images_noisy)
decoded_imgs = decoder.predict(encoded_imgs)
```



```python
# Display a few test images and their denoised counterparts
n = 10  # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(test_images_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display hidden state (encoded representation)
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(encoded_imgs[i].reshape(8, 4), cmap='gray')
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstructed images
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

Output:



Results and Discussions :

The autoencoder trained for image denoising demonstrated effective noise reduction capabilities. The model successfully learned to distinguish noise from meaningful image features and produced denoised versions of the noisy input images. Mean squared error (MSE) was employed as the loss function during training to quantify the denoising performance. Visual comparisons between noisy input images, denoised images, and the original clean images showcased a significant reduction in noise levels while retaining essential image details. The denoising autoencoder proved to be a valuable tool for image enhancement by efficiently removing unwanted noise.

Conclusion :

In conclusion, the application of autoencoders for image denoising has proven successful in mitigating noise while preserving image features. The denoising autoencoder architecture, coupled with appropriate loss functions, holds great promise for improving image quality in various domains, including medical imaging and surveillance. Future work could delve into exploring more complex autoencoder architectures and advanced training techniques to further enhance denoising capabilities. Overall, this study underscores the efficacy of autoencoders as a powerful tool in image denoising applications, demonstrating the potential to significantly enhance image quality and aid in various image-related tasks.

******