Task 15: Implementing a Denoising Autoencoder using TensorFlow/Keras

---

Objective

The goal of this assignment is to build and train a **Convolutional Autoencoder** (CAE) to remove noise from images. You will:

1. Load and preprocess the MNIST dataset.
2. Add noise to the dataset to simulate a noisy environment.
3. Design and implement a convolutional autoencoder.
4. Train the model to denoise the images and evaluate its performance.

Instructions

1. Load the MNIST Dataset

1. Import the MNIST dataset using TensorFlow/Keras.
2. Normalize the pixel values to the range `[0, 1]`.

```
from tensorflow.keras.datasets import mnist

import numpy as np

# Load the MNIST dataset

(x_train, _), (x_test, _) = mnist.load_data()



# Normalize pixel values to [0, 1]

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0



# Add a channel dimension to make it compatible with CNN layers

x_train = np.expand_dims(x_train, axis=-1)

x_test = np.expand_dims(x_test, axis=-1)
```

```
print("Training data shape:", x_train.shape)

print("Test data shape:", x_test.shape)
```

2. Add Noise to the Dataset

1. Introduce Gaussian noise to the MNIST dataset.
2. Ensure the noisy pixel values remain in the range `[0, 1]` using `np.clip()`.

```
# Add Gaussian noise

noise_factor = 0.5

x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)

x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)


# Clip pixel values to be in [0, 1]

x_train_noisy = np.clip(x_train_noisy, 0., 1.)

x_test_noisy = np.clip(x_test_noisy, 0., 1.)


print("Noisy training data shape:", x_train_noisy.shape)
```

3. Visualize the Dataset

Plot a few examples of the original and noisy images.

4. Design the Convolutional Autoencoder

The autoencoder should have:

- **Encoder**:
  - A series of convolutional layers to extract features and reduce spatial dimensions.
  - Pooling layers to downsample the feature maps.
- **Decoder**:
  - Transposed convolution (or upsampling) layers to restore spatial dimensions.
  - A final convolutional layer to reconstruct the cleaned image.

Architecture:

- Input: Shape `(28, 28, 1)` (grayscale MNIST images).
- Encoder:
  - Conv2D with 32 filters, kernel size `(3, 3)`, ReLU activation, and `same` padding.
  - MaxPooling2D with pool size `(2, 2)` to reduce dimensions.
  - Conv2D with 64 filters, kernel size `(3, 3)`, ReLU activation, and `same` padding.
  - MaxPooling2D with pool size `(2, 2)`.
- Decoder:
  - Conv2D with 64 filters, kernel size `(3, 3)`, ReLU activation, and `same` padding.
  - UpSampling2D to increase dimensions.
  - Conv2D with 32 filters, kernel size `(3, 3)`, ReLU activation, and `same` padding.
  - UpSampling2D.
  - Conv2D with 1 filter, kernel size `(3, 3)`, `sigmoid` activation, and `same` padding (output layer).

```
from tensorflow.keras import layers, models

# Define the Autoencoder

input_shape = (28, 28, 1)

autoencoder = models.Sequential([

  # Encoder

  ……

  # Decoder

  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),

  layers.UpSampling2D((2, 2)),

…….

])



# Compile the Autoencoder

autoencoder.compile(….)



# Summary of the model
```

```
autoencoder.summary()
```

5. Train the Autoencoder

1. Use the noisy images as inputs and the original images as targets.
2. Train the model for 10 epochs with a batch size of 128.

```
# Train the Autoencoder

autoencoder.fit(

  x_train_noisy, x_train,

  epochs=20,

  batch_size=128,

  validation_data=(x_test_noisy, x_test)

)
```

6. Evaluate and Visualize the Results

1. Use the trained autoencoder to clean the noisy test images.
2. Compare the original, noisy, and denoised images.

```
# Denoise the test images

denoised_images = autoencoder.predict(x_test_noisy)



# Visualize noisy and denoised images

n = 5  # Number of images to display

plt.figure(figsize=(10, 6))

for i in range(n):

  # Noisy images

  ax = plt.subplot(3, n, i + 1)

  plt.imshow(x_test_noisy[i].squeeze(), cmap="gray")
```

```
    plt.title("Noisy")
    plt.axis("off")

    # Denoised images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(denoised_images[i].squeeze(), cmap="gray")
    plt.title("Denoised")
    plt.axis("off")

    # Original images
    ax = plt.subplot(3, n, i + 1 + 2*n)
    plt.imshow(x_test[i].squeeze(), cmap="gray")
    plt.title("Original")
    plt.axis("off")
plt.show()
```