MNIST Numbers Denoising

Introduction

The goal of this project is to build a denoising autoencoder that can remove added noise from images of handwritten digits in the MNIST dataset. The autoencoder is trained on noisy versions of the images and learns to reconstruct the original, clean images.

Data Preprocessing

The MNIST dataset consists of grayscale images of size 28×28 pixels. The images are normalized to the range [0,1] to facilitate stable training. Artificial noise is introduced by randomly selecting a percentage of pixels and modifying them with random values from a specified range. This process ensures the model learns to generalize noise removal across different distortions.

Noise Addition Method

The function `add_noise_random_locations()` is responsible for adding noise to the images in the dataset. It works by randomly selecting a subset of pixels in each image and modifying their values using random noise sampled from a specified range. This process mimics real-world image corruption and enhances the model's ability to generalize.

Autoencoders

Autoencoders are a class of neural networks in deep learning that operate under unsupervised learning principles, meaning they don't require labeled data for training. Their primary function is to reconstruct input data by learning efficient representations, effectively capturing the essence of the data. They consist of encoders and decoders.
Encoder: This part compresses the input data into a latent space representation, often referred to as the bottleneck layer. Here, the input is reduced to a lower-dimensional vector, capturing the most critical features.
Decoder: This component reconstructs the original data from the compressed latent representation, aiming to produce an output as close as possible to the original input.

Model Architecture

The model follows an autoencoder structure comprising an **encoder** and a **decoder**:

Encoder

The encoder is responsible for compressing the input image into a compact latent representation while preserving essential features. It consists of:

1. **Convolutional Layer (3×3 kernel, 32 filters, ReLU activation, padding='same')**: Extracts local features such as edges and textures.
2. **MaxPooling Layer (2×2, padding='same')**: Reduces the spatial dimensions by half while retaining important information.
3. **Another Convolutional Layer (3×3 kernel, 32 filters, ReLU activation, padding='same')**: Extracts more complex patterns.
4. **Another MaxPooling Layer (2×2, padding='same')**: Further compresses the image, resulting in a reduced representation containing only the most significant features.

Decoder

The decoder reconstructs the original clean image from the compressed representation. It consists of:

1. **Convolutional Layer (3×3 kernel, 32 filters, ReLU activation, padding='same')**: Begins the reconstruction process by refining the feature map.
2. **UpSampling Layer (2×2)**: Expands the image size back to its previous dimensions.
3. **Another Convolutional Layer (3×3 kernel, 32 filters, ReLU activation, padding='same')**: Further refines the features.
4. **Another UpSampling Layer (2×2)**: Expands the image to its original 28×28 resolution.
5. **Final Convolutional Layer (3×3 kernel, 1 filter, Sigmoid activation, padding='same')**: Produces the final denoised image with pixel values between 0 and 1.

Training and Optimization

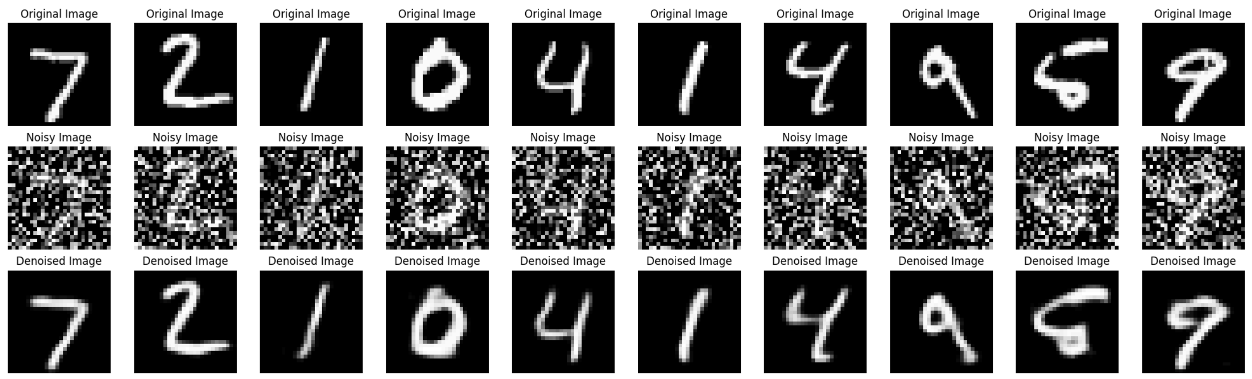
The autoencoder is trained using the **binary cross-entropy loss function**, which measures the pixel-wise difference between the output and the ground truth clean image. The **Adam optimizer** is used to update the network's weights efficiently. During training, the model learns to minimize reconstruction errors by adjusting convolutional filters to emphasize meaningful features while suppressing noise.

Rationale for Design Choices

1. **3×3 Kernels**: These allow the model to capture fine-grained details without excessive computational complexity.
2. **32 Filters**: A sufficient number of filters ensures that diverse image patterns are learned while avoiding overfitting.

3. **MaxPooling & UpSampling**: Pooling layers help reduce noise naturally while compressing the representation, and upsampling layers aid in reconstructing the original dimensions.
4. **Sigmoid Activation in the Output Layer**: Ensures pixel values remain within the valid range of grayscale images.

Results



C Sharp Console Application

A C# console application was developed to preprocess, add noise, and denoise images using an ONNX model. The application utilizes the `Microsoft.ML.OnnxRuntime` library to load and run the ONNX model. The core functionalities of the application include:

Preprocessing the Image:

- The image is loaded and resized to **28x28** pixels.
- Normalized to the range **[0,1]**.
- Random noise is added to simulate corruption.

Model Inference using ONNX

- The noisy image is flattened and reshaped into a **4D tensor** (1, 28, 28, 1) as required by the model.
- The ONNX model is loaded, and the denoised output is obtained.
- The output tensor is reshaped back into a **28x28** format.

Results

Input Image	Noisy image	Denoised Image
