Deep Learning for Image Denoising

Djahid ABDELMOUMENE

Image denoising is a crucial task in computer vision that aims to remove unwanted noise while preserving important image details. This is relevant in computer analysis tasks where noise can be introduced by sensor imperfections, transmission errors, or compression artifacts. In this exercice, you will:

- Study various types of image noise and their effects.
- Learn how to preprocess image datasets: MNIST, CIFAR-10 and COCO
- Implement and compare different deep learning architectures: FCNN, Autoencoder, and U-Net.
- Evaluate models using relevant metrics and visualize the results.

1 Background

1.1 Image Noise Types

Noise degrades the visual quality of images. We will be studying two types:

Gaussian Noise: Additive noise with a normal gaussian distribution added to the signal.

Salt-and-Pepper Noise: Speckle noise that randomly replaces some pixel values with extreme (black or white) intensities.

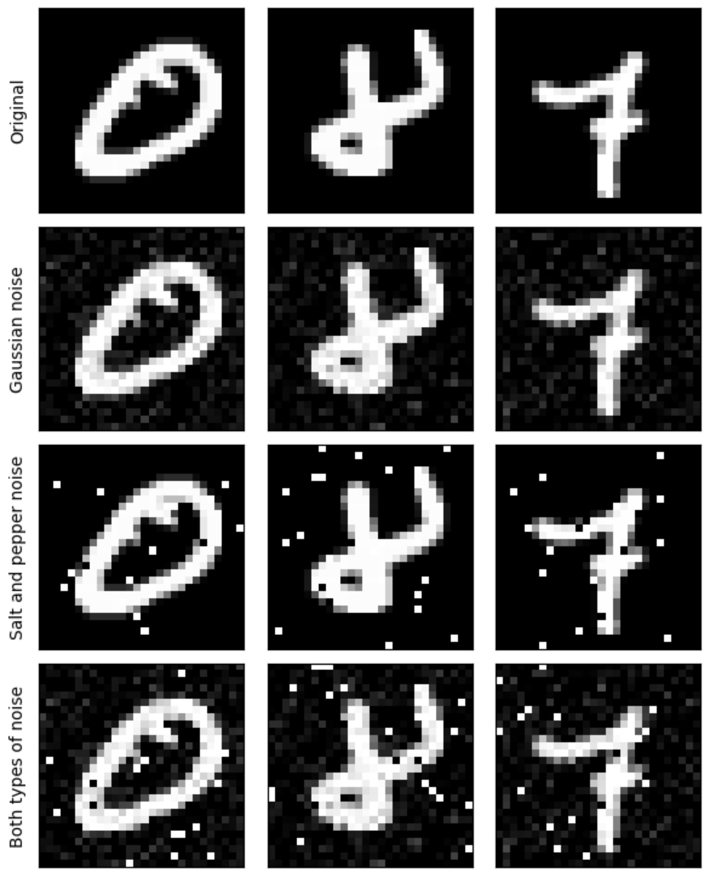

Figure 1: Different noise types applied to MNIST dataset

2 Datasets Description

We will be using three datasets to illustrate the effect of noise on different types of images.

2.1 MNIST

Description:

- **Content:** 60,000 grayscale images of handwritten digits (0–9) with a resolution of 28x28 pixels.

2.2 CIFAR-10

Description:

- **Content:** 60,000 color images across 10 classes (e.g., airplane, automobile, bird, etc.) with a resolution of 32x32 pixels.

2.3 COCO

Description:

- Dataset for image segmentation tasks. Contains 328k color images along with various segmentation and object detection information. We'll be using the images only.
- Can be downloaded here `https://cocodataset.org`

Figure 2: *
MNIST Dataset

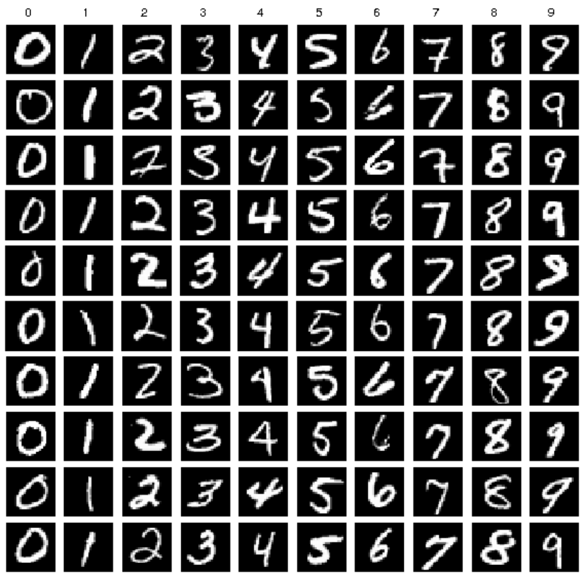

Figure 3: *
CIFAR Dataset

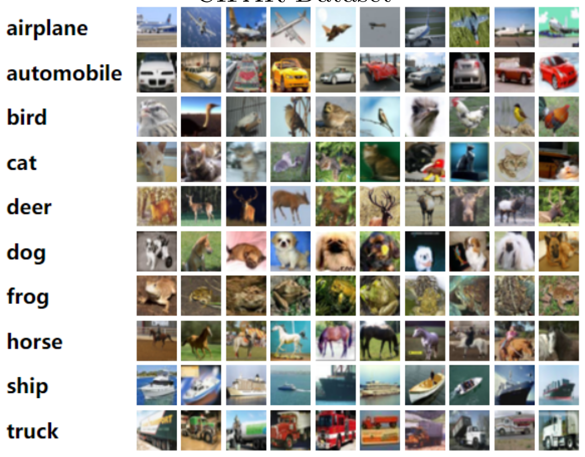

Figure 4: COCO Dataset

3 Deep Learning Models for Denoising

3.1 Fully Connected Neural Network (FCNN)

We will be using a simple FCNNs as a baseline for comparison to denoise the images.

- **Architecture:**
  - Input: Flattened image.
  - Hidden layers: A few dense layers with ReLU activations.
  - Output: Dense layer reshaped to the original image dimensions with a Sigmoid activation.

3.2 Autoencoder

Autoencoders learn to compress images into a latent representation and then reconstruct them. This compression leads to the network keeping only the most pertinent information for an image which when reconstructed will leave out the irrelevant noise.

- **Architecture:**

- **Encoder:** The input will be the flattened image followed by a series of Fully connected layers that progressively reduce the spatial dimensions.
- **Latent Space:** A bottleneck layer capturing the essential features.
- **Decoder:** A series of fully connected layers to reconstruct the flattened image.

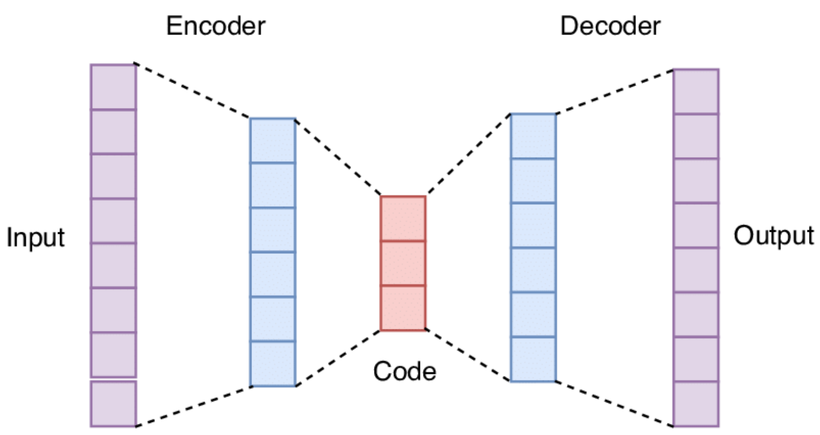

Figure 5: Auto encoder network

3.3 U-Net

U-Net architectures are powerful for image-to-image translation tasks due to their skip connections that preserve spatial information. It works in similar way to the autoencoder but with convolutional layers and skip connections

- **Architecture:**
  - **Contracting Path:** A sequence of convolutional layers with pooling to capture context.
  - **Expansive Path:** Upsampling layers with skip connections from the contracting path to combine high-resolution features.

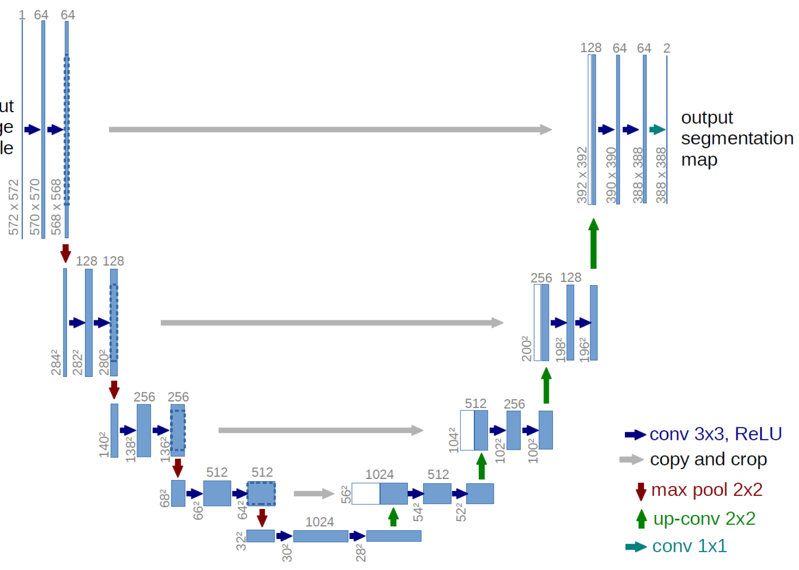

Figure 6: U-Net architecture illustrating the skip connections

4 Training Process

4.1 Loss Function and Optimizer

- **Loss Function:** Use Mean Squared Error (MSE) to measure the difference between the denoised image and the original:

  $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  You can also use the Structural Similarity Index Measure (SSIM) as a loss function, which evaluates the perceived quality of an image by comparing its luminance, contrast, and structure with a reference image. The SSIM formula is given by:

  $$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

  where: x and y are the reference and distorted images. μ_x, μ_y are the mean intensities of x and y. σ_x^2, σ_y^2 are the variances of x and y. σ_{xy} is the covariance between x and y. C_1 and C_2 are small constants to stabilize the division.

- **Optimizer:** The training optimizers that can be used are Adam or SGD.

4.2 Training Loop and Visualization

- **Training Setup:** Define a number of epochs (e.g., 10–50) and a batch size (e.g., 32 or 64).
- **Monitoring:** Track training and validation loss using either console logging or tensorboard.

5 Evaluation Metrics and Visualization Techniques

5.1 Evaluation Metrics

To assess the performance of the denoising models, consider:

- **Mean Squared Error (MSE):** Computes the average squared difference between the reconstructed image and the ground truth.
- **Peak Signal-to-Noise Ratio (PSNR):** Calculated as:

  $$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

  where MAX_I is the maximum possible pixel value.

- **Structural Similarity Index (SSIM):** Measures the perceived quality by comparing luminance, contrast, and structural similarity between images.

6 Exercices

For the practical exercices we'll be using PyTorch library.

1. **Data Preparation:**

- Download the COCO dataset. The MNIST and CIFAR datasets are available in PyTorch's datasets collection.
- Load the datasets using PyTorch's dataset tools.
- Resize and normalize the images when necessary.
- Create a utility functions to add the two types of noise to images.
- Create utility functions to flatten and de-flatten the images.
- Split the datasets into training and validation and test subsections.
- Visualize the image data.

2. **Model Implementation:** For each architecture (FCNN, Autoencoder, U-Net):

    - Build the networks architecture and implement the PyTorch Module and the forward method.

3. **Training:** Create a training function for each dataset with a parameter for the model name.

    - Define the loss functions (MSE, SSIM) and choose an optimizer (e.g., Adam, SGD).
    - Set training parameters for each model (epochs, batch size, learning rate).
    - Initialize each model according to the function parameter.
    - Create the training loop and calculate the loss for the validation subset at each epoch.

4. **Evaluation:**

    - Compute evaluation metrics (MSE, PSNR, SSIM) on the test sets.
    - Visualize the denoised images against their noisy and clean counterparts.
    - Plot loss curves and metric trends over epochs.
    - Compare the different models' performances on the various datasets.