Project 2 - Anomaly Detection with Autoencoder

5LSH0 - COMPUTER VISION AI AND 3D DATA ANALYSIS

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Date	Version
09.02.2025	2

Author
Jakub Frydrych

Contents

1 Implementation

1.1 Encoder and decoder parts

Initial implemented model, called “Autoencoder_FCN_simple” is a fully connected neural network (FCN) autoencoder designed for MNIST dataset consisting of 28x28 grayscale images. Model consists of two symmetrical components: encoder and decoder. The encoder compresses input image into lower dimensional space, while the decoder aims to reconstruct the original image from its compressed representation.

The encoder consists of three linear layers with decreasing neuron counts (784 -> 128 -> 64 -> 16), each followed by a ReLU activation function. Connecting encoder and decoder is a 16 neuron bottleneck layer using Tanh activation layer. The decoder mirrors the encoder's structure, expanding the latent representation back to the original image size (16 -> 64 -> 128 -> 784) using ReLU activations. Final output uses Sigmoid to ensure puxel values normalized between 0 and 1.

1.2 Mean Squared Error (MSE) loss function

Defined in “Training Setup” part of the code using PyTorch functionality:

criterion = nn.MSELoss() # Mean Squared Error loss function

1.3 Stochastic Gradient Descent (SGD)

Defined in “Training Setup” part of the code using PyTorch functionality:

optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE) # SGD optimizer

Learning rate is defined by according hyperparameter defined at the beginning of program.

1.4 The rest of the code (code structure, comments etc.)

Additional functions used to evaluate data conversion, model output size and report training progress have been implemented. Usage of clearly defined hyperparameters allows for easy experimentation. Appropriate comments explaining code have been added.

2 Questions

2.1 Take a subset (10%) of the MNIST test dataset and add excessive noise to corrupt those images. Visualize four samples of original and corrupted images. Note: The type of noise is optional. Be sure that the amount of noise is enough to find outliers (anomalies) in question 8 with the help of the threshold value.

Appropriate subset generation with gaussian noise has been implemented. Noise strength is controlled by hyperparameter “NOISE_VALUE”.

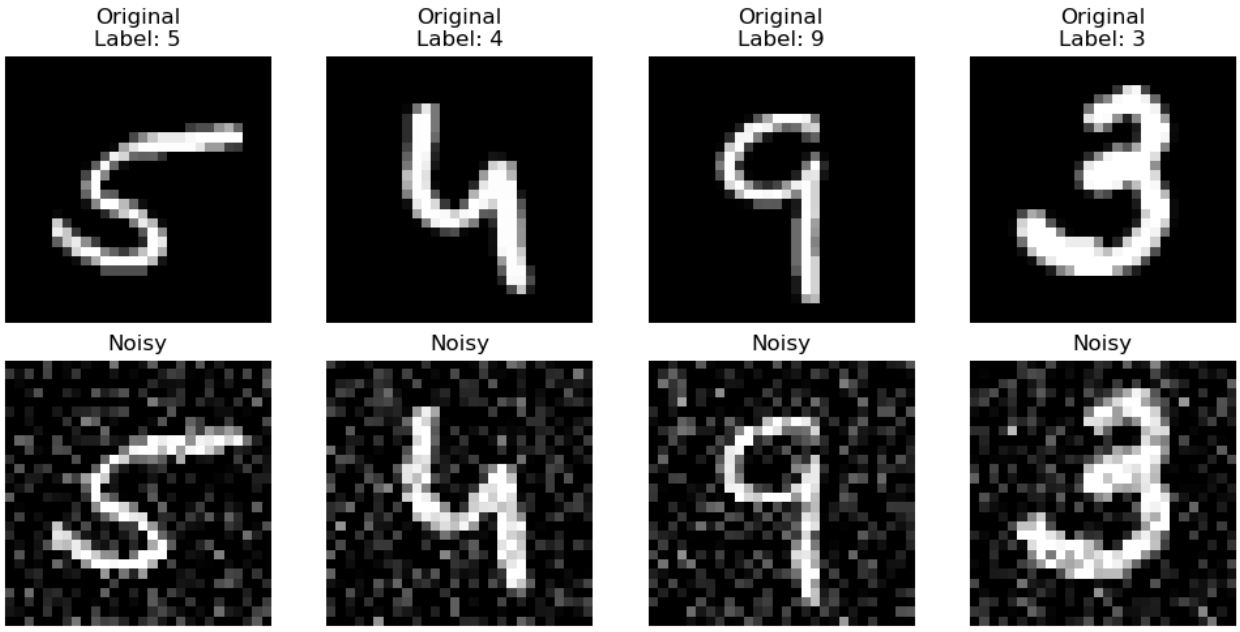

Figure 1 Generated noisy samples

2.2 Train your autoencoder architecture using the MNIST training dataset. Based on this trained autoencoder model, generate the output from corrupted and not corrupted input test images. Compare those output results with the input images by visualizing four different samples. Comment on the results. What did you expect, and what did you observe? What are the reasons for similarities and differences?

Proposed simple model seems to be insufficient for appropriate performance. Reconstructed image seems to be an averaged number based on all samples. No significant loss increase was observed when reconstructing from noisy images compared to originals.

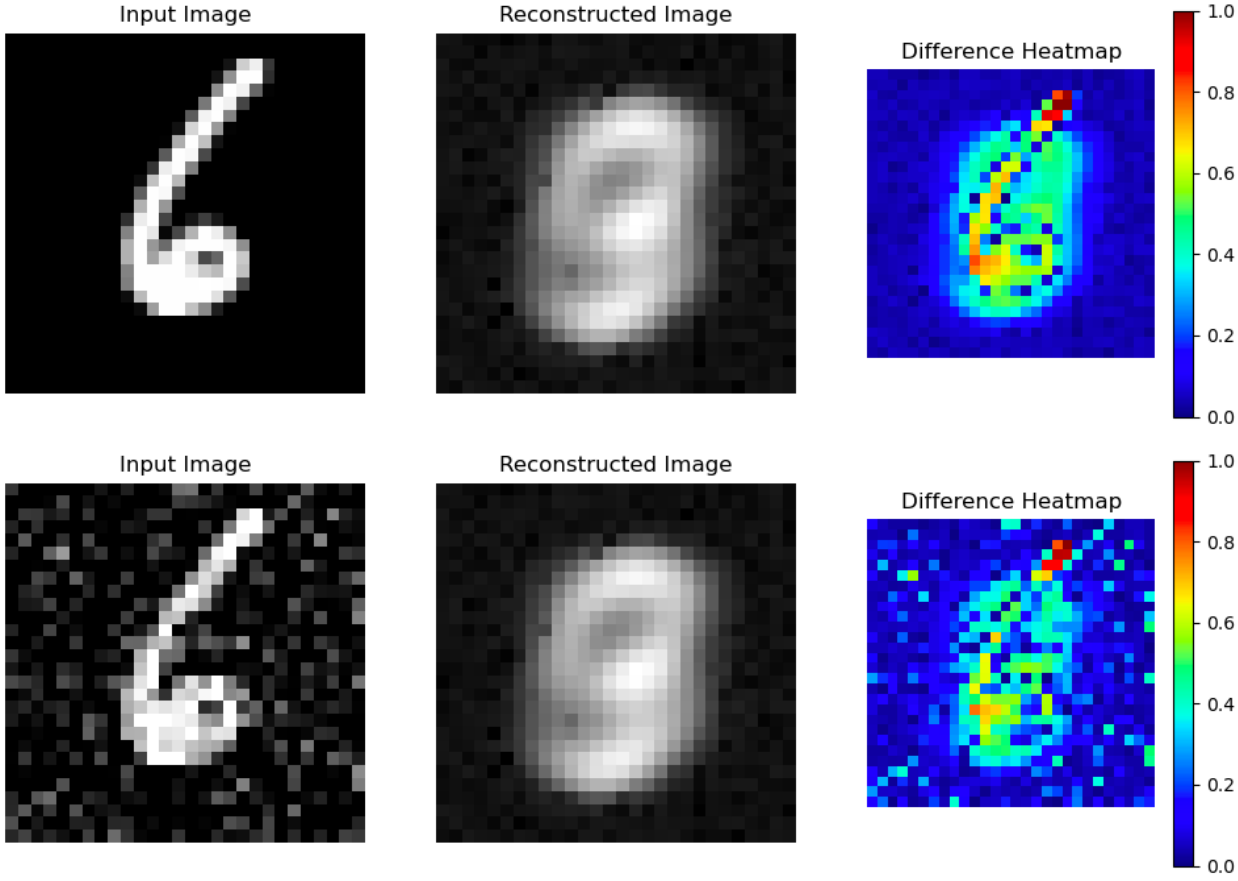

Figure 2 Results for Autoencoder_FCN_simple model

2.3 Comment on what happens if your dataset includes corrupted images not only in the testing dataset but also in the training dataset? How does it affect the performance of the model?

Addition of noisy images may lead to model learning to reconstruct noise along with original features. If the noise value is too strong, and amount of noisy images is to big model would try to learn of off random images, detreating autoencoder performance much more.

If model trained both on original and noisy images would be used in the context of anomaly detection, it might struggle to distinguish between normal and unexpected data, as it has learned to treat noise as normal.

2.4 Implement the SSIM loss function yourself without using any libraries (which directly include SSIM function) or in-built Python functions. Obtain the results by SSIM instead of the MSE loss function. Explain the difference between them and compare the results. Plot your training and validation losses to epoch numbers for both loss functions. Please keep your loss function as MSE for the following questions.

Did not attempt.

2.5 Implement deeper autoencoder architecture to obtain better results. Explain the improvements of the architecture. Visualize the results and loss function. Compare them with the first architecture you implemented. Comment on the results.

The improved model “Autoencoder_FCN_improved” increases its capacity by using more layers and larger hidden dimensions (e.g., 512, 256, 128), allowing it to capture more complex patterns and features. Additionally, batch normalization has been added after each linear layer in both encoder and decoder parts. Bottleneck remains at same size of 16 neurons. New decoder structure has layer sizes of 784 -> 512 -> 256 -> 128 -> 16, and decoder has accordingly layer sizes 16 -> 128 -> 256 -> 512 -> 784.

Results are greatly improved, with reconstructed images resembling originals. However, difference between reconstructing from original and noisy image is still hardly noticeable with loss increasing by only about 8% for noise value of 0.4.

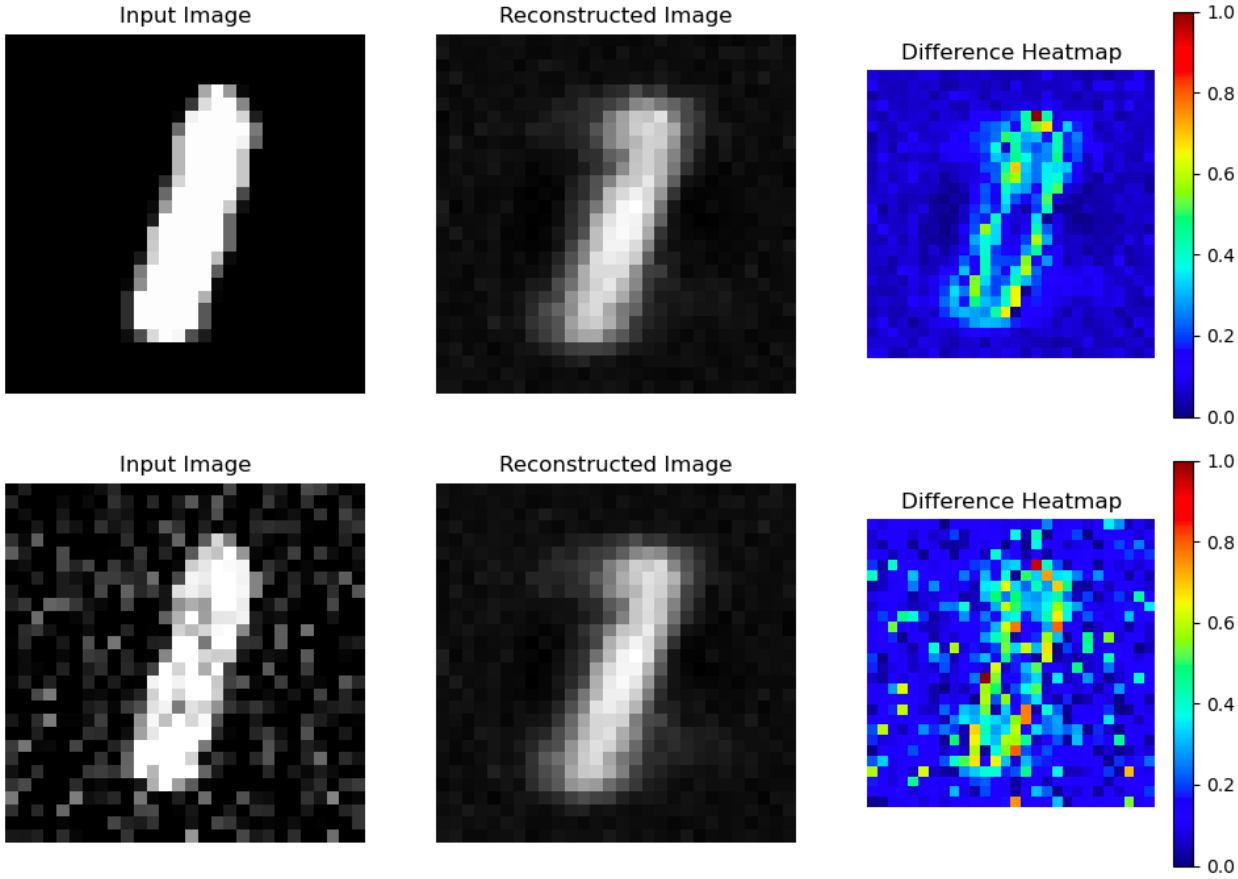

Figure 3 Results for Autoencoder_FCN_improved model

2.6 Implement the convolutional autoencoder model. Provide details of the architecture. Visualize and compare the results with basic and complex architectures you implemented. Comment on the results.

“Autoencoder_CNN” model uses encoder that transforms the input from 28x28x1 to 14x14x16 using a convolutional layer with ReLU, then to 7x7x32 with another convolutional layer and ReLU. The decoder reconstructs the image, upsampling from 7x7x32 to 14x14x16 using a transposed convolutional layer with ReLU, and finally back to 28x28x1 with another transposed convolutional layer and a sigmoid activation.

Resulting model is much simpler compared to “Autoencoder_FCN_improved”, yet it results with much better image reconstruction. However, some artifacts have been introduced in a form of regular grid, especially visible in Difference Heatmap. This can be caused by too short training or using too simple model.

Despite that comparing resulting noise between reconstructing from original and noisy images, a clear difference is visible. For noise value of 0.2 a difference in loss value of 250% has been observed. This indicates that CNN autoencoder has good potential for usage in anomaly detection.

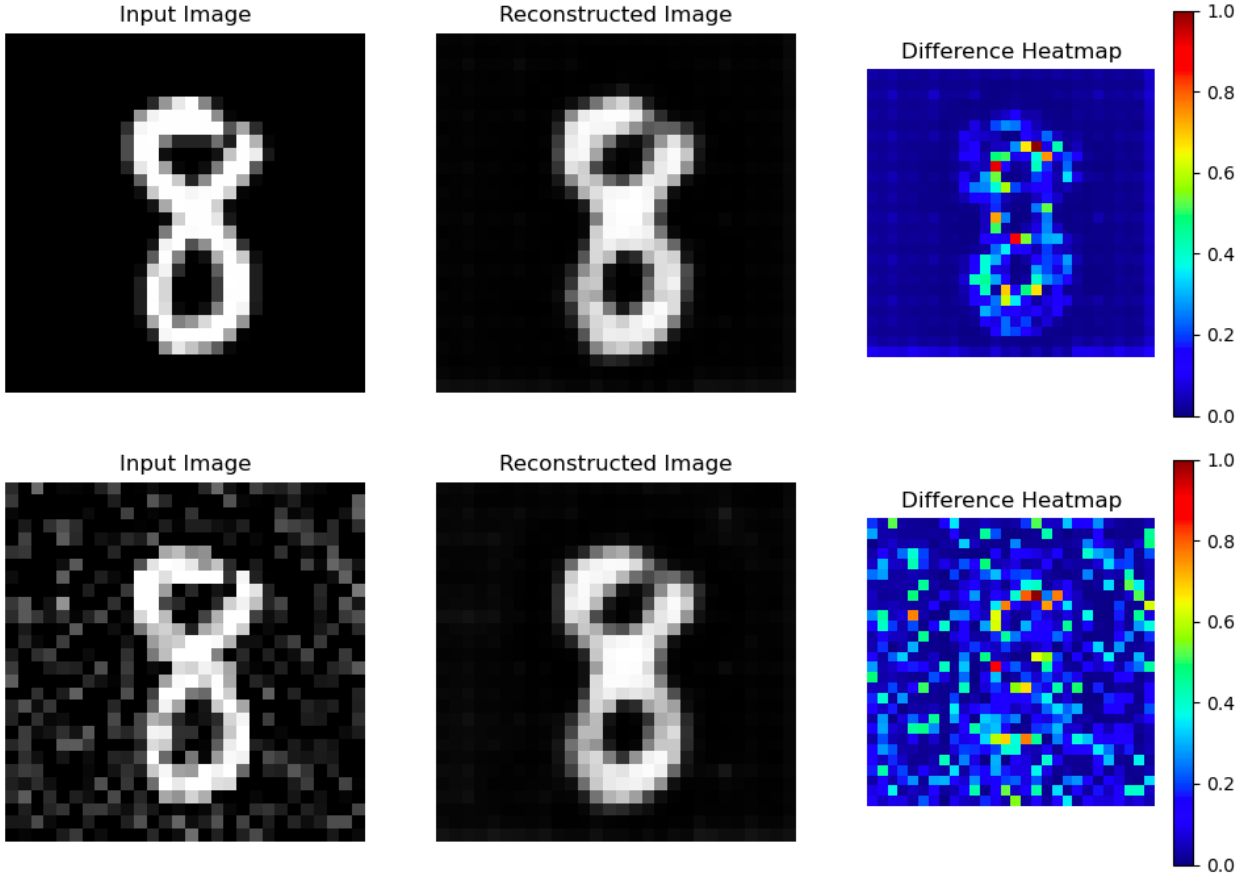

Figure 4 Results for Autoencoder_CNN model

2.7 Choose the best implementation based on the accuracy results by explaining the reason for that decision. Plot your best architecture's training and validation loss functions to epoch numbers. What is over-fitting? How can one detect and avoid over-fitting? Please answer the rest of the questions based on the best implementation

Best results came from implementation using CNNs. While for encoder-decoder structures accuracy as its usually understood in context of NN can be computed directly, both loss values and visual implementations were by far the best in this case.

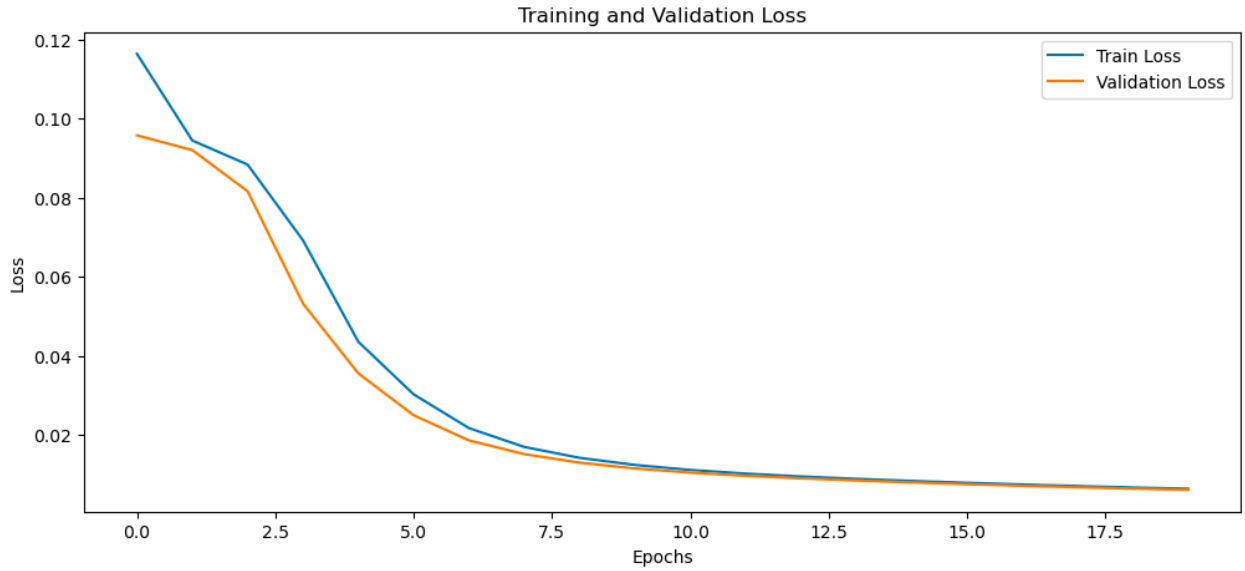

Figure 5 Autoencoder_CNN model training

Overfitting happens when a model learns noise or overly specific patterns from training data, failing to generalize to new data. In CNNs, it occurs with overly complex architectures or insufficient data, while in autoencoders it results in perfect reconstruction of training data but failure on previously unseen or noisy data. In context of anomaly detection any previously unseen data labelled as normal could be interpreted as anomaly.

To prevent overfitting, techniques such as dropout, regularization, and data augmentation can be used. Additionally, early stopping is a useful method that halts training when the model's performance on validation data plateaus. For autoencoders, limiting the capacity of the bottleneck layer or applying denoising techniques can enhance generalization, while in anomaly detection, maintaining a balanced dataset with an adequate mix of normal and anomaly samples is crucial to avoid overfitting to a single type of data.

2.8 Decide a threshold value in reconstruction error for detecting outliers (corrupted images) by drawing the loss distribution for normal and corrupted images. This threshold value should be used to determine whether the input sample is an anomaly or not. If there is no overlap between samples and your model does the job perfectly, please play with the noise level to have overlap so that you can play with different thresholds. Then, please explain why you decided on that threshold value by comparing it with other threshold values.

Figure 6 Test and noisy set loss distribution

Selected threshold value is equal to 0.016. During anomaly detection, false positives are much less problematic than false negatives, therefore threshold could realistically be set even lower (current value is used to identify at least one FN).

2.9 Provide true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) values for the threshold that you decided in the previous question. Plot the confusion matrix for test set.

True Positives = 99
False Positives = 5
True Negatives = 995
False Negatives = 1

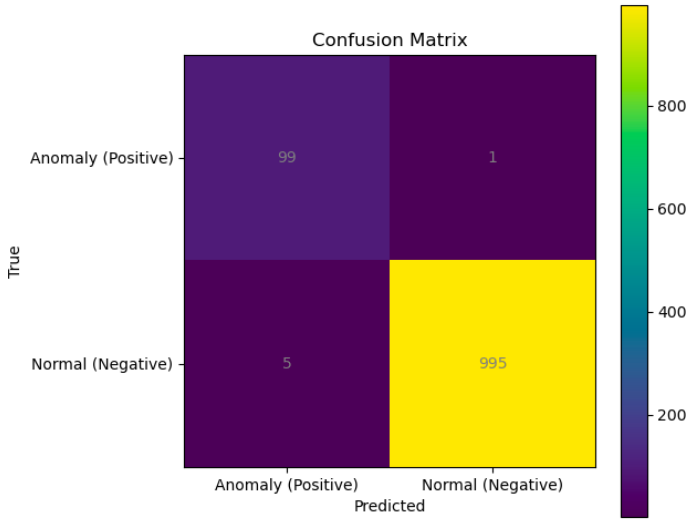

Figure 7 confusion matrix for test and noisy set

Confusion Matrix values don't add up to test set size of 1000, but to 1100 due to creating two separate sets instead of adding noise to test set.

2.10 Calculate precision, recall, and F1 score. Explain in detail how you can improve the F1 score. What is the effect of reconstruction error threshold value for precision and recall? Explain in detail.

Precision: 0.952
Recall: 0.990
F1 Score: 0.971