Eindhoven University Of Technology

Computer Vision and 3D Data Processing

Anomaly Detection with Autoencoder

2064472 – R.Regalo

February 3, 2024

I. Introduction

II. Question 1

To test our anomaly detector we need to have some ”anomaly” in our data. In this project, we are using the MNIST handwritten digit dataset and the anomalies will be corrupted images. The images will be corrupted by heavily adding noise. We considered salt and pepper noise which is a common occurrence in image processing and therefore an interesting case. Figure 1 shows examples of images before and after the addition of noise.

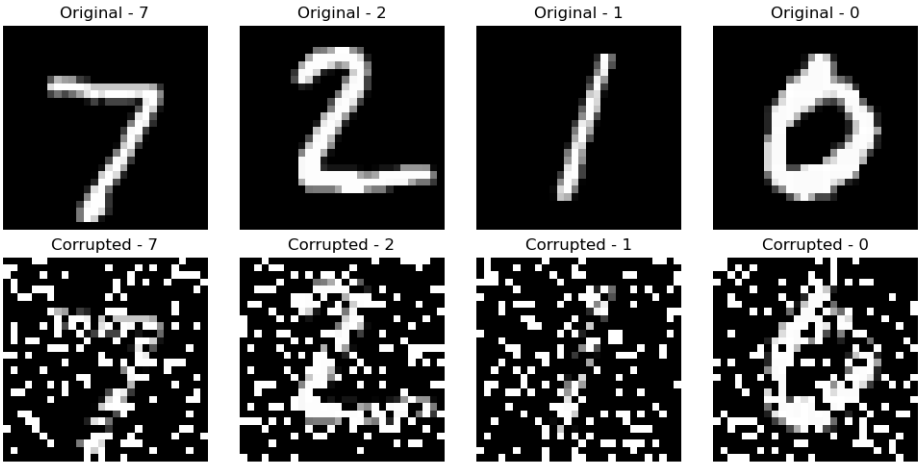

Figure 1: Title.

III. Question 2

Now, we trained the previously implemented autoencoder using the MNIST training dataset. We then obtained the output of the autoencoder for corrupt and non-corrupt images, and some examples are represented in figure 2

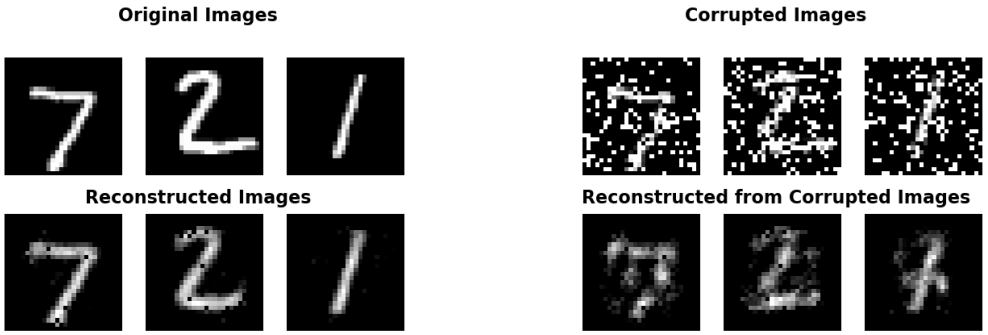

Figure 2: Simple Linear Autoencoder

We can see that the reconstructed images look fairly similar to the original ones, to the point that we

can still clearly identify each digit. We expected the autoencoder to be able to reconstruct the original images from the original dataset because they are the same kind of images that were used to train it. We also expect the autoencoder to be able to recreate the original image even when we give it the corrupted ones. This is because autoencoders work by reducing the image into a very compact feature space where only the most important features of the image are represented. Because the noise is not represented in this feature space, it is not reconstructed. Even so, it's noticeable that the output for the corrupted images is noisier than the uncorrupted ones.

IV. Question 3

We wanted to test how the model would perform if the training set included corrupted images. For that, we took 10% of the images in the training set and corrupted them by adding severe noise (as in question 1). The obtained results are in figure 3. We can see that the reconstruction of uncorrupted images remains the same, however, the reconstruction of corrupted images gets worse. This is because when training, the autoencoder will learn the noise as one of the features and thus be able to recreate it in the output.

Figure 3: Simple Linear Autoencoder with corrupted trainset

V. Question 4

The Structural Similarity Index Measure (SSIM) is a metric for evaluating the perceptual quality of digital images and videos differing from traditional pixel-based metrics such as MSE. The SSIM index focusses on assesing variations in structural information, luminance and contrast between the compared images. This approach aligns more closely with the human visual system's natural capacity for structural information extraction, thereby offering a more meaningful measure of image quality.

Formally, the SSIM index is computed across various windows within an image with the comparison between two windows x and y of equal size n n being defined as

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

where:

- μ_x and μ_y are the average pixel values in windows x and y, respectively.
- σ_x^2 and σ_y^2 are the variance of x and y.
- σ_{xy} is the covariance of x and y.
- C_1 and C_2 are constants used to stabilize the division with weak denominators; they depend on the dynamic range of the pixel values.

SSIM loss encourages models to produce outputs that not only align with the structural and perceptual attributes of the target images but also adhere to the visual interpretation preferred by the human eye consequently, this methodology often yields results that are visually more appealing and representative of the original images.

Training the previous model with SSIM loss led to the results in figure 4. We can see that there is no significant improvement in the quality of the images. This is due to the fact that the model is very simple, so there is not much room for improvement and fine-tuning.

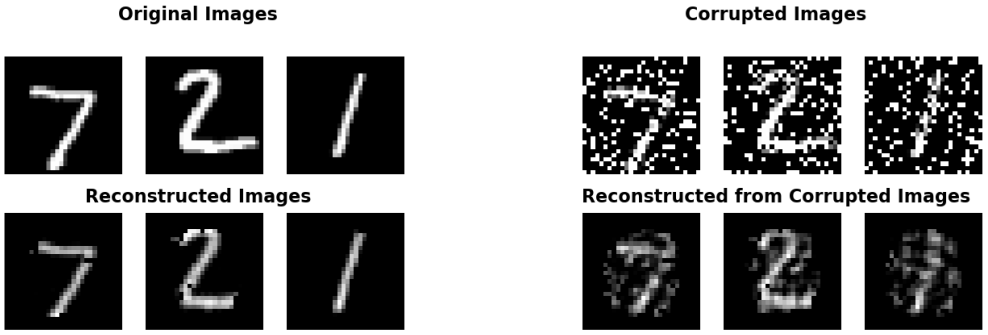

Figure 4: Simple Linear Autoencoder with SSIM loss function

VI. Question 5

Now, we'll aim to improve the performance of our autoencoder by implementing a more complex model whilst still using only fully connected layers. Apart from including more layers, we also applied batch-normalization and dropout layers. The optimizer was also changed to the Adam optimizer, which is known to have better performance. The results are shown in figure 5, where we can see that the reconstruction of the uncorrupted images is almost perfect. For the corrupted images, the reconstruction also slightly improved but is still not good, which shows the linear model's inability to learn only the important features.

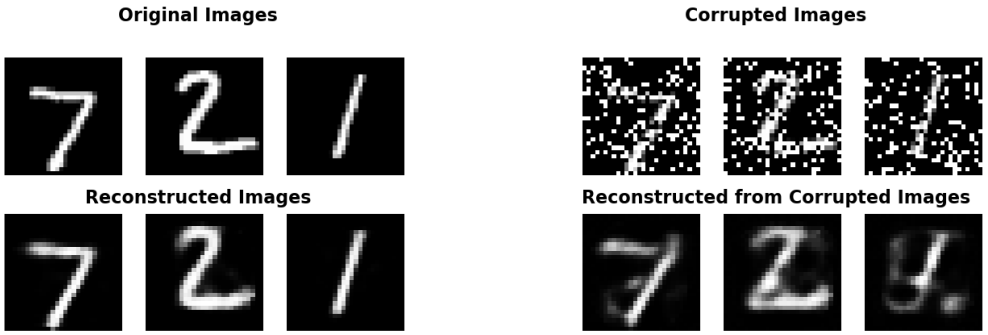

Figure 5: Complex Linear Autoencoder

VII. Question 6

We'll now implement the Convolutional Autoencoder (CAE). This autoencoder has convolutional layers followed by pooling layers to better extract the features of the images at different scales, just like a CNN. We decided to start simple with only two convolutional layers with MaxPooling in between them, which is a pretty standard implementation. Upon training and testing this model we verified that the autoencoder was learning the identity function, which is a known problem with autoencoders. Because we are trying to remove the noise in order to find anomalies this would prove useless. We then decided to change the training process to use some noisy images in the forward pass but still use the original for loss computation and backpropagation. This essentially forces the model to learn only the interesting features thereby filtering the noise. Using this result we were able to achieve the results in image 6

Figure 6: Convolutional Autoencoder

VIII. Question 7

To choose the best model, we started by qualitatively comparing the results of the different models. Immediately, we conclude that the best model is the CAE. To further support this claim, this model

is also the one that showed the lowest MSE. Training this model over 10 epochs leads to the losses shown in figure 7.

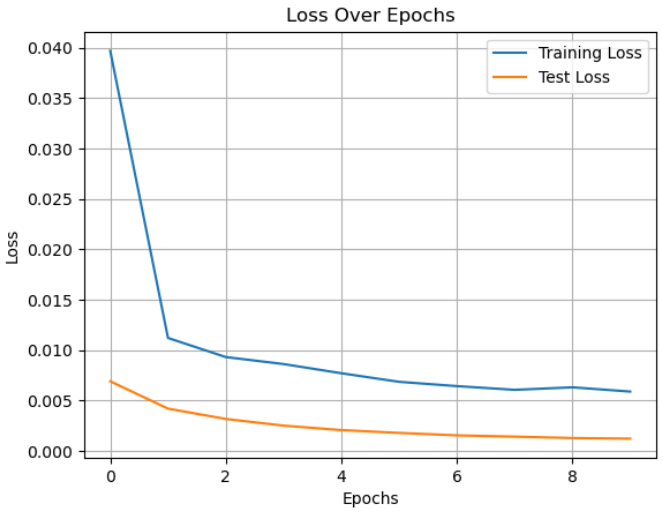

Figure 7: Title.

Over-fitting is when a model learns the training data too closely including its noise, leading to poor performance on unseen data. To detect over-fitting, we can look out for a significant gap between training and validation performance metrics and analyze learning curves for diverging trends. Looking at figure 7 we can conclude that our model is not over-fitting.

Strategies to avoid over-fitting include simplifying the model using regularization techniques like L1 or L2 regularization, dropout, data augmentation, cross-validation and even implementing early stopping during training.

IX. Question 8

Our objective is to implement an anomaly detector so we must come up with a way to decide if an image is corrupted or not based on the autoencoder's output. Because the autoencoder completely removes the noise, the MSE between the original and reconstructed images will always be lower in the case of uncorrupted images. This means that we can classify the image as an anomaly if the MSE error is above a certain threshold. To find this threshold we'll start by plotting the loss distribution for images on the original testset and noisy testset, represented in figure 8. As we can see, there is a clear separation between the two so we have multiple options to choose our threshold. This is a 1D classification problem that can be solved simply by picking the value that maximizes the margin to each distribution (like SVMs do). Because this distributions are computed based on a limited test dataset, there is always a chance that in reality, we'll get a sample that will fall outside the distributions shown in the figure, and choosing a threshold too close to each curve would then incorrectly classify that sample.

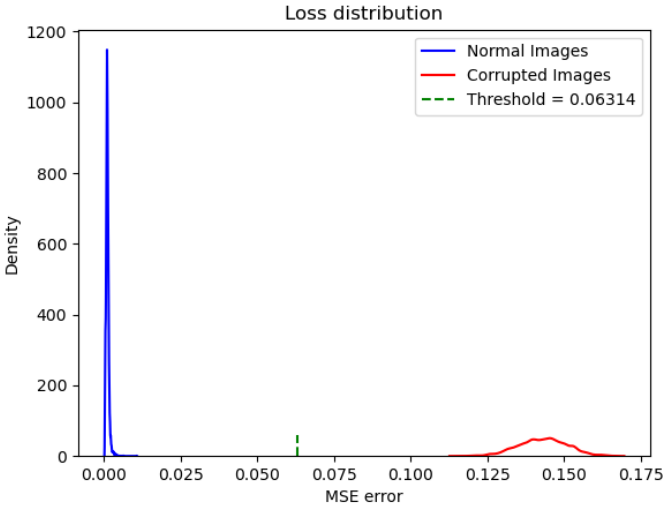

Figure 8: Title.

X. Question 9

In order to evaluate the performance of our model we started by obtaining the number of true positives (TP), true negatives(TN), false positives(FP), and false negatives(FN). For that, we computed the model output for the original MNIST test set as well as for the corrupted dataset. The results are presented in the confusion matrix in figure 9

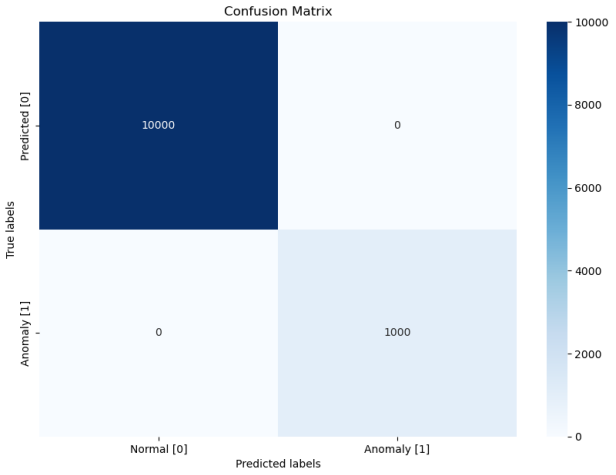

Figure 9: Title.

XI. Question 10

The formulas for precision, recall, and F1 score are

$$\text{precision} = \frac{TP}{TP + FP}, \quad \text{recall} = \frac{TP}{TP + FN}, \quad \text{F1 score} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}.$$

Given the previously computed confusion matrix, we get that precision $=$ recall $=$ F1 score $= 1$. This means that our model perfectly identifies all anomalies. This behavior is uncommon but is explained due to the simplicity of the task. Typically, the F1 score is not 1 and improving it directly translates into improving the model performance.

Improving the F1 score involves improving both recall and performance and the relationship between them. Training techniques like data augmentation, using pre-trained weights taken from other models, and using application-specific custom loss functions can help improve performance. Considering ensemble methods for more robust predictions or post-processing, the outputs may also improve the F1 score.

Choosing the threshold value for the reconstruction error to consider the image an anomaly is extremely important. Lowering the threshold leads to more instances being classified as anomalies, thus increasing recall but decreasing precision. Conversely, raising the threshold will increase precision but decrease recall. It is important to find the threshold that balances precision and recall to achieve a higher F1 score.

XII. Question 11

The ROC curve and AUC are important metrics for assessing the performance of binary classification models. The ROC curve plots the true positive rate against the false positive rate at various threshold settings, illustrating the trade-off between correctly identifying positives and falsely identifying negatives. It provides a graphical representation of a classifier's ability to distinguish between the two classes across all possible thresholds. On the other hand, the AUC summarises this performance into a single scalar value ranging from 0 to 1, where a higher value indicates better discrimination between positive and negative classes. An AUC of 1 means perfect classification, while 0.5 means no better accuracy than random guessing.

XIII. Question 12

The false alarm rate, or false positive rate, measures how often the model incorrectly classifies normal images as anomalies and is tied to the selected threshold. This is very important for anomaly detection because a high false alarm might lead to wasting resources (like in the case of a security camera). Conversely, if the threshold is too low, it may lead to some anomalies being classified as normal, which can have a seriously negative impact in some cases (healthcare, for example). Therefore, the ideal false

alarm rate depends on the application and should be carefully chosen to improve performance.

XIV. Question 13

Despite the versatility of autoencoders for dimensionality reduction and anomaly detection, they have some noticeable limitations. The first common problem is learning the identity function. Sometimes, autoencoders learn how to simply copy the input to the output without capturing important features in the data. This may especially happen if the model is too complex for the available data. If this happens, the autoencoder is rendered useless. Another noticeable problem with autoencoders is their struggle to learn complex data distribution. Basic autoencoders may focus on the dominant features and miss smaller but important details. This hinders their performance in tasks requiring a thorough understanding of the data.

XV. Question 14

Apart from autoencoder, there are other solutions for the anomaly detection problem. One technique that can be used is the Isolation Forest. It assumes that the outliers are few and different from each other. The algorithm randomly selects a split value between a certain feature's maximum and minimum values and splits the data into two based on this value. This leads to anomalies being isolated earlier in the tree, resulting in shorter paths in the tree structure, which makes them easy to identify. Isolation forests work by building an ensemble of isolation trees that each isolate and detect anomalies based on the paths of each sample. They are particularly efficient for high-dimensional data and have a low computational cost.

Another option for anomaly detection is the Local Outlier Factor algorithm. It considers outliers as points with a substantially lower density than their neighbors. The LOF score of a point is computed based on the ratio of the average local density of its neighbors to its local density, with higher scores indicating a higher likelihood of being an outlier. This method effectively detects outliers that may not be distinguishable based on global measures due to the locality aspect of the algorithm, making it particularly useful in datasets where the concept of an outlier is not globally defined but rather context-dependent on the local data structure.

References

[1] Pytorch documentation

[2] https://www.geeksforgeeks.org/implementing-an-autoencoder-in-pytorch/

[3] https://medium.com/pytorch/implementing-an-autoencoder-in-pytorch-19baa22647d1

[4] https://www.kaggle.com/code/alaaeddinebenzekri/image-denoising-using-autoencoder-pytorch