

# TrustMAE: A Noise-Resilient Defect Classification Framework using Memory-Augmented Auto-Encoders with Trust Regions

Daniel Stanley Tan<sup>1,2</sup>, Yi-Chun Chen<sup>2</sup>, Trista Pei-Chun Chen<sup>2</sup>, Wei-Chao Chen<sup>2,3</sup>

<sup>1</sup>National Taiwan University of Science and Technology, Taiwan

<sup>2</sup>Inventec Corporation, Taiwan

<sup>3</sup>Skywatch Innovation Inc., Taiwan

d10515805@mail.ntust.edu.tw, yichun8447@gmail.com

chen.trista@inventec.com, weichao.chen@skywatch24.com

## Abstract

*In this paper, we propose a framework called TrustMAE to address the problem of product defect classification. Instead of relying on defective images that are difficult to collect and laborious to label, our framework can accept datasets with unlabeled images. Moreover, unlike most anomaly detection methods, our approach is robust against noises, or defective images, in the training dataset. Our framework uses a memory-augmented auto-encoder with a sparse memory addressing scheme to avoid over-generalizing the auto-encoder, and a novel trust-region memory updating scheme to keep the noises away from the memory slots. The result is a framework that can reconstruct defect-free images and identify the defective regions using a perceptual distance network. When compared against various state-of-the-art baselines, our approach performs competitively under noise-free MVTec datasets. More importantly, it remains effective at a noise level up to 40% while significantly outperforming other baselines.*

## 1. Introduction

For manufacturing facilities, product appearance assessment is an essential step in quality assurance. Undetected defects, such as scratches, bumps, and discolorations, can result in costly product returns and customer trust loss. Today, humans still perform most appearance inspection tasks because of the difficulty of describing various defects using traditional computer vision algorithms in the automatic optical inspection machines (AOIs). However, managing human inspectors has been a significant management problem because it is difficult to maintain a consistent inspection standard across different product lines.

At first, it appears that one could address the problem with the ever-popular object detector networks [9, 10, 24,

37]. However, these fully-supervised models [21, 30, 31] require datasets with clearly annotated bounding boxes, which can be laborious and equally tricky to label with consistency. Additionally, because these methods tend to perform poorly for defects not present in the dataset, it can take an indefinite amount of time to collect sufficient training data. In many cases, we have experienced months of delays between data collection and model acceptance by the facilities, which is unacceptable for products with relatively short life cycles.

Instead of relying on explicitly labeled defects, it is possible to learn the distribution from normal samples and treat those deviating too far as defects, thus enabling the models to detect previously unseen defects [8, 13, 19]. A popular candidate is an auto-encoder that can erase defects from the input images upon trained with normal images [3, 15, 25, 27, 44, 47]. However, in practice, the auto-encoders can become overly general and learn to reconstruct the defects [12]. In particular, when the surfaces contain lots of texture [2, 3], the reconstructions can be erratic, leading to many false-positives.

Also, while the generative approaches do not require detailed labeling of the images, they often assume the input data are free of defective images. As a result, the algorithms can become overly sensitive to noise when defective images accidentally leak into the dataset, which frequently occurs in many manufacturing facilities. Furthermore, many input images tend to contain some imperfections, and if we need to detect these as defects, the percentage of normal images would undoubtedly drop. Thus, it becomes essential for our approach to be resilient to data noise, as the defect-free image regions can also be helpful for training purposes.

To address these issues using auto-encoders, we need to limit the latent space effectively such that it can still reconstruct normal image regions without over-generalizing to defects. For this purpose, we devise a scheme inspired

by MemAE [12] with several significant differences. We adopt a memory store for the latent space, but during the update phase, we increase the sparsity such that the updated information focus only on essential memory slots. Additionally, we add trust regions, which essentially classify defective latent space samples and avoid noisy samples from polluting the memory slots. To solve the textured surface problem, we further employ a perceptually-based distance approach to effectively separate defects from normal. Figure 1 illustrates the proposed TrustMAE method. We can achieve competitive results on the MVTec [2] dataset across all product types against baselines. Our framework is resilient to noise, achieving good performance even when the training data contain over 40% of defective images. We further include an ablation study in the results to demonstrate the effectiveness of these ideas. Our contributions are as follows:

- A defect classification framework resilient to noise in the training dataset,
- A novel sparse memory addressing scheme to avoid over-generalization of memory slots for the auto-encoder,
- A memory update scheme using trust regions to avoid noise contamination, and
- A perceptual-based classification method to effectively detect defective high-level regions for textured images.

## 2. Trust Memory Auto-Encoder (TrustMAE)

Given a dataset containing both normal and defective images, we aim to learn a model to separate normal from defective images without accessing labels that differentiate the two. Moreover, by treating small imperfections as defects, we reduce the percentage of defect-free images in the dataset. Consequently, it would be desirable to utilize both the normal images and the good image patches within the defective images to increase the available data for training, which means the model should be robust against noise (defect images).

Figure 1 shows an overview of our proposed method, which contains the following three components:

1. A memory-augmented auto-encoder (Section 2.1) with sparse memory addressing. It stores normal image features in a memory bank and reconstructs a normal version of the input image from memory. Large differences between the input and its reconstruction indicate potential defects.
2. Trust region memory updates (Section 2.2). This update process prevents the memory from storing unwanted defect features.

3. Spatial perceptual distance (Section 2.3). It produces patch-level differences using deep features between the input image with its reconstructed normal, making our methods more robust to slight shifts and color inaccuracies than shallow pixel-wise distances such as mean square error.

Next, we describe these components and how they can increase robustness against data noise.

### 2.1. Memory-Augmented Auto-Encoder

The standard auto-encoder follows an encoder-decoder structure wherein an encoder  $E$  projects the input image  $x \in \mathbb{R}^{H \times W}$  into a lower-dimensional latent vector  $z$ , and the decoder  $D$  produces an output image  $\hat{x} \in \mathbb{R}^{H \times W}$  which is a reconstruction of the input image  $x$ . To prevent defects from being reconstructed, we incorporate an external memory [12, 14] module  $M$  that stores a set of normal feature prototypes during training. We can think of these feature prototypes as puzzle pieces that the network uses to reconstruct the input image. At inference time, the set of feature prototypes are fixed, which makes it harder for the auto-encoder to reconstruct defects because the memory module only contains normal features, assuming it was trained on only normal images.

#### 2.1.1 Reconstructing from Memory

The memory module is implemented as a tensor  $\mathbf{M} \in \mathbb{R}^{M \times Z}$  where  $M$  denotes the number of memory slots, and  $Z$  denotes the dimensions of the latent vector  $z$ . Given an input image  $x$ , we first compute its latent representation  $\mathbf{Z} = E(x)$ . Note that in our setting, we want to preserve the spatial information so we designed our encoder to produce a feature map  $\mathbf{Z} \in \mathbb{R}^{h \times w \times Z}$  with a lower resolution than the original input image. For convenience, we use  $z_i \in \mathbb{R}^Z$  to refer to the  $i^{th}$  element of  $\mathbf{Z}$ . Each vector  $z_i$  in  $\mathbf{Z}$  represents features of a patch in the input image. Instead of passing the features map  $\mathbf{Z}$  directly to the decoder  $D$ , we compute approximate features  $\hat{z}_i$  for every  $z_i$  using a convex combination of the feature prototypes stored in the memory module. Eq. 1 defines the memory addressing operation, where  $w$  is a weight vector indicating how similar  $z$  is with each of the feature prototypes stored in the memory module  $\mathbf{M}$ . The decoder then outputs a reconstruction  $\hat{x} = D(\hat{\mathbf{Z}})$  using only approximates  $\hat{z}$  derived from the memory entries.

$$\hat{z} = w\mathbf{M}, \sum_{i=1}^M w_i = 1 \quad (1)$$

The weight vector  $w$  acts as a soft-addressing mechanism that retrieves the closest feature prototypes from the memory that are necessary for reconstruction. We measured

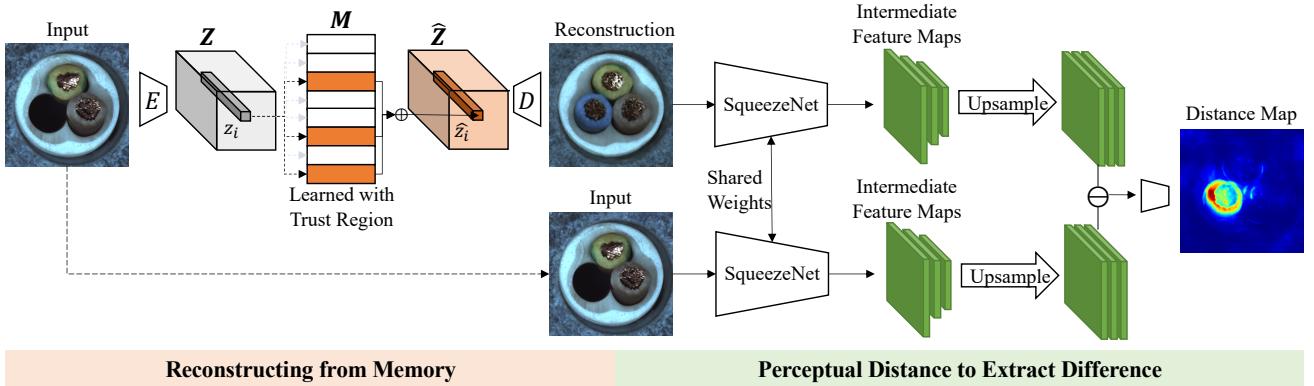


Figure 1: Framework of TrustMAE. Given a defective image, TrustMAE reconstructs a normal version of the input from memory. Then, it computes the perceptual difference between the input and its reconstruction, generating a distance map where large errors are indicative of defects.

the similarity between the feature vector  $z$  and the memory items  $\mathbf{M}_i$  using negative euclidean distance and apply a softmax function to normalize the weights, as shown in Eq. 2.

$$w_i = \frac{\exp(-\|z - \mathbf{M}_i\|_2)}{\sum_{j=1}^M \exp(-\|z - \mathbf{M}_j\|_2)}, i = \{1, \dots, M\}. \quad (2)$$

### 2.1.2 Sparse Memory Addressing

Enforcing sparsity in memory addressing forces the model to approximate the feature vector  $z$  using fewer but more relevant memory items. It effectively prevents the model from unexpectedly combining several unrelated memory items to reconstruct defects. Moreover, it implicitly performs memory selection [29], allowing us to save computation by removing items in the memory that were never accessed when reconstructing the training data.

Let superscript  $w^{(i)}$  denote an ordinal rank indexing of the elements of  $w$ , where  $w^{(1)} > w^{(2)} > \dots > w^{(M)}$ . We compute a sparse approximation  $\hat{w}$  of the weight vector  $w$  that corresponds to getting the  $k$  closest memory items followed by a re-normalization step, as shown in Eq. 3, where  $\mathbb{1}$  is the indicator function that returns a value of 1 if the condition inside is true and 0 otherwise.

$$\hat{w}^{(i)} = \frac{w^{(i)} \mathbb{1}\{i \leq k\}}{\sum_j w^{(j)} \mathbb{1}\{j \leq k\}} \quad (3)$$

Since we are only using a select few memory items for reconstruction, it is desirable to prevent the model from learning redundant memory items. Therefore, we impose a margin between the closest memory item  $\mathbf{M}^{(1)}$  and second closest memory item  $\mathbf{M}^{(2)}$  with respect to the input latent vector  $z$ .

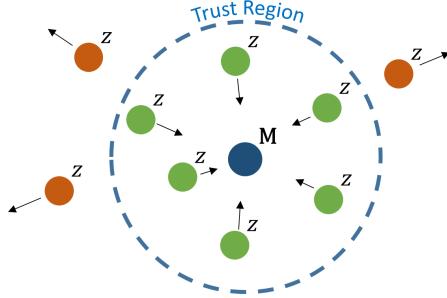


Figure 2: Illustration of the trust region memory updates.

$$L_{margin} = [\|z - \mathbf{M}^{(1)}\|_2 - \|z - \mathbf{M}^{(2)}\|_2 + 1]_+ \quad (4)$$

### 2.2. Trust Region Memory Updates

Without the assumption that the training data set only contains normal samples, the memory auto-encoder will treat defective samples as normal and learn to store defect features into the memory, leading to poor defect detection performance. We leverage on two key observations to prevent defective samples from contaminating the memory: (1) Defects are rare and they do not always appear in the same location, which means that the proportion of defects at the patch level will be significantly smaller than the proportion of defects at the image level. (2) Normal data have regularity in appearance, making it easier for the memory auto-encoder to reconstruct normal as compared to defects during the early stages of training. This implies that normal features are initially mapped closer to the memory items than defective features.

Based on these insights, we update memory items based on a specified trust region that pulls the features within the

region towards the closest memory item and push the features outside the region away from the memory item, as illustrated in Figure 2. More specifically, we split the feature space based on a distance radius with respect to each memory item in  $\mathbf{M}$ , as shown in Eq. 5. Everything within  $\delta_1$  radius are considered normal features that we want to pull closer, while everything outside are considered potential defects that we want to push farther. To prevent the model from pushing the defective features to infinity, we ignore everything farther than a predefined trust threshold  $\delta_2$ .

$$r(z, \mathbf{M}^{(1)}) = \begin{cases} 1 & \|z - \mathbf{M}^{(1)}\|_2 \leq \delta_1 \\ -1 & \delta_1 < \|z - \mathbf{M}^{(1)}\|_2 \leq \delta_2 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

We implement the trust region updates as an additional loss function defined in Eq. 6, where  $\mathbf{M}^{(1)}$  denotes the closest memory item in  $\mathbf{M}$  with respect to  $z$ .

$$L_{trust} = r(z, \mathbf{M}^{(1)}) \|z - \mathbf{M}^{(1)}\|_2 \quad (6)$$

Since patches that are easy to reconstruct tend to have smaller distances to the memory slots than patches that are harder to reconstruct, we require  $\delta_1$  to be adaptive to each of these cases. We set  $\delta_1$  to be the average distances between memory item  $\mathbf{M}_i$  with the features  $z_i$  in the current batch that retrieves  $\mathbf{M}_i$  as its closest memory item. Our main intuition is that, assuming normal features are abundant and are similar to each other, then normal features will mostly be pulled closer to the memory and only occasionally pushed out, but defect features will always be pushed out since they would always be farther than the average distance, thus, preventing contamination of the memory items.

### 2.2.1 Loss Functions

Following recent works on image synthesis [26], we employ several loss functions to encourage sharper images: Reconstruction loss  $L_{rec}$  [26], SSIM loss  $L_{sm}$  [3], VGG feature loss  $L_{vgg}$  [18], GAN loss  $L_{GAN}$  [13], and GAN feature loss  $L_{feat}$  [42, 43]. Please refer to the supplementary materials for more details on these loss functions. The total loss function is then defined in Eq. 7, where the  $\lambda$  co-efficients are hyper-parameters that control the relative weighting of each term. We used similar weightings as [26].

$$\begin{aligned} L_{total} = & \lambda_{rec} L_{rec} + \lambda_{sm} L_{sm} + \lambda_{vgg} L_{vgg} + \lambda_{GAN} L_{GAN} \\ & + \lambda_{feat} L_{feat} + \lambda_{margin} L_{margin} + \lambda_{trust} L_{trust} \end{aligned} \quad (7)$$

### 2.3. Spatial Perceptual Distance

The difference between the input and its normal reconstruction dictates whether it will be considered a defect

or not. Ideally, we want a distance measure that assigns high scores on defective regions while assigning low scores on normal regions. Previous approaches [2, 3, 12, 25] use shallow pixel-wise distances such as mean squared error, which is sensitive to slight global shifts and small reconstruction inaccuracies. As a result, they do not work well when our focus is on texture pattern similarity rather than exact pixel alignments. Therefore, we use a spatial perceptual distance [46] that computes differences on a patch level using deep features extracted from a pre-trained convolutional neural network. Computing deep features allows us to incorporate high-level information in measuring distance, which has been shown to correlate well with human perception [46], making it a more suitable distance measure. Moreover, operating on a patch level allows us to capture texture information that are impossible to get from shallow pixel-wise distances such as the mean squared error that only considers pixel intensities.

Let  $\phi_l(x) \in \mathbb{R}^{H_l \times W_l \times C_l}$  denote the features extracted from the  $l$ th layer of a convolutional neural network. The perceptual distance  $PD(x, \hat{x}) \in \mathbb{R}^{H \times W}$  is defined as

$$PD(x, \hat{x}) = \sum_l \gamma_l^T g(\phi_l(x) - \phi_l(\hat{x}))^2, \quad (8)$$

where  $g$  is a bilinear up-sampling function that resizes the feature tensors to  $H \times W$  and  $\gamma_l \in \mathbb{R}^{C_l}$  is a weight vector used to aggregate the channel-wise feature differences at layer  $l$ . Note that, for brevity, we abuse the notation of dot product to refer to the channel-wise aggregation, which can be implemented efficiently as a  $1 \times 1$  convolution. We adopt the weights of [46] for  $\gamma_l$  and  $\phi$ , which were optimized for the Berkeley Adobe Perceptual Patch Similarity (BAPPS) dataset.

We define the reconstruction error as a multiplicative combination of the mean squared error and the perceptual distance, which further highlights defective regions and suppress the non-defective regions.

## 3. Experimental Results

We demonstrate the effectiveness of our proposed method on a real-world **MVTec** [2] dataset consisting of 15 different industrial products. It contains over 70 different types of defects labeled with both defect types and segmentation masks. We use the original train-test splits to compare against prior works in a noise-free setting. But for the noisy setting, we re-sample half of the defect images as part of our training data since the original training set does not contain any defective image. The number of normal and defective images in the new training set are 3629 and 610, respectively. In the testing set, there are 467 normal images and 648 defective images.

Table 1: Image-based comparison in terms of AUC with several state-of-the-art baselines on the MVTec Dataset. (Best score in bold and the second best score in italic.)

| Method                            | mean AUC     | bottle       | cable        | capsule      | caper        | grid         | hazelnut     | leather      | metal_nut    | pill         | screw        | tile         | toothbrush   | transistor   | wood         | zipper       |
|-----------------------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| GeoTrans [11]                     | 67.23        | 74.40        | 67.00        | 61.90        | 84.10        | 63.00        | 41.70        | 86.90        | <b>82.00</b> | 78.30        | 43.70        | 35.90        | 81.30        | 50.00        | 97.20        | 61.10        |
| GANomaly [1]                      | 76.15        | 89.20        | 73.20        | 70.80        | 84.20        | 74.30        | 79.40        | 79.20        | 74.50        | 75.70        | 69.90        | 78.50        | 70.00        | 74.60        | 65.30        | 83.40        |
| ARNet [16]                        | 83.93        | 94.10        | 68.10        | <b>88.30</b> | 86.20        | 78.60        | 73.50        | 84.30        | <b>87.60</b> | 83.20        | 70.60        | 85.50        | 66.70        | <b>100</b>   | <b>100</b>   | <b>92.30</b> |
| f-Ano-GAN [33]                    | 65.85        | 91.36        | 76.37        | 72.79        | 56.57        | 59.63        | 63.15        | 62.50        | 59.70        | 64.07        | 50.00        | 61.34        | 67.31        | 77.92        | 75.00        | 50.00        |
| MemAE [12]                        | 81.85        | 86.19        | 56.65        | 75.71        | 74.55        | <b>98.91</b> | <b>97.39</b> | <b>95.48</b> | 53.12        | 77.88        | <b>83.59</b> | 79.11        | <b>96.78</b> | 71.62        | 97.11        | 83.71        |
| <b>TrustMAE-noise free (max)</b>  | <b>90.78</b> | <b>96.98</b> | <b>85.06</b> | 78.82        | <b>97.43</b> | <b>99.08</b> | <b>98.50</b> | <b>95.07</b> | 76.10        | <b>83.31</b> | <b>82.37</b> | <b>97.29</b> | <b>96.94</b> | 87.50        | <b>99.82</b> | 87.45        |
| <b>TrustMAE-noise free (mean)</b> | <b>88.04</b> | <b>98.57</b> | <b>91.64</b> | <b>79.34</b> | <b>87.40</b> | 92.73        | 93.36        | 87.33        | 74.14        | <b>92.12</b> | 61.94        | <b>89.50</b> | 90.28        | <b>91.71</b> | 96.32        | <b>94.22</b> |

Table 2: Pixel-based comparison in terms of Area Under the Receiver Operating Curve (AUC) with several state-of-the-art baselines on the MVTec Dataset. (Best score in bold and the second best score in bold italic.)

| Method                           | mean AUC     | bottle       | cable        | capsule      | caper        | grid         | hazelnut     | leather      | metal_nut    | pill         | screw        | tile         | toothbrush   | transistor   | wood         | zipper       |
|----------------------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| AE-L2 [3]                        | 80.40        | 90.90        | 73.20        | 78.60        | 53.90        | 96.00        | 97.60        | 75.10        | 88.00        | 88.50        | 97.90        | 47.60        | 97.10        | 90.60        | 63.00        | 68.00        |
| AE-SSIM [3]                      | 81.83        | 93.30        | 79.00        | 76.90        | 54.50        | 96.00        | 96.60        | 71.00        | 88.10        | 89.50        | <b>98.30</b> | 49.60        | 97.30        | 90.40        | 64.10        | 82.80        |
| MemAE [12]                       | 85.74        | 85.04        | 71.17        | <b>92.95</b> | 81.16        | 95.56        | 97.15        | 92.91        | 78.99        | <b>93.25</b> | 95.57        | 70.76        | 94.77        | 66.73        | 85.44        | 84.65        |
| Towards Visually Explaining [22] | 86.07        | 87.00        | 90.00        | 74.00        | <b>78.00</b> | 73.00        | <b>98.00</b> | <b>95.00</b> | <b>94.00</b> | 83.00        | 97.00        | 80.00        | 94.00        | 93.00        | 77.00        | 78.00        |
| CNN Feature Dictionary [23]      | 78.07        | 78.00        | 79.00        | 84.00        | 72.00        | 59.00        | 72.00        | 87.00        | 82.00        | 68.00        | 87.00        | <b>93.00</b> | 77.00        | 66.00        | <b>91.00</b> | 76.00        |
| AnoGAN [34]                      | 74.27        | 86.00        | 78.00        | 84.00        | 54.00        | 58.00        | 87.00        | 64.00        | 76.00        | 87.00        | 80.00        | 50.00        | 90.00        | 80.00        | 62.00        | 78.00        |
| AE-SSIM Grad [6]                 | 86.38        | <b>95.10</b> | 85.90        | 88.40        | 77.40        | 98.00        | 96.60        | 60.20        | <b>92.00</b> | 92.70        | 92.50        | 62.60        | <b>98.40</b> | <b>93.40</b> | 73.80        | 88.70        |
| $\gamma$ -VAE Grad [6]           | 88.77        | 93.10        | 88.00        | 91.70        | 72.70        | 97.90        | <b>98.80</b> | 89.70        | 91.40        | <b>93.50</b> | 97.20        | 58.10        | 98.30        | <b>93.10</b> | 80.90        | 87.10        |
| AE-L2 Grad [6]                   | 88.77        | 91.60        | 86.40        | <b>95.20</b> | 73.40        | <b>98.10</b> | 98.40        | 92.10        | 89.90        | 91.20        | <b>98.00</b> | 57.50        | 98.30        | 92.10        | 80.50        | <b>88.90</b> |
| VAE Grad [6]                     | <b>89.29</b> | 92.20        | <b>91.00</b> | 91.70        | 73.50        | 96.10        | 97.60        | 92.50        | 90.70        | 93.00        | 94.50        | 65.40        | <b>98.50</b> | 91.90        | 83.80        | 86.90        |
| <b>TrustMAE-noise free</b>       | <b>93.94</b> | <b>93.39</b> | <b>92.85</b> | 87.42        | <b>98.53</b> | 97.45        | <b>98.51</b> | <b>98.05</b> | 91.76        | 89.90        | 97.63        | <b>82.48</b> | 98.10        | 92.72        | <b>92.62</b> | <b>97.76</b> |

### 3.1. Implementation Details

We train our model on unlabeled training data for 100 epochs using Adam optimizer with a learning rate of 0.0001 for the generator and 0.0004 for the discriminator. We set the batch size to 16, the number of memory slots to  $M = 64$ , and the latent dimension  $Z = 512$ . As shown by Gong et al. [12], the number of memory slots  $M$  have little effect in performance as long as it is large enough. Based on trial and error, using top  $k = 3$  memory slots work well. We set  $\delta_1$  to be the mean distances between the features and memory item and  $\delta_2 = 20$ . As we show in Section 3.6, the performance of the model is robust to the setting of  $\delta_2$ . We adopt a symmetric architecture for our encoder-decoder networks with three residual blocks in the bottleneck layer on both the encoder and decoder networks. We use five down-sampling layers in the encoder, except on grid, screw, pill, and capsule where we only use three down-sampling layers. We defer the discussion on this design choice to Section 3.7. Following previous works, we

evaluate the models using the area under the receiver operating characteristic curve (AUC).

### 3.2. Varying Noise Levels

We conduct experiments for defect classification on increasing noise levels by introducing more defects into the training data. For defect classification, we need to define a single defect score for an input image to indicate the likelihood of it being defective. We define our defect score as the spatial pooling of the reconstruction errors computed with the perceptual distance. Since there are limited number of defective images, we reduced the number of normal data instead to match the desired noise percentage for experiments with higher noise (defect) level (30% - 40%). Figure 4 shows defect classification results at varying noise levels. For auto-encoders trained using mean squared error (AE-MSE) and SSIM (AE-SSIM), their performance significantly drop as we increase the noise levels. Auto-encoders are sensitive to noise since defect information can be directly passed from the encoder to the decoder, thus,

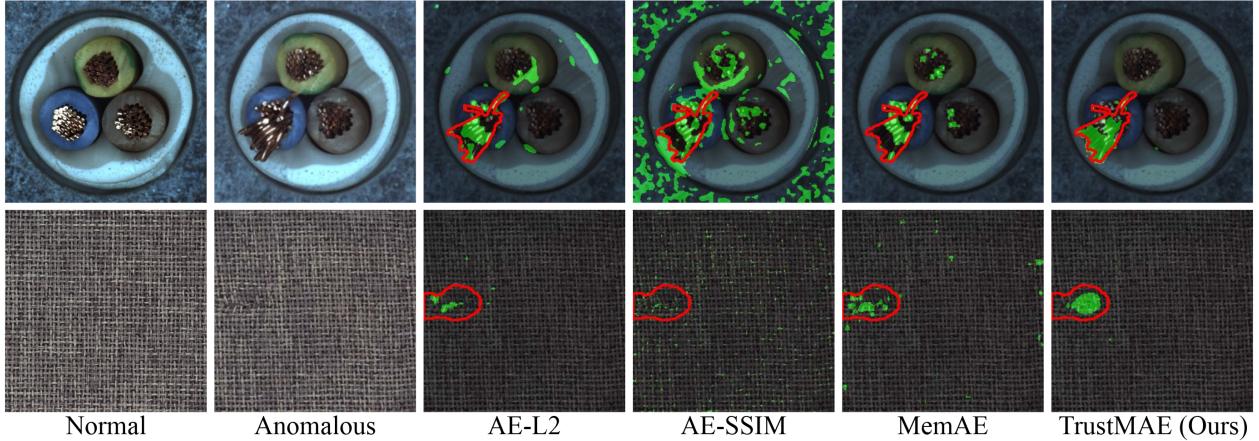


Figure 3: Visual comparison of defect segmentation results with state-of-the-art baselines. Ground-truth is represented by the red contour, and predicted defect segmentation is represented by the green overlay.

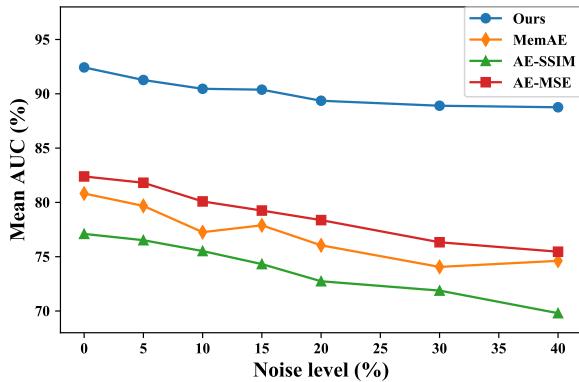


Figure 4: Comparison with baselines at different noise percentages.

easily reconstructing defects. MemAE [12] also does not perform well, because without enough sparsity and filtering mechanism, it could still memorize and reconstruct defects. Building on top of MemAE, our model enhances the noise robustness of the memory through our trust region memory updates, and uses a more suitable perceptual distance, thus, improving the overall performance, even at high noise levels.

### 3.3. Comparison with Baselines

To better compare with prior works, we report the performance of our model using spatial max pooling and mean pooling of the reconstruction errors as our defect score, which are the two most common pooling methods. We use the original train-test splits and train our model on only normal data (noise-free) to maintain a fair comparison. Table 1 shows our defect classification performance measured in terms of image level AUC in comparison with several re-

cent works on defect detection [1, 11, 12, 16, 33]. Our model with max pooling achieves the best AUC score on majority of the classes as well as the best mean AUC overall.

We analyzed the classes with lower performance (e.g. metal nut, pill, screw, and capsule), and observed that the model have difficulties modeling large variations in the normal data due to large rotations or randomly appearing prints. This is because the sparse memory addressing combined with the trust region memory updates restricts the variations that the memory can capture. Note that we can improve the performance by adding a warping module to align the data but it is outside the scope of this paper.

We also compared against prior works in the defect segmentation setting, which is evaluated using pixel-level AUC. To get a segmentation map, we can directly impose a threshold on the reconstruction errors. Table 2 shows our defect segmentation performance in comparison with several recent works on defect segmentation [3, 5, 6, 12, 19, 22, 23, 34]. We can observe that our method consistently performs competitively across all classes, achieving the best mean AUC among all the baselines.

We also show visual results on defect segmentation in Figure 3. Our model is able to capture a more complete segmentation of the defects as compared with the baselines, especially on textured objects. This is due to our spatial perceptual distance that can better measure texture similarity as opposed to shallow distance metrics that the baselines used.

### 3.4. Ablation Study

We conduct an ablation study to evaluate how each of our design choices affect the performance of the model. Figure 5 shows the defect classification results in terms of mean AUC across all classes at varying noise levels.

**Trust Region Memory Updates.** In the low noise regime (5% – 10% noise levels), we can observe similar

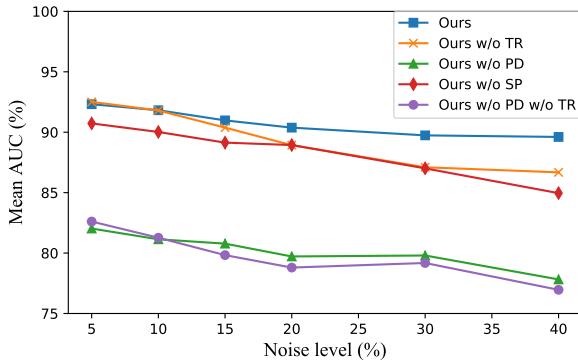


Figure 5: Ablation studies of our model trained with different levels of noisy data.

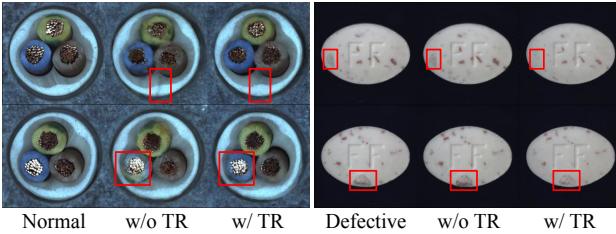


Figure 6: When the model is trained without trust region (TR), it could hallucinate defects when reconstructing from either normal data (left) or defective (right) data when training data are polluted with noises.

performance between the models with and without trust region memory updates (TR). However, the performance gap starts to widen as the noise levels increase. We observe that at higher noise levels, there are more chances for the memory slots to be contaminated with defective features. This causes the model to hallucinate defects (as shown in Figure 6 left) or reconstruct defects well (as shown in Figure 6 right), leading to poorer performance. Our proposed trust region memory updates (TR) makes the memory more robust to defect contamination, leading to better normal reconstructions, thus, better defect detection performance.

**Perceptual Distance.** Without the perceptual distance, we can observe a significant drop in performance across all noise levels. As we show in Figure 7, shallow pixel-wise distances, such as MSE, are inadequate for defect detection since they produce lots of noise across the whole image due to slight shifts and differences in textures between the input image and its reconstruction, leading to an increased number of false positives. Structural similarity index measure (SSIM) accounts for patch information but it normalizes distances independently for each patch, which makes it hard to separate defects since the error values are close to each other. Moreover, SSIM only accounts for the low

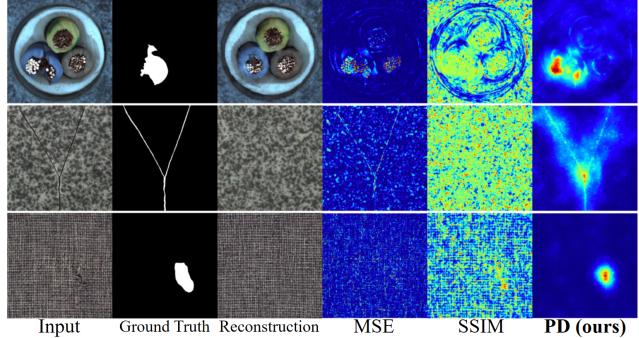


Figure 7: Visual comparison of the spatial perceptual distance (PD) with commonly used distance metrics such as mean squared error (MSE), and structural similarity index measure (SSIM).

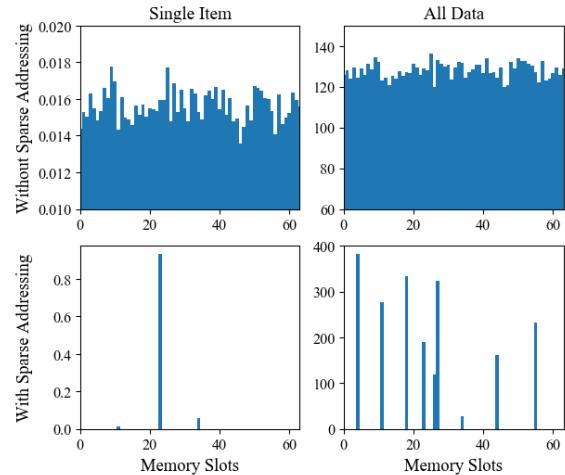


Figure 8: Memory Access with and without sparse addressing.

order statistics (means and covariances) of patch intensities, which are inadequate for defects that have very similar patch statistics with normal images such as the tile and carpet example shown in the bottom two rows of Figure 7. In contrast, we use deep features to compute the differences, allowing us to capture textures and other high level information. We can observe that the perceptual distance assigns noticeably higher values on defective regions and lower values on normal regions, making defects more pronounced and easier to separate.

### 3.5. Memory Access Patterns

We visualize memory access patterns with and without sparse addressing in Figure 8 for a single item  $z_i$  (left), and aggregated access frequencies across all the data (right). Without any restriction, the model tends to spread the information across all memory slots. This is because using

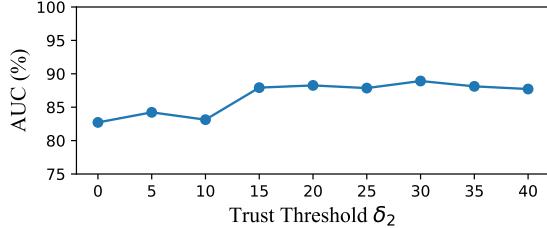


Figure 9: Effect of varying the trust threshold  $\delta_2$ .

more memory slots gives more flexibility for reconstruction. This in itself is not detrimental if the model is trained on only normal data. However, if the training data contains defective images then defect features will also spread to all the memory slots, making it harder to remove them. This is reflected in the model’s poorer performance without sparse addressing (SP), as shown in Figure 5. In contrast, sparse memory addressing forces the model to rely only a few memory slots. As a consequence, defect features can only affect a few memory slots, making it easier to prevent the memory from being contaminated with defect features.

### 3.6. Effect of Trust Threshold $\delta_2$

Figure 9 shows the results of varying  $\delta_2$  on cable at 30% noise level. We can observe that the model is quite robust in the range  $\delta_2 = [15, 40]$ , achieving approximately 88% AUC. Setting  $\delta_2 = 0$  is equivalent to ignoring everything outside the trust region, which performs worse since there is nothing encouraging the model to map defect features far from the memory items. We also tried removing  $\delta_2$ , but the model becomes unstable since it will just keep pushing the feature vectors to infinity.

### 3.7. Effect of Down-Sampling Layers

The number of down-sampling layers control the receptive field, and thus, the patch size that the memory slots operate on. More down-sampling corresponds to larger patch sizes, and vice versa. We observe that using fewer down-sampling achieves better results for classes that have many small details that greatly vary across different instances (e.g. grid, screw, pill, and capsule). This is because the memory combined with the trust region updates has the effect of limiting the variations of patch features that the memory can store. Without enough variations captured by the memory slots, it becomes difficult for the network to reconstruct varying small details accurately using limited set of large patch features.

## 4. Related Works

Techniques for defect detection can be broadly grouped into: classification-based [11, 24, 32, 35, 36], detection-based

[10, 37], segmentation-based [7, 10, 17, 20, 28, 38–40], and reconstruction-based [3, 5, 6, 23, 25, 34, 41, 47].

Our method belongs to the reconstruction-based, which relies on auto-encoders trained on normal data to reconstruct an image that is close to the normal data. It then classifies defects based on their reconstruction error. Schlegl et al. [34] and Chen et al. [4] combined auto-encoders with generative adversarial networks (GAN) [13] to better model the distribution of normal images. Likewise, Akcay et al. [1] used a conditional GAN and computed defect scores using the discriminator. Schlegl et al. [33] detected defects by combining discriminator feature error and image reconstruction error.

To prevent the model from reconstructing defects, Haselmann et al. [15] cropped out holes and inferred the pixels in the hole. Gong et al. [12] augmented auto-encoders with memory and reconstructs inputs based on features stored in its memory. Ye et al. [16] restored original data by forcing the network to learn high-level feature embeddings of selected attributes. While existing approaches work well under the assumption that training data only contains normal samples, they tend to perform poorer when training data are polluted with noise, which is common in reality. We propose trust region memory updates to further increase model’s resistance to pollution in training data.

Other than modeling normal images, different ways to compute distances are also gaining focus. Several works [2, 3, 12, 25] used shallow pixel-wise distances such as MSE. However, they are sensitive to slight shifts, making it unsuitable for texture similarity. Bergmann et al. [3] used SSIM, while Liu et al. [22] used gradient attention maps computed from image embeddings to segment defects. In contrast, we propose to use spatial perceptual distance that computes patch-wise differences of deep features.

## 5. Conclusion

We presented TrustMAE, a noise-resilient and unsupervised framework for product defect detection. It addresses the common defect data collection problem using unlabeled images while enabling a mixture of defective images within the training dataset. Our framework compares favorably to state-of-the-art baselines, and we showed the effectiveness of its components via an extensive ablation study.

We believe the noise-resilience property is a novel and useful one because the normal-defective classification of the training images can change according to practical application requirements. While most anomaly detection frameworks require noise-free datasets, we hope to gain more insights into these methods and create a noise-robust framework that performs well across all product classes soon.

## References

- [1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Gandomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018.
- [2] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtac ad—a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9592–9600, 2019.
- [3] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Carsten Steger. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, pages 372–380, 2019.
- [4] Xiaoran Chen and Ender Konukoglu. Unsupervised detection of lesions in brain mri using constrained adversarial auto-encoders. In *International Conference on Medical Imaging with Deep Learning, MIDL 2018, 4-6 July 2018, Amsterdam, Netherlands*, 2018.
- [5] Bin Dai and David Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*, 2019.
- [6] David Dehaene, Oriel Frigo, Sébastien Combexelle, and Pierre Eline. Iterative energy-based projection on a normal data manifold for anomaly localization. *arXiv preprint arXiv:2002.03734*, 2020.
- [7] Xinghui Dong, Chris J Taylor, and Tim F Cootes. Small defect detection using convolutional neural network features and random forests. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.
- [8] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Mastropietro, and Aaron C. Courville. Adversarially learned inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
- [9] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H Law. Automatic localization of casting defects with convolutional neural networks. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1726–1735. IEEE, 2017.
- [10] Max K Ferguson, Ak Ronay, Yung-Tsun Tina Lee, and Kincho H Law. Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning. *Smart and sustainable manufacturing systems*, 2, 2018.
- [11] Izhak Golan and Ran El-Yaniv. Deep anomaly detection using geometric transformations. In *Advances in Neural Information Processing Systems*, pages 9758–9769, 2018.
- [12] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. *arXiv preprint arXiv:1904.02639*, 2019.
- [13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.
- [15] Matthias Haselmann, Dieter P Gruber, and Paul Tabatabai. Anomaly detection using deep learning based image completion. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1237–1242. IEEE, 2018.
- [16] Chaoqin Huang, Fei Ye, Jinkun Cao, Maosen Li, Ya Zhang, and Cewu Lu. Attribute Restoration Framework for Anomaly Detection. *arXiv e-prints*, page arXiv:1911.10676, Nov. 2019.
- [17] Yibin Huang, Congying Qiu, Yue Guo, Xiaonan Wang, and Kui Yuan. Surface defect saliency of magnetic tile. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 612–617. IEEE, 2018.
- [18] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.
- [19] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
- [20] Jie Lei, Xin Gao, Zunlei Feng, Huamou Qiu, and Mingli Song. Scale insensitive and focus driven mobile screen defect detection in industry. *Neurocomputing*, 294:72–81, 2018.
- [21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [22] Wenqian Liu, Runze Li, Meng Zheng, Srikrishna Karanam, Ziyuan Wu, Bir Bhanu, Richard J Radke, and Octavia Camps. Towards visually explaining variational autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8642–8651, 2020.
- [23] Paolo Napoletano, Flavio Piccoli, and Raimondo Schettini. Anomaly detection in nanofibrous materials by cnn-based self-similarity. *Sensors*, 18(1):209, 2018.
- [24] Vidhya Natarajan, Tzu-Yi Hung, Sriram Vaikundam, and Liang-Tien Chia. Convolutional networks for voting-based anomaly classification in metal surface inspection. In *2017 IEEE International Conference on Industrial Technology (ICIT)*, pages 986–991. IEEE, 2017.
- [25] Duc Tam Nguyen, Zhongyu Lou, Michael Klar, and Thomas Brox. Anomaly detection with multiple-hypotheses predictions. In *International Conference on Machine Learning*, pages 4800–4809, 2019.
- [26] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
- [27] Yongzhi Qu, Miao He, Jason Deutsch, and David He. Detection of pitting in gears using a deep sparse autoencoder. *Applied Sciences*, 7(5):515, 2017.

- [28] Domen Racki, Dejan Tomazevic, and Danijel Skocaj. A compact convolutional neural network for textured surface anomaly detection. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1331–1339. IEEE, 2018.
- [29] Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Timothy Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, pages 3621–3629, 2016.
- [30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [32] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International Conference on Machine Learning*, pages 4393–4402, 2018.
- [33] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Georg Langs, and Ursula Schmidt-Erfurth. f-anogan: Fast unsupervised anomaly detection with generative adversarial networks. *Medical image analysis*, 54:30–44, 2019.
- [34] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International Conference on Information Processing in Medical Imaging*, pages 146–157. Springer, 2017.
- [35] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.
- [36] Benjamin Staar, Michael Lütjen, and Michael Freitag. Anomaly detection with convolutional neural networks for industrial surface inspection. *Procedia CIRP*, 79:484–489, 2019.
- [37] Xiaohong Sun, Jinan Gu, Rui Huang, Rong Zou, and Benjamin Giron Palomares. Surface defects recognition of wheel hub based on improved faster r-cnn. *Electronics*, 8(5):481, 2019.
- [38] Domen Tabernik, Samo Šela, Jure Skvarč, and Danijel Skočaj. Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing*, pages 1–18, 2019.
- [39] Daniel Stanley Tan, Robert Neil Leong, Ann Franchesca Laguna, Courtney Anne Ngo, Angelyn Lao, Divina Amalin, and Dionisio Alvindia. A framework for measuring infection level on cacao pods. In *2016 IEEE Region 10 Symposium (TENSYMP)*, pages 384–389. IEEE, 2016.
- [40] Daniel Stanley Tan, Robert Neil Leong, Ann Franchesca Laguna, Courtney Anne Ngo, Angelyn Lao, Divina M Amalin, and Dionisio G Alvindia. Autodidac: Automated tool for disease detection and assessment for cacao black pod rot. *Crop Protection*, 103:98–102, 2018.
- [41] Xian Tao, Dapeng Zhang, Wenzhi Ma, Xilong Liu, and De Xu. Automatic metallic surface defect detection and recognition with convolutional neural networks. *Applied Sciences*, 8(9):1575, 2018.
- [42] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.
- [43] Xiangyu Xu, Deqing Sun, Jinshan Pan, Yujin Zhang, Hanspeter Pfister, and Ming-Hsuan Yang. Learning to super-resolve blurry face and text images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 251–260, 2017.
- [44] Yong-Ho Yoo, Ue-Hwan Kim, and Jong-Hwan Kim. Convolutional recurrent reconstructive network for spatiotemporal anomaly detection in solder paste inspection. *arXiv preprint arXiv:1908.08204*, 2019.
- [45] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning*, pages 7354–7363, 2019.
- [46] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.
- [47] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018.

## Supplementary Materials

In this supplementary material, we provide additional details that are necessary for reproducing our results as well as additional results that we could not include in the paper due to the page limit.

### A. Additional Implementation Details

Table 1 and Table 2 shows a detailed architecture of our TrustMAE and the discriminator used to train it.

As a pre-processing step, we resize the images to  $256 \times 256$  and scaled the values to be within  $-1$  to  $1$ . During training, we augment our training data using horizontal flips, vertical flips, and random rotations, except for some products, such as cable, metal nut, and transistors, that have fixed orientations, and will be considered defective if it is transformed.

To encourage sharper reconstructions, our model optimizes for a weighted sum of several loss terms, as shown in Eq. 9.

$$L_{total} = \lambda_{rec} L_{rec} + \lambda_{sm} L_{sm} + \lambda_{vgg} L_{vgg} + \lambda_{GAN} L_{GAN} + \lambda_{feat} L_{feat} + \lambda_{margin} L_{margin} + \lambda_{trust} L_{trust} \quad (9)$$

We set the loss weights as follows  $\lambda_{rec} = 10, \lambda_{sm} = 10, \lambda_{vgg} = 10, \lambda_{GAN} = 1, \lambda_{feat} = 10, \lambda_{margin} = 0.1, \lambda_{trust} = 1$ . Each of the loss terms are detailed below:

- Reconstruction loss encourages the output to be close to the input image. We implement it as the mean absolute error between the input and output, as shown below.

$$L_{rec} = \frac{1}{HWC} \sum_{h,w,c} |x - \hat{x}| \quad (10)$$

- SSIM loss matches the luminance, contrast, and structure between two images by matching the mean, standard deviation, and covariance of their patches, as shown in Eq. 11

$$L_{sm}(p, q) = \frac{(2\mu_p\mu_q + c_1)(2\sigma_{pq} + c_2)}{(\mu_p^2 + \mu_q^2 + c_1)(\sigma_p^2 + \sigma_q^2 + c_2)}, \quad (11)$$

where  $\mu_p, \mu_q$  are means of patch  $p$  and  $q$ ,  $\sigma_p, \sigma_q$  are the standard deviations of the patch  $p$  and  $q$ ,  $\sigma_{pq}$  is the covariance between the two patches, and  $c_1, c_2, c_3$  are constants that prevents numerical issues caused by the divisions. We use a patch size of  $11 \times 11$  and set the constants to their recommended defaults  $c_1 = 0.01^2, c_2 = 0.03^2, c_3 = c_2/2$ .

- VGG feature loss is a form of perceptual loss that encourages feature representations to be similar rather than exact pixel matching [18]. It is defined in Eq. 12

$$L_{vgg} = \sum_{l=1}^L \lambda^{(l)} \|\psi^{(l)}(x) - \psi^{(l)}(G(x))\|_1, \quad (12)$$

where  $\psi^{(l)}(x)$  denotes the output at the  $l$ -th layer of a pre-trained VGG-19 network given an input  $x$ ,  $G(x)$  denotes the reconstructed output of the auto-encoder, and  $\lambda^{(l)}$  are hyperparameters that adjusts the relative importance of each layer  $l$ . The lower layers preserves low level features such as edges, while higher layers preserves high level features such as texture and spatial structure. We used the outputs of conv1\_2, conv2\_2, conv3\_4, conv4\_4, and conv5\_4 layers with layer weights  $\lambda^{(1)} = 1/32, \lambda^{(2)} = 1/16, \lambda^{(3)} = 1/8, \lambda^{(4)} = 1/4, \lambda^{(5)} = 1$ .

- GAN loss [13] introduces a separate discriminator network  $D_{disc}$  that aims to distinguish whether an input image looks real or fake. Our memory auto-encoder  $G$  is jointly optimized with the discriminator  $D_{disc}$  in an adversarial fashion wherein the discriminator classifies the synthesized reconstructions as fake, while the memory auto-encoder tries to produce images that fools the discriminator into classifying it as real. We adopt the hinge loss [45] formulation as defined in Eq. 13 for the discriminator and Eq. 14 for the auto-encoder  $G$ .

$$L_{GAN}^{(Disc)} = -\mathbb{E}[\min(0, -1 + D_{disc}(x))] - \mathbb{E}[\min(0, -1 - D_{disc}(G(x)))] \quad (13)$$

$$L_{GAN}^{(G)} = -\mathbb{E}[D_{disc}(G(x))] \quad (14)$$

- GAN feature loss [42, 43] is similar to the VGG feature loss but uses the intermediate layers of the discriminator instead of a VGG network, as shown in Eq. 15, where  $D_{disc}^{(l)}$  denotes the intermediate output of the discriminator at layer  $l$ . Wang et al. [42] showed that matching the statistics of real images through the discriminator features at multiple scales help stabilize the adversarial training. Another advantage is that the discriminator is trained on the dataset that we care about, which means that the features we are matching are more appropriate for the dataset we are using [43], as opposed to using features extracted by the VGG network that were optimized for ImageNet.

$$L_{feat} = \sum_{l=1}^L \|D_{disc}^{(l)}(x) - D_{disc}^{(l)}(G(x))\|_1 \quad (15)$$

## B. Baselines Implementation Details

We adopted most of the baseline performance values from Bergmann et al. [2], Dehaene et al. [6], Huang et al. [16], and Liu et al. [22].

For the Memory auto-encoder (MemAE) [12] baseline (which our model was built on top of), we trained the model patterned after their publicly available code<sup>1</sup>. We adapted their network architecture and changed the 3D convolutions to 2D. We trained the network on images sized  $256 \times 256$  for 200 epochs using the Adam optimizer with a learning rate of  $1e^{-4}$ . The memory size was set to 128 with a latent dimension of 256.

The baselines AE-SSIM [3] and AE-L2 [2] follows the setting of Dehaene et al. [6]. They used the network architecture of Bergmann et al. [3] with a latent dimension of 100 and trained for 300 epochs with a learning rate of  $1e^{-4}$ . The images were resized to  $128 \times 128$  except for textures, which required larger resolutions due to high frequency content, thus, the texture images were first resized to  $512 \times 512$ , before cropping random patches of  $128 \times 128$  for training.

## C. Additional Results

Table 3 shows the results of our ablation study on each of the classes in MVTec dataset, while Table 4 shows the results of our comparison with the baselines on each of the classes in the MVTec dataset. Overall, our method with both trust region memory updates and spatial perceptual distance achieves competitive performance across a large range of noise levels.

We also show additional qualitative results in Figures 1, 2, and 3. We can observe that the large values in the error map computed with our spatial perceptual distance matches well with the ground truth defect segmentation.

---

<sup>1</sup><https://github.com/donggong1/memae-anomaly-detection>

Table 1: Network architecture of our TrustMAE. The abbreviations are as follows: N denotes number of filters, K denotes kernel size, S denotes stride, P denotes padding, and CBN denotes conditional batch normalization.

| Input → Output Shape  | Layer Information                                      |
|---|--|
| Encoder   |  |
| $(h, w, 3) \rightarrow (h, w, 32)$  | Conv-(N32, K7 × 7, S1, P3), CBN, ReLU                  |
| $(h, w, 32) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$                           | Conv-(N64, K3 × 3, S2, P1), CBN, ReLU                  |
| $(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$      | Conv-(N128, K3 × 3, S2, P1), CBN, ReLU                 |
| $(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$     | Conv-(N256, K3 × 3, S2, P1), CBN, ReLU                 |
| $(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{16}, \frac{w}{16}, 512)$   | Conv-(N512, K3 × 3, S2, P1), CBN, ReLU                 |
| $(\frac{h}{16}, \frac{w}{16}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Conv-(N512, K3 × 3, S2, P1), CBN, ReLU                 |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Residual Block: Conv-(N512, K3 × 3, S1, P1), CBN, ReLU |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Residual Block: Conv-(N512, K3 × 3, S1, P1), CBN, ReLU |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Residual Block: Conv-(N512, K3 × 3, S1, P1), CBN, ReLU |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Memory Module $\in \mathbb{R}^{512 \times M}$          |
| Decoder   |  |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Residual Block: Conv-(N512, K3 × 3, S1, P1), CBN, ReLU |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Residual Block: Conv-(N512, K3 × 3, S1, P1), CBN, ReLU |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$ | Residual Block: Conv-(N512, K3 × 3, S1, P1), CBN, ReLU |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{16}, \frac{w}{16}, 512)$ | ConvTrans-(N512, K4 × 4, S2, P1), CBN, ReLU            |
| $(\frac{h}{16}, \frac{w}{16}, 512) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$   | ConvTrans-(N256, K4 × 4, S2, P1), CBN, ReLU            |
| $(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$     | ConvTrans-(N128, K4 × 4, S2, P1), CBN, ReLU            |
| $(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$      | ConvTrans-(N64, K4 × 4, S2, P1), CBN, ReLU             |
| $(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (h, w, 32)$                           | ConvTrans-(N32, K4 × 4, S2, P1), IN, ReLU              |
| $(h, w, 32) \rightarrow (h, w, 3)$  | Conv-(N3, K7 × 7, S1, P3), Tanh                        |

Table 2: Network architecture of our discriminator.

| Input → Output Shape   | Layer Information                             |
|--|---|
| $(h, w, 3) \rightarrow (\frac{h}{2}, \frac{w}{2}, 32)$                             | Conv-(N32, K4 × 4, S2, P1), LeakyReLU-(0.2)   |
| $(\frac{h}{2}, \frac{w}{2}, 32) \rightarrow (\frac{h}{4}, \frac{w}{4}, 64)$        | Conv-(N64, K4 × 4, S2, P1), LeakyReLU-(0.2)   |
| $(\frac{h}{4}, \frac{w}{4}, 64) \rightarrow (\frac{h}{8}, \frac{w}{8}, 128)$       | Conv-(N128, K4 × 4, S2, P1), LeakyReLU-(0.2)  |
| $(\frac{h}{8}, \frac{w}{8}, 128) \rightarrow (\frac{h}{16}, \frac{w}{16}, 256)$    | Conv-(N256, K4 × 4, S2, P1), LeakyReLU-(0.2)  |
| $(\frac{h}{16}, \frac{w}{16}, 256) \rightarrow (\frac{h}{32}, \frac{w}{32}, 512)$  | Conv-(N512, K4 × 4, S2, P1), LeakyReLU-(0.2)  |
| $(\frac{h}{32}, \frac{w}{32}, 512) \rightarrow (\frac{h}{64}, \frac{w}{64}, 1024)$ | Conv-(N1024, K4 × 4, S2, P1), LeakyReLU-(0.2) |
| $(\frac{h}{64}, \frac{w}{64}, 1024) \rightarrow (\frac{h}{64}, \frac{w}{64}, 1)$   | Conv-(N1, K3 × 3, S1, P1)                     |

Table 3: Detailed ablation study showing the performance of the different components of our model on each of the products in the MVTec dataset.

| Method             | Noise % | mean AUC | bottle | cable | capsule | carpet | grid  | hazelnut | leather | metal nut | pill  | screw | tie   | toothbrush | transistor | wood  | zipper |
|--------------------|---------|----------|--------|-------|---------|--------|-------|----------|---------|-----------|-------|-------|-------|------------|------------|-------|--------|
| ours w/o PD w/o TR | 5%      | 82.61    | 88.28  | 71.17 | 75.16   | 80.82  | 96.83 | 91.04    | 95.57   | 64.96     | 77.29 | 75.13 | 83.73 | 95.56      | 65.25      | 94.23 | 84.17  |
| ours w/o SP        | 5%      | 90.73    | 99.84  | 89.03 | 78.96   | 91.26  | 97.78 | 96.88    | 97.92   | 88.07     | 75.76 | 79.18 | 96.76 | 99.44      | 82.08      | 99.15 | 88.83  |
| ours w/o PD        | 5%      | 82.03    | 96.72  | 70.58 | 73.37   | 80.32  | 91.27 | 93.96    | 94.14   | 59.28     | 82.93 | 82.41 | 76.11 | 94.44      | 53.92      | 94.74 | 86.22  |
| ours w/o TR        | 5%      | 92.51    | 99.13  | 90.94 | 79.04   | 93.54  | 98.41 | 97.22    | 99.48   | 88.16     | 86.99 | 79.25 | 99.44 | 99.44      | 82.42      | 100   | 94.26  |
| ours               | 5%      | 92.39    | 99.37  | 92.77 | 78.96   | 93.47  | 98.1  | 98.82    | 97.27   | 86.55     | 84.77 | 81.73 | 99.01 | 96.67      | 84         | 100   | 94.42  |
| ours w/o PD w/o TR | 10%     | 81.27    | 84.84  | 69.55 | 71.64   | 80.78  | 95.08 | 90.69    | 93.1    | 59.56     | 78.56 | 76.77 | 81.54 | 94.44      | 63.92      | 92.87 | 85.66  |
| ours w/o SP        | 10%     | 90.02    | 98.59  | 88.21 | 77.8    | 92.17  | 96.83 | 97.36    | 95.57   | 87.59     | 76.82 | 77.49 | 95.98 | 98.33      | 79.83      | 99.47 | 88.32  |
| ours w/o PD        | 10%     | 81.14    | 94.53  | 72.19 | 76.55   | 79.26  | 89.37 | 91.94    | 87.76   | 62.03     | 81.09 | 76.33 | 79    | 92.78      | 57.25      | 91.85 | 85.14  |
| ours w/o TR        | 10%     | 91.8     | 99.06  | 89.84 | 79.95   | 91.95  | 98.58 | 97.99    | 98.05   | 88.64     | 85.77 | 77.13 | 97.46 | 97.22      | 81.08      | 99.83 | 94.47  |
| ours               | 10%     | 91.9     | 99.38  | 91.2  | 77.02   | 93.77  | 98.41 | 99.03    | 97.59   | 87.22     | 83.83 | 78.93 | 98.87 | 97.22      | 81.33      | 99.83 | 94.83  |
| ours w/o PD w/o TR | 15%     | 79.83    | 85.31  | 71.86 | 70.11   | 75.46  | 93.65 | 90.76    | 92.51   | 59.09     | 71.03 | 74.69 | 79.35 | 92.22      | 63.83      | 93.72 | 83.81  |
| ours w/o SP        | 15%     | 89.14    | 97.62  | 87.09 | 76.4    | 90.96  | 95.24 | 96.53    | 95.01   | 87.22     | 75.13 | 76.41 | 95.6  | 97.22      | 80.92      | 98.64 | 87.14  |
| ours w/o PD        | 15%     | 80.78    | 97.03  | 69.6  | 73.6    | 75.08  | 84.29 | 91.18    | 90.27   | 63.45     | 77.34 | 81.33 | 79.15 | 95.56      | 58.75      | 91.74 | 83.35  |
| ours w/o TR        | 15%     | 90.38    | 98.89  | 87.12 | 79.1    | 92.10  | 96.67 | 96.67    | 96.88   | 86.27     | 76.98 | 77.45 | 97.32 | 98.33      | 78.08      | 99.15 | 94.72  |
| ours               | 15%     | 91.05    | 99.37  | 89.18 | 73.76   | 91.57  | 98.1  | 99.24    | 97.79   | 87.03     | 82.46 | 75.13 | 97.11 | 98.89      | 82.17      | 100   | 94.01  |
| ours w/o PD w/o TR | 20%     | 78.8     | 85.94  | 67.02 | 67.86   | 74.92  | 94.15 | 88.12    | 91.67   | 53.88     | 69.92 | 72.45 | 82.45 | 92.78      | 64.83      | 92.02 | 84.02  |
| ours w/o SP        | 20%     | 88.94    | 98.28  | 87.45 | 76.75   | 88.45  | 96.03 | 96.32    | 96.55   | 85.13     | 77.87 | 75.05 | 94.93 | 96.67      | 79.17      | 99.32 | 86.12  |
| ours w/o PD        | 20%     | 79.72    | 93.75  | 66.87 | 73.37   | 73.94  | 85.08 | 92.15    | 89.45   | 60.13     | 79.77 | 80.25 | 79.07 | 94.44      | 53.75      | 90.15 | 83.66  |
| ours w/o TR        | 20%     | 88.93    | 97.19  | 84.85 | 77.93   | 91.79  | 94.92 | 96.81    | 96.55   | 80.4      | 78.19 | 72.33 | 94.22 | 98.89      | 77.25      | 99.32 | 93.29  |
| ours               | 20%     | 90.32    | 99.69  | 89.8  | 74.15   | 92.17  | 98.25 | 98.54    | 98.57   | 84.66     | 82.09 | 75.85 | 92.67 | 97.22      | 79.5       | 98.64 | 92.93  |
| ours w/o PD w/o TR | 30%     | 79.18    | 87.34  | 65.19 | 70.57   | 77.05  | 89.72 | 91.88    | 93.88   | 52.56     | 66.97 | 75.89 | 85.13 | 90.56      | 63.08      | 94.23 | 83.66  |
| ours w/o SP        | 30%     | 87.01    | 96.41  | 86.76 | 75.18   | 89.51  | 92.38 | 95.56    | 92.06   | 81.25     | 75.18 | 74.69 | 95.28 | 91.11      | 73.75      | 99.15 | 86.89  |
| ours w/o PD        | 30%     | 79.8     | 90.31  | 67.72 | 67.78   | 77.89  | 70.32 | 93.89    | 94.86   | 65.81     | 82.09 | 82.21 | 82.8  | 92.78      | 56.25      | 90.49 | 81.81  |
| ours w/o TR        | 30%     | 87.1     | 97.03  | 83.2  | 74.91   | 88.68  | 89.21 | 96.04    | 94.66   | 78.41     | 75.08 | 71.41 | 93.23 | 96.11      | 77.42      | 98.47 | 92.67  |
| ours               | 30%     | 89.89    | 97.66  | 89.1  | 75.08   | 91.49  | 95.56 | 97.92    | 96.81   | 83.52     | 81.61 | 76.37 | 93.02 | 97.78      | 78.33      | 100   | 94.06  |
| ours w/o PD w/o TR | 40%     | 76.96    | 83.81  | 66.47 | 70.42   | 72.72  | 83.71 | 89.93    | 89.71   | 52.65     | 53.64 | 71.57 | 81.61 | 95         | 61.58      | 95.25 | 86.27  |
| ours w/o SP        | 40%     | 84.96    | 94.22  | 83.93 | 72.28   | 85.87  | 83.49 | 95.14    | 94.79   | 78.79     | 74.97 | 74.53 | 93.73 | 86.67      | 71.58      | 98.3  | 86.12  |
| ours w/o PD        | 40%     | 77.82    | 91.88  | 69.59 | 72.13   | 64.74  | 70.16 | 93.06    | 90.49   | 59.94     | 79.72 | 78.49 | 85.41 | 91.11      | 56.17      | 81.32 | 83.04  |
| ours w/o TR        | 40%     | 86.68    | 96.41  | 81.95 | 75.22   | 89.44  | 84.96 | 96.46    | 96.48   | 79.07     | 72.34 | 71.69 | 92.6  | 97.22      | 76.92      | 97.45 | 92.01  |
| ours               | 40%     | 89.87    | 99.34  | 88.26 | 76.01   | 91.11  | 93.49 | 98.54    | 96.35   | 84.28     | 79.77 | 75.97 | 94.01 | 97.78      | 79.33      | 99.83 | 93.95  |

Table 4: Detailed performance on varying noise levels of our model compared with the baselines on each of the products in the MVTec dataset.

| Method  | Noise % | mean AUC | bottle | cable | capsule | carpet | grid  | hazelnut | leather | metal nut | pill  | screw | tile  | toothbrush | transistor | wood  | zipper |
|---------|---------|----------|--------|-------|---------|--------|-------|----------|---------|-----------|-------|-------|-------|------------|------------|-------|--------|
| Ours    | 5%      | 92.39    | 99.37  | 92.77 | 78.96   | 93.47  | 98.1  | 98.82    | 97.27   | 86.55     | 84.77 | 81.73 | 99.01 | 96.67      | 84         | 100   | 94.42  |
| MemAE   | 5%      | 79.68    | 86.93  | 49.63 | 78.42   | 75.76  | 96.98 | 95.62    | 94.27   | 44.31     | 77.33 | 81.95 | 77.68 | 95.37      | 68.58      | 96.49 | 75.85  |
| AE-SSIM | 5%      | 76.52    | 87.81  | 80.74 | 71.58   | 61.93  | 78.78 | 78.82    | 57.68   | 62.31     | 83.25 | 76.01 | 69.06 | 91.67      | 78.17      | 90.35 | 79.71  |
| AE-MSE  | 5%      | 81.81    | 92.66  | 76.85 | 73.37   | 71.31  | 90.98 | 96.25    | 86.98   | 60.51     | 77.08 | 76.68 | 78.44 | 93.89      | 68.25      | 95.08 | 88.78  |
| Ours    | 10%     | 91.9     | 99.38  | 91.2  | 77.02   | 93.77  | 98.41 | 99.03    | 97.59   | 87.22     | 83.83 | 78.93 | 98.87 | 97.22      | 81.33      | 99.83 | 94.83  |
| MemAE   | 10%     | 77.26    | 87.34  | 49.66 | 72.77   | 71.73  | 94.28 | 93.8     | 90.45   | 43.65     | 74.08 | 74.12 | 73.29 | 95         | 64.33      | 95.98 | 78.38  |
| AE-SSIM | 10%     | 75.52    | 87.62  | 76.45 | 69.95   | 60.07  | 75.19 | 80.14    | 57.29   | 60.51     | 82.35 | 77.82 | 69.7  | 90.56      | 77.83      | 87.46 | 79.92  |
| AE-MSE  | 10%     | 80.09    | 92.34  | 74.5  | 71.89   | 73.88  | 88.73 | 96.74    | 84.82   | 53.88     | 76.1  | 76.77 | 72.55 | 91.11      | 68.75      | 93.89 | 85.4   |
| Ours    | 15%     | 91.05    | 99.37  | 89.18 | 73.76   | 91.57  | 98.1  | 99.24    | 97.79   | 87.03     | 82.46 | 75.13 | 97.11 | 98.89      | 82.17      | 100   | 94.01  |
| MemAE   | 15%     | 77.89    | 90.78  | 47.54 | 77.1    | 72.44  | 95.39 | 91.06    | 97.74   | 36.74     | 74.97 | 77.97 | 73.17 | 93.33      | 63.03      | 96.32 | 80.77  |
| AE-SSIM | 15%     | 74.32    | 84.6   | 75.06 | 69.18   | 58.15  | 73.49 | 78.33    | 56.52   | 61.46     | 81.51 | 75.13 | 69.48 | 91.11      | 76.58      | 86.23 | 77.92  |
| AE-MSE  | 15%     | 79.25    | 91.72  | 76.05 | 69.64   | 68.02  | 89.21 | 96.11    | 85.36   | 56.34     | 73.55 | 74.25 | 73.45 | 92.22      | 69.33      | 91.85 | 81.61  |
| Ours    | 20%     | 90.32    | 99.69  | 89.8  | 74.15   | 92.17  | 98.25 | 98.54    | 98.57   | 84.66     | 82.09 | 75.85 | 92.67 | 97.22      | 79.5       | 98.64 | 92.93  |
| MemAE   | 20%     | 76.05    | 88.33  | 48.12 | 76.06   | 74.82  | 93.01 | 93.77    | 91.3    | 33.21     | 73.04 | 74.13 | 70.61 | 93.89      | 55.11      | 96.15 | 79.15  |
| AE-SSIM | 20%     | 72.74    | 85.95  | 72.96 | 67.31   | 57.14  | 70.48 | 76.88    | 53.45   | 61.17     | 82.67 | 73.25 | 64.39 | 86.67      | 73.67      | 86.67 | 78.38  |
| AE-MSE  | 20%     | 78.37    | 90.16  | 75.5  | 69.25   | 69.3   | 88.3  | 95.56    | 84.57   | 52.56     | 73.4  | 71.47 | 71.93 | 88.89      | 67.75      | 93.89 | 82.99  |
| Ours    | 30%     | 89.89    | 97.66  | 89.1  | 75.08   | 91.49  | 95.56 | 97.92    | 96.81   | 83.52     | 81.61 | 76.37 | 93.02 | 97.78      | 78.33      | 100   | 94.06  |
| MemAE   | 30%     | 74.06    | 85.6   | 48.86 | 76.45   | 63.8   | 91.26 | 91       | 94.29   | 35.92     | 68.12 | 74.22 | 58.87 | 90.93      | 59.31      | 95.81 | 76.4   |
| AE-SSIM | 30%     | 71.88    | 83.44  | 73.66 | 67.69   | 58.79  | 67.62 | 77.36    | 52.8    | 59.56     | 82.46 | 74.41 | 61.5  | 85.56      | 72.5       | 84.55 | 76.28  |
| AE-MSE  | 30%     | 76.33    | 90.31  | 71.72 | 67.39   | 68.86  | 87.46 | 93.86    | 82.13   | 49.81     | 72.55 | 69.77 | 65.37 | 88.33      | 62.67      | 91.68 | 83.04  |
| Ours    | 40%     | 89.87    | 99.34  | 88.26 | 76.01   | 91.11  | 93.49 | 98.54    | 96.35   | 84.28     | 79.77 | 75.97 | 94.01 | 97.78      | 79.33      | 99.83 | 93.95  |
| MemAE   | 40%     | 74.62    | 81.51  | 47.35 | 77.07   | 68.57  | 88.73 | 90.37    | 91.62   | 40.56     | 65.1  | 78.56 | 61.36 | 90.37      | 62.14      | 96.49 | 79.53  |
| AE-SSIM | 40%     | 69.8     | 80.94  | 69.74 | 66.54   | 54.94  | 66.92 | 74.31    | 50.98   | 57.1      | 81.09 | 70.28 | 60.64 | 87.22      | 71.42      | 82.17 | 72.66  |
| AE-MSE  | 40%     | 75.46    | 88.91  | 71.2  | 66.23   | 67.38  | 84.29 | 90.61    | 80.74   | 49.24     | 73.13 | 67.97 | 66.17 | 89.44      | 62.58      | 90.15 | 83.81  |

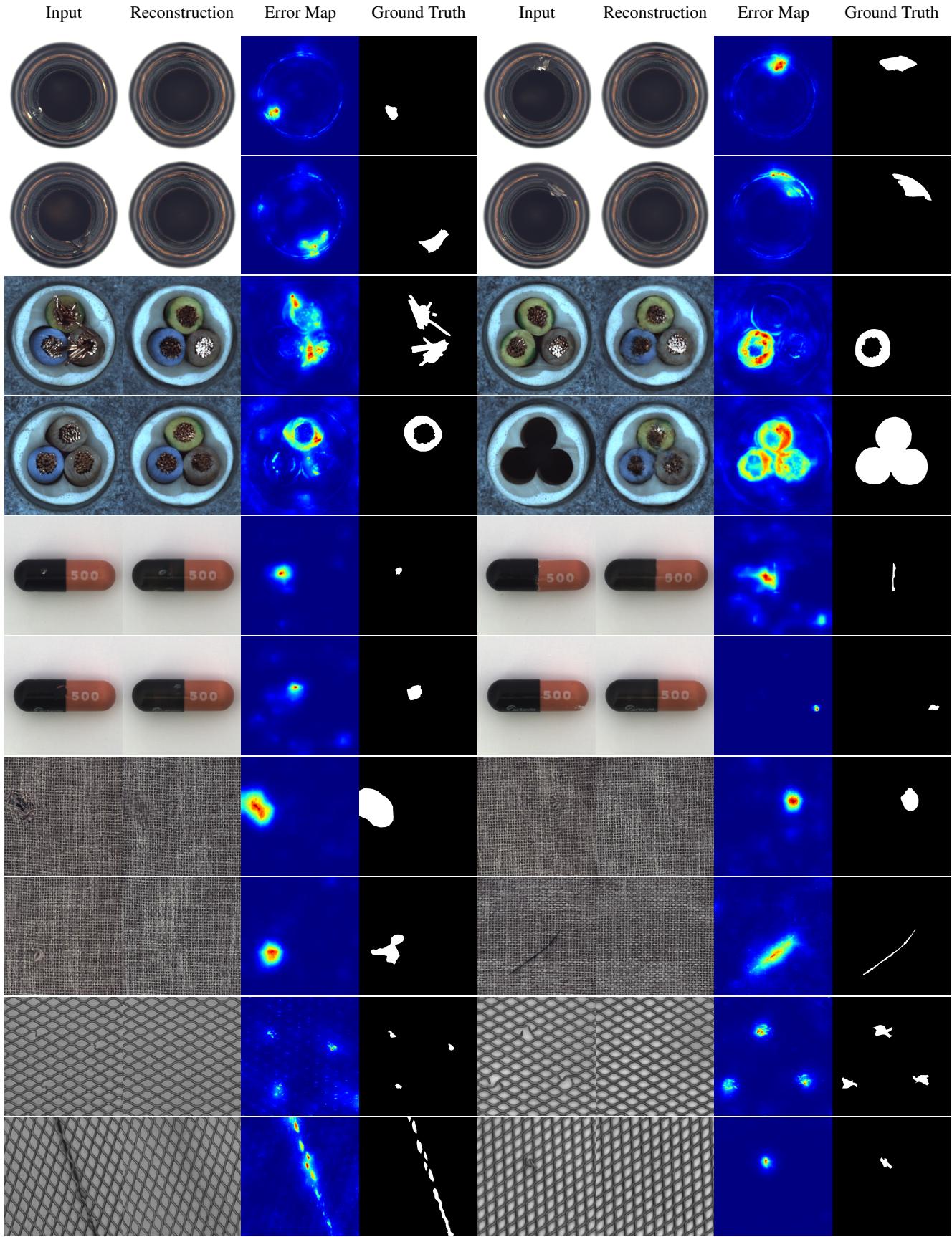


Figure 1: More visual results of our proposed TrustMAE.

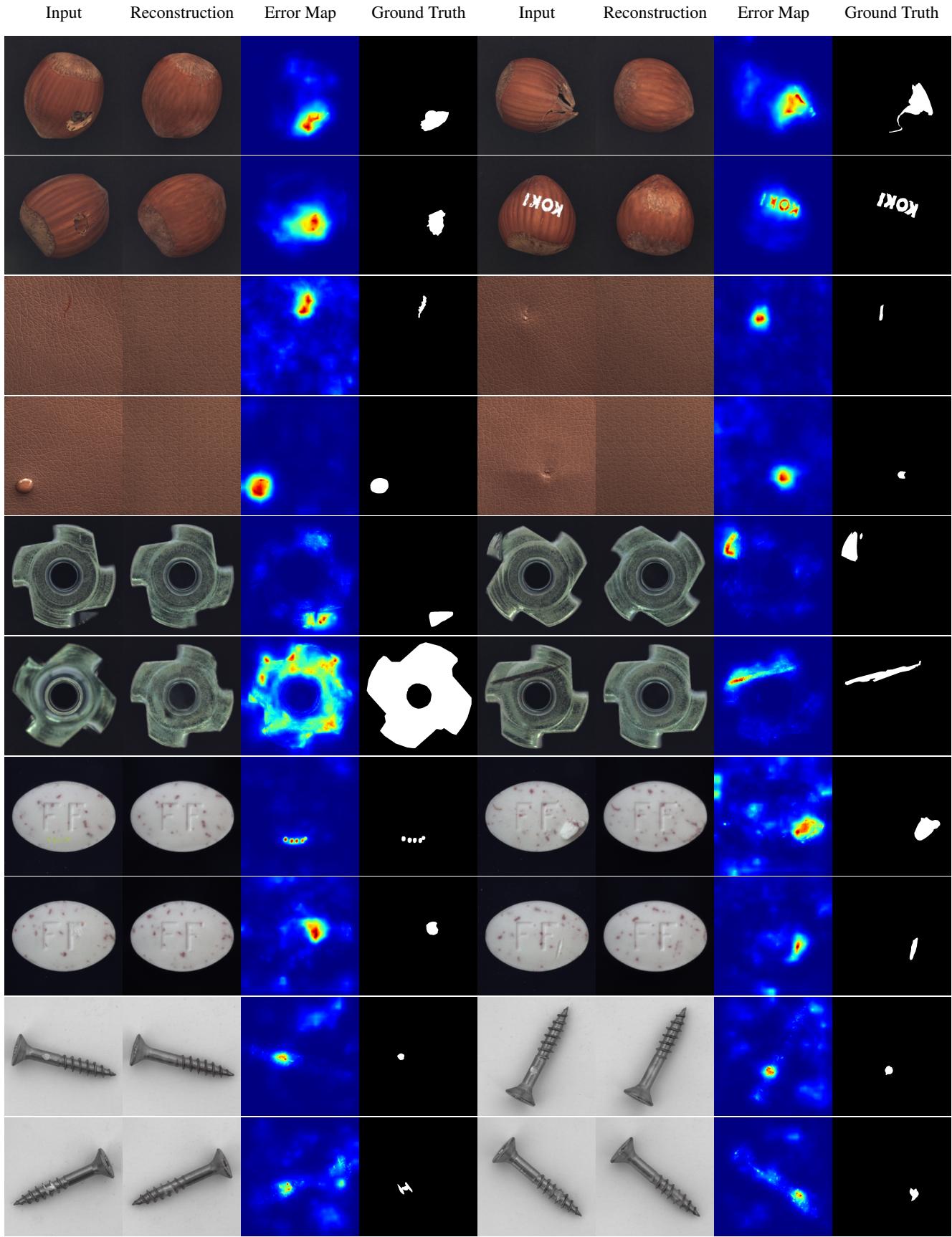


Figure 2: More visual results of our proposed TrustMAE.

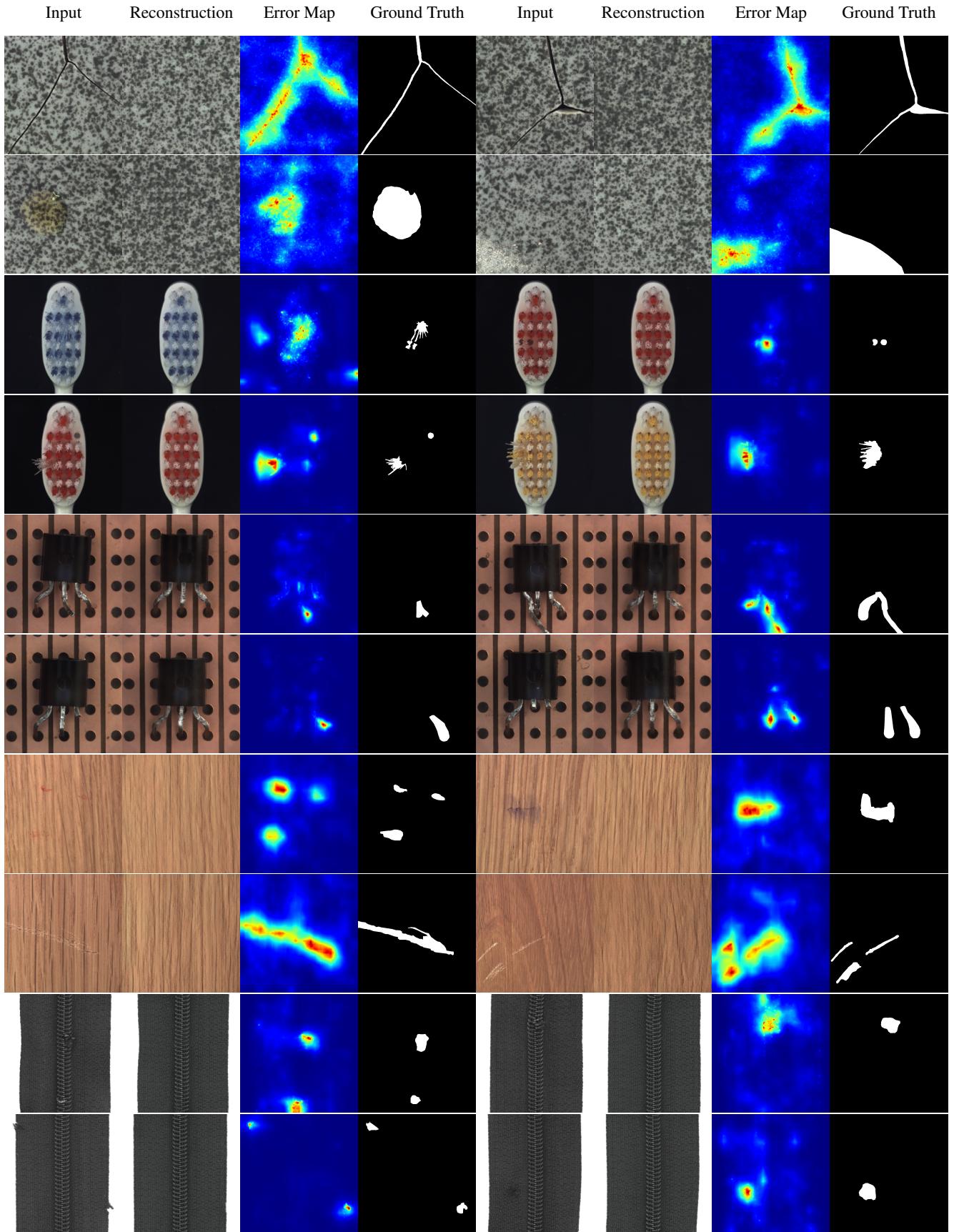


Figure 3: More visual results of our proposed TrustMAE.