SAKARYA UNIVERSITY JOURNAL OF COMPUTER AND INFORMATION SCIENCES
VOL. 5, NO. 3, DECEMBER 2022
DOI: 10.35377/saucis.05.03.1196381

Research Article

Pseudo-Supervised Defect Detection Using Robust Deep Convolutional Autoencoders

Mahmut Nedim Alpdemir
TÜBİTAK, Informatics and Information Security Research Center (BİLGEM); nedim.alpdemir@tubitak.gov.tr;

Received 29 October 2022; Accepted 22 November 2022; Published online 31 December 2022

Abstract

Robust Autoencoders separate the input image into a *Signal(L)* and a *Noise(S)* part which, intuitively speaking, roughly corresponds to a more stable background scene (L) and an undesired anomaly (or defect) (S). This property of the method provides a convenient theoretical basis for divorcing intermittent anomalies that happen to clutter a relatively consistent background image. In this paper, we illustrate the use of Robust Deep Convolutional Autoencoders (RDCAE) for defect detection, via a pseudo-supervised training process. Our method introduces synthetic simulated defects (or structured noise) to the training process, that alleviates the scarcity of true (real-life) anomalous samples. As such, we offer a pseudo-supervised training process to devise a well-defined mechanism for deciding that the *defect-normal discrimination* capability of the autoencoders has reached to an acceptable point at training time. The experiment results illustrate that pseudo supervised Robust Deep Convolutional Autoencoders are very effective in identifying surface defects in an efficient way, compared to state of the art anomaly detection methods.

Keywords: robust autoencoders, anomaly detection, defect detection, machine learning, convolutional neural networks

1. Introduction

Detecting data samples with deviating features compared to a set of examples deemed as "normals", constitutes a major research area. Research communities with varying precedence and focus employ different names for this problem, such as outlier detection, novelty detection, anomaly detection, defect detection, noise detection, deviation detection or exception mining. Despite the variation in naming, the fundamental problem is to define a region in the feature space that represents "the normal" for a data set, and subsequently identify all cases that lie outside the boundaries of that region. The application areas of the problem include fraud detection, structural defect detection, intrusion detection, time-series monitoring, loan application processing, medical condition monitoring, motion segmentation, detecting novelty in text etc. [1]. The solution space to the problem has been explored by different communities resulting in a partitioning along several axes, reflecting approaches, methodologies and tools adopted by those communities. Many surveys on the subject probe into different methods, techniques and approaches employed to solve the problem [1, 2, 3]. As more recent developments in *Deep Learning* [4] offer promising results in extracting relevant features in an automated way, in particular for computer vision applications [5], more recent surveys such as [6] and [7] provide a deep learning centric account of the subject.

Anomaly detection problem is relatively difficult to solve in general. Therefore, most of the techniques in the literature tend to solve a specific instance of the general problem based on the type of application, the type of input data and model, the availability of labels for the training data, and the type of anomalies. The problem of defect detection in flat surfaces, where scarcity of defect samples is a common issue, is a good example of a research domain that stands to benefit from an anomaly detection perspective. Scarcity of abnormal data is something that promotes the use of anomaly detection as a candidate solution, since anomaly detection methods rely only on normal (as apposed to abnormal or anomalous) samples at training phase. In this domain, it is also important to increase the accuracy, recall and precision of the detection process to ensure the applicability of the method for complex real life

problems in industrial settings. So methods that avoid dependency on abundant defect samples whilst improving the detection performance are of particular interest.
Recently, a specific form of neural networks, namely an *Autoencoder (AE)* [8], has attracted the attention of researchers from different domains, due to its ability to learn an efficient, compressed representation of its input via its encoder part and reconstruct this input via its decoder part (see for instance [9] and [10]). The primary motivation for those researchers has been to exploit this capability to learn the features characterizing the normal samples and later use the error generated by the difference between the input and the reconstructed output to identify the anomalous samples. AEs, come in different flavors, such as Convolutional AE (CAE), Variational AE (VAE) and Robust AE (RAE) to name a few, each introducing an additional capability on top the central ability mentioned above. A variant of AEs called Robust Convolutional Autoencoder(RCAE) [11, 12] appears to be a particularly promising solution for two reasons:

- First, Convolutional Autoencoders(CAE), in general, combine the good aspects of AEs and Convolutional Neural Networks (CNN). As stated above, an AE is known for its talent to learn a low level compressed representation of a normal class via minimization of the reconstruction error through its encoding and decoding layers. A CNN on the other hand preserves the spatial locality [13] of important features and this is very important for 2D images since defects are locally positioned in an image.
- Second, *Robust* AEs, in particular, separate the input image into a *Signal(L)* and a *Noise(S)* part which, intuitively speaking, roughly corresponds to a more stable background scene (L) and an undesired anomaly (or defect) (S) for image based applications. This property of the method provides a convenient theoretical basis for divorcing intermittent anomalies that happen to clutter a relatively consistent background image. There is a large family of automated visual quality inspection applications that can benefit from such anomaly detection capability.

As described by [11], use of the *Robust AE* by feeding normal and anomalous samples during training and letting the RAE differentiate the anomalies as noise at the end of an iterative process, is relatively straightforward. However, this requires the availability of abnormal samples at training time and does not accommodate subsequent (post-training) anomalous instance identification. Training the RAE using only normal samples to specify a threshold based on a learned reconstruction score of the normal samples and later use that threshold to detect unseen anomalies is what we are interested in. This process is named as *inductive anomaly detection* by [12] and can be challenging when RAE struggles to learn the distribution of the normal data to a sufficient degree that it can differentiate out of distribution (OOD) samples correctly. This may be due to two problems:

1. the normal samples used for training may not be sufficiently representative (both in terms of quality and number), or
2. the training architecture and process may not be vigorous enough to cater for signal-noise separation in the absence of some representative abnormal samples (i.e. qualitative noise).

In this paper, we primarily target the improvement of the latter of the problems mentioned above. Our overall contribution is twofold:

1. We illustrate the viability of *Robust Deep Convolutional AEs* as an efficient solution for surface defect detection utilizing two relatively recent industrial quality datasets.
2. We propose a method to further enhance this solution, by introducing *synthetic simulated defects (or structured noise)* to the training process in a novel way, so that a safe discriminating threshold can be determined by relying on a more robust convergence criteria during training iterations, in the absence of true (real life) abnormal samples. As such, we offer a specific form of *pseudo-supervised* training process as a well defined mechanism to alleviate the dependency of RAEs to true abnormal samples.

The rest of the paper is organized as follows: Section 2 provides some theoretical background and related work, Section 3 introduces the details of our methodology, Section 4 provides information on

the structure of the experiments, Section 5 presents results and discussions and finally Section 6 provides some concluding remarks.

2. Background and Related Work

The problem of defect detection in textured surfaces is a good example of a research domain that stands to benefit from an anomaly detection perspective and therefore it is selected as the target case study for our method. Anomaly detection context, re-casts the defect identification problem into the problem of recognizing the *divergence from the normal* with reference to distinct features characterizing the normal. As such, relatively regular and uniform patterns that characterize a non-defected (normal) surface, provide a convenient referential basis for learning-based anomaly detection approaches. Traditionally, automated defect detection literature tend to categorize the solution methods into several groups including structural, statistical, spectral, model-based, learning-based and hybrid methods. [14, 15] and [16], for instance, provide a comprehensive account of these methods including a comparative study of their detection and classification performance. The most important aspect of the classical methods is that they apply a processing pipeline to an image containing the surface to be inspected, that starts with low level image processing techniques such as filtering, transformation, distribution identification etc., continues with feature extraction and defect detection, and finally terminates with defect classification. More recent surveys such as [17] focusing on textile domain and [18] offering a more diverse view of industrial surface inspection applications, place greater emphasis on learning-based approaches and in particular on deep-learning. Both acknowledge that deep learning methods simplify the aforementioned processing pipeline since they automate feature extraction to a greater extent, but also note that they may require abundant and quality data samples for effective training of the classifiers. This latest trend in fabric defect detection research towards learning-based methods, suggests that the techniques and methods that are mostly developed by the machine learning community, are well placed to capitalize on.

In that respect, there are several recent techniques and methods in the literature that are notably promising for anomaly detection cases that suffer from scarcity of abnormal samples. These are AEs [8], One-Class Support Vector Machines (OC-SVM) [19], Isolation Forests (IF) [20, 21], One-Class Support Vector Data Description (OC-SVDD) [22, 23] and One-Class Neural Networks (OC-NN) [24]. Earlier anomaly detection methods such as the One-Class SVM (OC-SVM) or Kernel Density Estimation (KDE) [25] are known to rely on tractable feature spaces with moderate dimensions and are prone to failure in cases involving large scale, complex data manipulation due to curse of dimensionality. More novel neural network based solutions such as Deep Convolutional Autoencoders(DCAE) [26, 27] and Deep One-Class Neural Networks [28, 29] have been on the agenda of contemporary research efforts, introducing some improvements that alleviate the deficiencies of the earlier methods.

As stated in the introduction, Robust Convolutional Autoencoders(RCAE), in particular, exhibit distinctly useful behaviors since they combine the ability to learn a highly efficient, locality preserving and non-linear representation of their input, with the ability to progressively learn to separate signal (normal surface) from the noise (defects). These convenient properties form the rationale for our adoption of the RCAE as a viable solution. From a methodological point of view, the most relevant work in the literature to the work presented in this paper is that of [11]. The authors augment an AE with a filter layer that culls out the anomalous parts of the input data, X, that are difficult to reconstruct. They then propose that the remaining portion of the data, S can be represented by the low-dimensional hidden layer, L_D, with small reconstruction error. The problem of finding anomalies is than cast into the following optimization problem:

$$\underset{\theta,S}{\text{Min}} = \| L_D - D_\theta \left(E_\theta \left(L_D\right)\right) \|_2 + \lambda \| S \|_{2,1}; \;\; s.t. X - L_D - S = 0 \tag{1}$$

Here the input data X is split into two parts, L_D and S. L_D is the input to an AE $D_\theta \left(E_\theta \left(L_D\right)\right)$ and the AE is trained by minimizing the reconstruction error $\| L_D - D_\theta \left(E_\theta \left(L_D\right)\right) \|_2$ through back-

propagation. S, on the other hand, contains outliers which are difficult to represent using the AE. We use this general framework for the formulation of the problem at hand but there are some differences between their approach and ours, and also some improvements provided by our work that deserve mentioning:

- They provide two distinct regularization methods one targeting denoising and the other targeting anomaly detection. We only aim at anomaly detection (or more specifically defect detection), so only $l_{2,1}$ regularization is applied (as opposed l_1 regularization which is used for denoising).
- They require true anomalous samples (i.e. real samples labeled as defects) for tuning the hyperparameter controlling noise-signal separation (i.e. λ in the equation above). As such, part of the training has to be done in a semi-supervised manner. On the contrary, we perform the same hyper parameter tuning using synthetic defects leading to a pseudo-supervised training procedure. This divorces the training process from the dependency on labeled, true anomalous samples.
- Their convergence control logic relies on the use of real defected samples and depend on the result of two conditional inquiry: 1 - check if $X - (L_D + S) < \epsilon$ and 2 - check if L_D and S have converged to a fixed point (i.e. there is no a significant change any more). Whereas, we use synthetic defects in our training process and check the Area Under ROC Curve (AUC) score obtained by testing the AE performance in separating the (synthetic) defected samples from the normal samples. When the training ends we also obtain an outlier-threshold based on the reconstruction error that characterizes the normal samples (i.e. during actual testing, instances that produce a reconstruction error below that threshold are identified as normal).

Before delving into the details of our methodology in the next section, we proceed by providing some background information on DCAE and RAE in the following subsections.

2.1 Deep Convolutional Autoencoders (DCAE)

An AE is known for its ability to compress its input into an efficient feature representation via its encoding part and then reconstruct it via its decoding part. In the middle of the two parts lies its bottleneck layer (also known as latent space) where the input is encoded into an efficient, much lower dimensional feature space. The encoder and decoder parts can be defined as transitions E and D , such that:

$$E: \mathcal{X} \to \mathcal{F}$$
$$D: \mathcal{F} \to \mathcal{X}$$
$$\mathbf{x}' = \| \mathbf{x} - D\left(E\left(\mathbf{x}\right)\right) \| \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^d = \mathcal{X}$ refers to an input in the $\mathcal{X}$ domain and $\mathbf{x}'$ denotes the reconstructed input. The hidden bottleneck layer, then, can be represented by $\mathbf{z} = E(\mathbf{x}) \in \mathbb{R}^p = \mathcal{F}$ in the $\mathcal{F}$ domain. In the most popular form of AEs, D and E are neural networks. In the special case that D and E are linear operations, we get a linear AE, where we would achieve the same latent representation as Principal Component Analysis (PCA) [30]. Therefore, an AE is in fact a generalization of PCA, where instead of finding a low dimensional hyperplane in which the data lies, it is able to learn a non-linear manifold [8]. In particular, an AE can be viewed as a solution to the following optimization problem:

$$\min_{D,E} = \| \mathbf{x} - D\left(E\left(\mathbf{x}\right)\right) \| \tag{3}$$

Where $\|.\|$ is usually the l_2 norm. CAEs differ from conventional AEs in that their architecture contains an encoder part with convolutional and pooling layers, and an analogous decoder part with deconvolutional and upsampling layers. As such, recalling that $\mathbf{x}'$ denotes the reconstruction, the encoder and decoder processes can be expanded as:

$$\mathbf{z} = E\left(W \circ \mathbf{x} + b\right)$$
$$\mathbf{x}' = D\left(W' \circ \mathbf{z} + b'\right) \tag{4}$$

where "$\circ$" is the convolution process; W and W' are the weight matrices; $\boldsymbol{b}$ and $\boldsymbol{b}'$ are the bias vectors for the encoder and decoder, respectively; and E and D are the nonlinear mapping processes,

specifically, the convolutional, pooling, deconvolutional, and upsampling processes. Particularly, the pooling and upsampling processes are usually conducted in the form of max pooling and max unpooling.

2.2 Robust Autoencoders (RAE)

Robust AEs are built on a theoretical basis borrowed from Robust Principle Component Analysis (RPCA) [31, 32]. Specifically, RPCA splits a data matrix X into a low-rank matrix L and a sparse matrix S such that;

$$X = L + S \tag{5}$$

where the matrix L contains a low-dimensional representation of X and the matrix S consists of element-wise outliers, which can not be efficiently captured by the low-dimensional representation L. Similar to RPCA, a Robust Deep Autoencoder also splits input data X into two parts;

$$X = L_D + S \tag{6}$$

where L_D represents the part of the input data that is well represented by the hidden layer of the AE, and S contains noise and outliers which are difficult to reconstruct. So the idea is that, just as in RPCA, by iteratively separating out the noise and outliers from X into S, the remaining data L_D can be accurately reconstructed by an AE. As such, RAE combines non-linear representation capabilities of AEs with the anomaly detection capabilities of RPCA. The peculiar behavior of the AE that is conveniently exploited in reconstruction-based anomaly detection in a general context is that noise and outliers are essentially difficult to compress and therefore cannot effectively be projected to a low-dimensional hidden layer. So, if those outliers could be incorporated into the AE loss function in an appropriate way, then the low-dimensional hidden layer could provide accurate reconstruction, except for those few outliers [11].

3. Methodology

Our method introduces synthetic simulated defects (or structured noise) to the training process, so that a safe discriminating threshold can be determined by relying on a more robust convergence criteria during training iterations, in the absence of true (real life) abnormal samples. As such, we offer a pseudo-supervised training process to devise a well-defined mechanism for deciding that the defect-normal discrimination capability of the AE has reached to an acceptable point at training time.

Using noisy inputs in the training of the robust AEs has been adopted by other researchers. For instance in [33] white Gaussian random noise is used to simulate anomalous samples so that the AE can learn the distribution generating normal samples more efficiently. The main difference of our method is to use a more complex model (i.e. structured noise) for anomalies. Another example is the use of random noise in denoising AEs [34]. A denoising AE is a stochastic extension to classic AE where the AE is forced to learn the reconstruction of input given its noisy version, usually using a stochastic corruption process to randomly set some of the inputs to zero. It is important to note that, the use of structured noise in our case is categorically different from this type of noise utilization. In contrast to denoising AEs, we force the AE to separate common, stable features from the anomalous ones. As it will be elaborated on in Section 5.1, our findings show that, for such cases, the incorporation of structured noise (or synthetic defects) produces better results compared to injecting random noise into some of the normal samples. A similar approach, in terms of incorporating structured synthetic noise at training time is used in more recent works (see for instance [35] and [36]). Of these two work, [35] use an autoencoder equipped with skip connections and train it to reconstruct a clean image out of a synthetically corrupted version of it. To corrupt the training images they introduce a synthetic model, named Stain, that adds an irregular elliptic structure of variable color and size to the input image. Their main motivation for using synthetic anomalies is to alleviate a vulnerability of their skipped architecture that causes the network to perform identity mapping on uncorrupted clean images. In the work reported by [36], on the other hand, synthetic anomalies are generated using a sort of "confetti noise", a simple noise model that inserts colored blobs into images and reflects the local nature of anomalies. Since their approach is semi-supervised, essentially they utilize the synthetic defects to model the incorporation of few known anomalies into the training process (in the absence of true anomalies), an approach which they report to be effective. Compared to our method, there are some differences, however, in the way both of these

approaches generate the synthetic defects as well as the exact purpose of and the final benefits obtained from using those synthetic defects:

1. Our defect generation model is based on a more complex, Julia set-based algorithm that facilitates modeling of pseudo-topological patterns that are well suited to structured fabric surfaces. The merits of modeling the defects in a structured way, compared to simple statistical noise, has been explored in this paper. However exploring the impact of modeling structured synthetic defects relatively more faithfully compared to other structured modeling approaches would require a thorough experimentation and analysis using different types of datasets, something we do not intend to endeavor in this particular paper. Intuitively speaking, our approach may stand to benefit from increased defect modeling fidelity for certain datasets by causing AE to learn normal/abnormal separation earlier and more effectively. A more extensive comparative analysis remains as an interesting future work.
2. The use of synthetic defects in our work is an integral part of the optimization process, both in terms of constituting an important ingredient of the noise component S in Equation 6 and also being an important facilitator for deciding on the outlier threshold during the training phase. Determining a threshold at training time with good discriminating power for testing phase is an important success factor for reconstruction based approaches and synthetic defects in our method play a critical role in managing this process.

Note that, in our case, synthetic defects are not used to train a classifier for defect classification, so extensive defect modeling is not needed here. Structured noise is used only to better exploit the signal-noise separation capability of the robust auto encoder for efficient and effective *defect detection*, not to ensure accurate *classification* of different defect types. In the following three subsections, the formulation of the optimization problem, the algorithm used to generate synthetic defects and the optimization algorithm are explained in more detail.

3.1 Formulation of the Optimization Problem

The optimization problem is formulated along similar lines to the one given by [11] for anomaly detection. Specifically;

$$\begin{aligned} &\underset{\theta,S}{\text{Min}} = \| L_D - D_\theta\left(E_\theta\left(L_D\right)\right) \|_2 + \lambda \| S^T \|_{2,1} \\ &s.t. X - L_D - S = 0 \end{aligned} \tag{7}$$

where $E_\theta(\cdot)$ denotes an encoder, $D_\theta(\cdot)$ denotes a decoder, and S captures the anomalous data, L_D is a low dimension manifold and λ is a parameter that tunes the level of sparsity in S. Here the loss function for a given input X could be thought of as the 'grouped $l_{2,1}$ norm of S, balanced against the reconstruction error of L_D. The $l_{2,1}$ norm of any X is defined as:

$$\| X \|_{2,1} = \sum_{j=1}^{n} \| x_j \|_2 = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} |x_{ij}|^2\right)^{\frac{1}{2}} \tag{8}$$

and, it can be viewed as inducing a 2 norm regularizer over members of each group and then a 1 norm regularizer between groups. In equation 7, λ plays an essential role in the defect and background separation. In particular, a small λ will encourage much of the data to be isolated into S as noise or outliers, and therefore minimize the reconstruction error for the AE. Similarly, a large λ will discourage data from being isolated into S as noise or outliers, and therefore increase the reconstruction error for the AE.

Note that here the $l_{2,1}$ norm minimization problem can be implemented efficiently as a proximal problem as defined by [37] and adopted by [11], where the proximal operator is a block-wise soft-thresholding function.

3.2 Synthetic Defect Modeling

Algorithm 1 Pseudo Code for Julia Set Defect Generator

```
Procedure GenerateDefect (α, β, Δx, Δy, cx, cy, zf, w, h)
   R = escape radius                    //such that R > 0, R² − R > √(cx² + cy²)
   For each pixel (x, y) on the image of size (w, h)
      zx = α(x − w/2) / (0.5 * zf * w) + Δx
      zy = β(y − h/2) / (0.5 * zf * h) + Δy
      max_iter = 255
      i = max_iter
      While zx² + zy² < R² and i>max_iter
         zy = 2zx * zy + cy
         zx = zx² + zy² + cx
         i = i - 1
      End While
      image[x,y] = i
   End For
End Procedure
```

As mentioned earlier we are not using the synthetic defects for high fidelity defect simulation, so decoration of the normal fabric surface with defect-like structures are deemed sufficient. To this aim, we employ Julia set fractals [38]. Note that, other fractal pattern generation mechanisms can potentially be used for this purpose. However, Julia sets have a feature known as centro-symmetry [39] that facilitates modeling of some pseudo-topological patterns. As such, with proper parameter tuning it is possible to create certain Julia Set patterns that mimic some common defects in textured fabric surfaces, such as yarn tails, thick bars, holes, stains etc. Julia set fractals can be obtained by using a complex number $z = x + yi$ where $i^2 = -1$ and x and y are image pixel coordinates. The fractal is generated by repeatedly updating z using the formula $z = z^2 + c$, where c is another complex number that gives a specific Julia set. After numerous iterations, if the magnitude of z is less than a certain escape radius we say that pixel is in the Julia set and color it to generate desired patterns. Performing this calculation for a whole grid of pixels gives a fractal image.

We employ a parametric algorithm to generate different defect patterns. Algorithm *1* given above is controlled by nine parameters, where α and β determine the extent (i.e. length) and alignment of the defect (i.e. horizontally extended, vertically extended, point-like), $\Delta x, \Delta y$ are used as position offsets with respect to the center of the image; cx, cy are coefficients that determine the shape of the fractal pattern, zf is the zoom factor to determine coverage area of the defect, and finally w, h are width and height of the image to be generated.

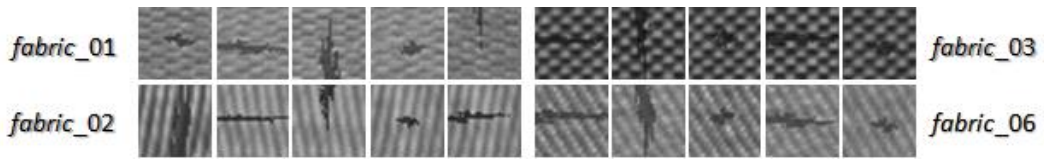


Figure 1 Examples of synthetic defects generated by decorating normal fabric samples. Various different types of defects (i.e. horizontal, vertical, point-like etc.) are shown using four different fabric type.

Figure 1 illustrates examples generated using four different fabric types in our dataset. The $32x32$ image patches are first extracted from non-defect fabric images and then they are decorated using the above algorithm. The original non-defect image patch and the $32x32$ matrix returned by the defect generator algorithm are blended by averaging the weighted pixel values to obtain a smoother overlay.

3.3 Optimization Algorithm

Algorithm 2 The pseudo code for our optimization procedure

```
1  Procedure Optimize (α, β, Δx, Δy, cx, cy, zf, w, h)
2     Input X                                  //X is a set of input images
3     L_D = 0                                  //init L_D to 0(same size as X)
4     while upper-limit not reached
5       L_D = X − S                            //Remove S from X, use L_D to train the AE
6       trainer.train(D(E(.)), L_D, epochnum)  //Minimize recons. error using ADAM
7       L_D = D(E(L_D))                        //Set L_D to reconstruction from trained AE
8       S = X − L_D                            //Set S to be the difference between X and L_D
9       prox_{λ,l_{2,1}}(S^T)                  //Optimize S^T using a proximal operator
10      trainer.test(D(E(.)), X)               //test X for defect/non-defect separation
11      If AUC > 0.999 then
12         Break                               //convergence criteria met, so break
13      End If
14    End While
15    Return L_D, S and outlier_treshold
16 End Procedure
```

Our optimization procedure (given in Algorithm 2) starts with reading the input X and initializing S and L_D to zero. X contains all the images used for training (i.e normal images and images with synthetic defects). Note that synthetic defects and normal samples are explicitly labeled (-1 and 0 respectively). X is a 4 dimensional tensor, so if the training set size is t, dimensions of X would be (t, w, h, c), where w and h are width and height of the image in pixels; and c is the number of color channels of the images. The optimization loop starts by first setting L_D to $X - S$. Thus, in the first iteration L_D becomes equal to X since S is zero. Then the AE ($D(E(.))$) is trained with the objective of minimizing the reconstruction error via the ADAM [40] optimization algorithm. The next step is to use the trained AE to get the reconstructed L_D set and assigning it onto itself, and later to set S to be the difference between X and L_D. So the purpose of lines 3.3 and 3.3 is to capture features that are easy to reconstruct in L_D and isolate the difference of the reconstructed and the original (which are supposed to be the features that are hard to reconstruct, and hence the noisy parts) in S. Then the second part of our optimization formulation given in Equation 7 (i.e. $\lambda \parallel S^T \parallel_{2,1}$) is handled using a proximal operator [37, 11]. Finally we test for defect/ non-defect separation by checking the Area Under ROC Curve (AUC) score. If AUC is greater than 99.9%, then the algorithm returns L_D, S and an outlier_treshold to be used for further defect identification. The outlier_treshold is obtained by finding the highest 99.9% quantile of the set containing the reconstruction errors of non-defect (normal) samples. [11] use a different convergence criteria in their method using actual abnormal samples. They calculate the sum of L_D and S to see if the sum is close to the input X, and also they check if L_D and S have converged to a fixed point (i.e. there is no a significant change any more). Our AUC score calculation serves a similar purpose, but prioritizes the learning level of the AE. In both approaches it is necessary to put an upper limit on the iterations to avoid cases leading to futile convergence.

4. Experiments

4.1 The Experimental Setup

The architecture of our AE follows the style of a LeNet type CNN [41], where each convolutional module consists of a convolutional layer followed by leaky ReLU activation and 2×2 max-pooling. Our CNN contains three modules, $32 \times (5 \times 5 \times 3)$-filters, $64 \times (5 \times 5 \times 3)$-filters, and $128 \times (5 \times 5 \times 3)$-filters, followed by a final dense layer of 256 units. This final dense layer implements the compressed latent representation. The batch size used in the experiments was 128 and weight decay hyper-parameter was set to $\gamma = 10^{-6}$. The workstation used had an i5 CPU with 6 cores and an NVIDIA GTX 1060 graphics card with 6GB RAM and 1280 GPU cores. We used PyTorch [42] as our main

machine learning framework. All of the neural networks and associated optimizers of our method are implemented in PyTorch with CUDA option enabled, to exploit the multi core capability of our GPU. We also utilized the popular python machine learning library scikit-learn [43] for generating the results of other methods we endorsed for comparison (see Section 5.3).

4.2 The Datasets Used

We used two main datasets for the experiments: 1 - The AITEX Fabric Dataset is a recently introduced industrial quality image dataset targeted for fabric defect detection [44]. The dataset consists of 245 images of 4096×256 pixels captured by the system of seven different fabric structures. The fabrics in the dataset are mainly plain, which is very convenient for illustrating the utility of our method, yet covers a reasonable range of fabric types. There are 140 defect free images in the database, sampled from 20 different types of fabric. The remaining 105 images are defected, containing 12 different types of fabric defects which commonly appear in the textile industry. 2 - The Kolektor Surface Defect Dataset 2 (KSDD2) is yet another recently introduced dataset that is constructed from images of defected production items that were provided and annotated by Kolektor Group d.o.o. [45]. The images were captured in a controlled industrial environment. The dataset consists of 356 images with visible defects and 2979 images without any defect, with image sizes of approximately 230 x 630 pixels. The defected images contain several different types of defects (scratches, minor spots, surface imperfections, etc.). Both of the datasets contain a number of *mask images* each of which corresponds to a unique image containing one or more defects. The mask image is a black-and-white image depicting the exact pixel wise location of a defect, in an unambiguous way. This enabled us to generate labels for defected and defect-free patches of arbitrary sizes, in an automated and deterministic way.

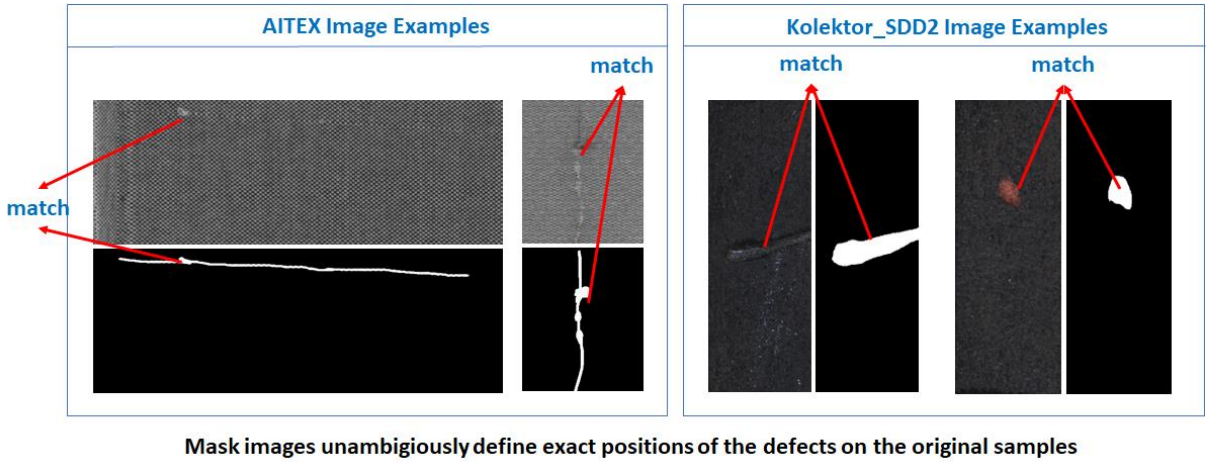


Figure 2 Examples of original samples and mask images from AITEX and KSDD2 datasets. Note that mask images pinpoint the position of defects

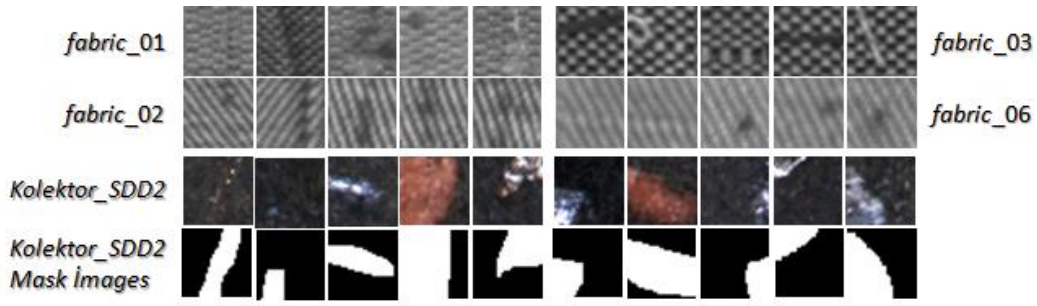


Figure 3 Examples of actual defect samples of size 32x32 pixel each, generated using both defect images and corresponding mask images provided by the AITEX and KSDD2 datasets.

Thus, we implemented a custom sample generator class that can be configured to generate defect free and defected patches of different sizes (e.g. 32×32, 64×64 pixels etc) using the original images contained in both of the datasets. The custom generator is also able to decorate the samples with synthetic structural defects or random noise (e.g. Gaussian, Poisson etc.) with a pre-defined ratio of the total training set. We assume that in a typical industrial setting an image scan camera would send image frames to a processing computer at a certain rate, and that the processing unit would generate small patches (32×32 pixel in our case) to be fed into the defect detector to facilitate easier localization of defects. So an input image to our neural network is a $32x \times 32 \times 3$ matrix (3 at the end is for RGB color channels). Figure 2 illustrates the original and the mask image examples from both datasets, and Figure 3 shows 32×32 patches generated from images containing defects, and some corresponding mask image patches.

4.3 Performance Evaluation Method

Table 1 AUC, F1 Score and Accuracy results for different lambda values.

Dataset	λ	Noise Ratio	Noise Type	RDCAE AUC	F1 Score	Accuracy
$fabric\_00$	1.0	3	$struct.$	0.993 ± 0.005	0.208 ± 0.006	0.135 ± 0.032
	1.55	3	$struct.$	1.000 ± 0.000	0.994 ± 0.009	0.999 ± 0.002
	1.6	3	$struct.$	1.000 ± 0.000	0.998 ± 0.004	1.000 ± 0.001
	1.65	3	$struct.$	1.000 ± 0.000	0.991 ± 0.008	0.998 ± 0.002
	2.5	3	$struct.$	0.992 ± 0.011	0.936 ± 0.069	0.987 ± 0.013
$fabric\_01$	1.0	3	$struct.$	0.999 ± 0.001	0.715 ± 0.028	0.576 ± 0.054
	1.55	3	$struct.$	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	1.6	3	$struct.$	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	1.65	3	$struct.$	1.000 ± 0.000	1.000 ± 0.001	1.000 ± 0.001
	2.5	3	$struct.$	1.000 ± 0.000	0.990 ± 0.006	0.990 ± 0.006
$fabric\_02$	1.0	3	$struct.$	1.000 ± 0.000	0.918 ± 0.112	0.940 ± 0.090
	1.55	3	$struct.$	1.000 ± 0.000	0.999 ± 0.002	1.000 ± 0.001
	1.6	3	$struct.$	1.000 ± 0.000	0.999 ± 0.002	1.000 ± 0.001
	1.65	3	$struct.$	1.000 ± 0.000	0.999 ± 0.002	0.999 ± 0.001
	2.5	3	$struct.$	1.000 ± 0.000	0.995 ± 0.004	0.997 ± 0.002
$fabric\_03$	1.0	3	$struct.$	1.000 ± 0.000	0.865 ± 0.011	0.795 ± 0.019
	1.55	3	$struct.$	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	1.6	3	$struct.$	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	1.65	3	$struct.$	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	2.5	3	$struct.$	1.000 ± 0.001	0.987 ± 0.011	0.983 ± 0.014
$fabric\_04$	1.0	3	$struct.$	0.995 ± 0.004	0.186 ± 0.022	0.230 ± 0.106
	1.55	3	$struct.$	1.000 ± 0.000	0.994 ± 0.007	0.999 ± 0.001
	1.6	3	$struct.$	1.000 ± 0.000	0.997 ± 0.006	0.999 ± 0.001
	1.65	3	$struct.$	1.000 ± 0.000	0.992 ± 0.017	0.998 ± 0.003
	2.5	3	$struct.$	0.998 ± 0.002	0.934 ± 0.109	0.991 ± 0.015
$fabric\_06$	1.0	3	$struct.$	0.929 ± 0.041	0.260 ± 0.167	0.950 ± 0.035
	1.55	3	$struct.$	0.951 ± 0.040	0.323 ± 0.249	0.977 ± 0.007
	1.6	3	$struct.$	0.951 ± 0.040	0.287 ± 0.191	0.976 ± 0.005
	1.65	3	$struct.$	0.951 ± 0.040	0.287 ± 0.191	0.976 ± 0.005
	2.5	3	$struct.$	0.953 ± 0.041	0.210 ± 0.128	0.974 ± 0.003
$KSDD2$	1.0	3	$struct.$	0.863 ± 0.047	0.903 ± 0.003	0.824 ± 0.004
	1.55	3	$struct.$	0.939 ± 0.041	0.903 ± 0.056	0.856 ± 0.078
	1.6	3	$struct.$	0.976 ± 0.012	0.960 ± 0.015	0.936 ± 0.024
	1.65	3	$struct.$	0.938 ± 0.038	0.903 ± 0.053	0.855 ± 0.074
	2.5	3	$struct.$	0.838 ± 0.033	0.759 ± 0.032	0.674 ± 0.034

We conducted the experiments using 1600 defect-free patches. 70% of these defect-free patches are used for training and 30% are left for testing. The number of defect samples for testing depends on the

number of defects available in each of the dataset. For instance, some fabric types in the AITEX dataset include a wider range of defect types and so a higher number of defects. For each dataset type we generated synthetic defects by a pre-determined percentage of normal samples using Julia set fractals as described in Section 3.2. The metrics used for performance evaluation are Area Under ROC Curve (AUC), F1 Score and Accuracy. We resort to these metrics selectively depending on their utility. For instance to compare the performance of our method with respect to the use of different noise types (i.e. random or structured), or to assess the effect of λ parameter we employ all of the metrics above. Whereas to compare our method to other state of the art methods we use AUC only.

5. Results and Discussions

5.1 Tuning the Lambda parameter and comparison of noise types

As indicated in Section 3.1 the value of the λ parameter is critical to ensure that the signal and noise (normal background and defect) separation is optimal. This requires many experiments for tuning. We experimented extensively to search for an optimal value. Although dataset type seem to have an effect on the optimal value to a certain degree, $\lambda = 1.6$ offers a reasonable compromise for the whole dataset range. This is illustrated in Table 1. So in subsequent runs we set $\lambda = 1.6$.

Table 2 AUC, F1 Score and Accuracy results for different image patch sizes.

Dataset	λ	patch size	RCAE AUC	F1 Score	Accuracy
$fabric\_00$	1.6	32×32	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.918 \pm 0.050}$	$\mathbf{0.979 \pm 0.014}$
	6.7	64×64	0.132 ± 0.012	0.169 ± 0.000	0.092 ± 0.000
$fabric\_01$	1.6	32×32	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	6.7	64×64	0.955 ± 0.012	0.931 ± 0.016	0.934 ± 0.015
$fabric\_02$	1.6	32×32	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	6.7	64×64	0.999 ± 0.001	0.933 ± 0.008	0.950 ± 0.007
$fabric\_03$	1.6	32×32	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	6.7	64×64	0.491 ± 0.012	0.800 ± 0.000	0.667 ± 0.000
$fabric\_04$	1.6	32×32	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.993 \pm 0.007}$	$\mathbf{0.999 \pm 0.001}$
	6.7	64×64	0.929 ± 0.000	0.854 ± 0.004	0.965 ± 0.000
$fabric\_06$	1.6	32×32	$\mathbf{0.993 \pm 0.003}$	$\mathbf{0.955 \pm 0.045}$	$\mathbf{0.997 \pm 0.003}$
	6.7	64×64	0.510 ± 0.006	0.192 ± 0.058	0.921 ± 0.003
$KSDD2$	1.6	32×32	$\mathbf{0.976 \pm 0.012}$	$\mathbf{0.960 \pm 0.015}$	$\mathbf{0.936 \pm 0.024}$
	3.3	64×64	0.814 ± 0.023	0.919 ± 0.008	0.855 ± 0.014

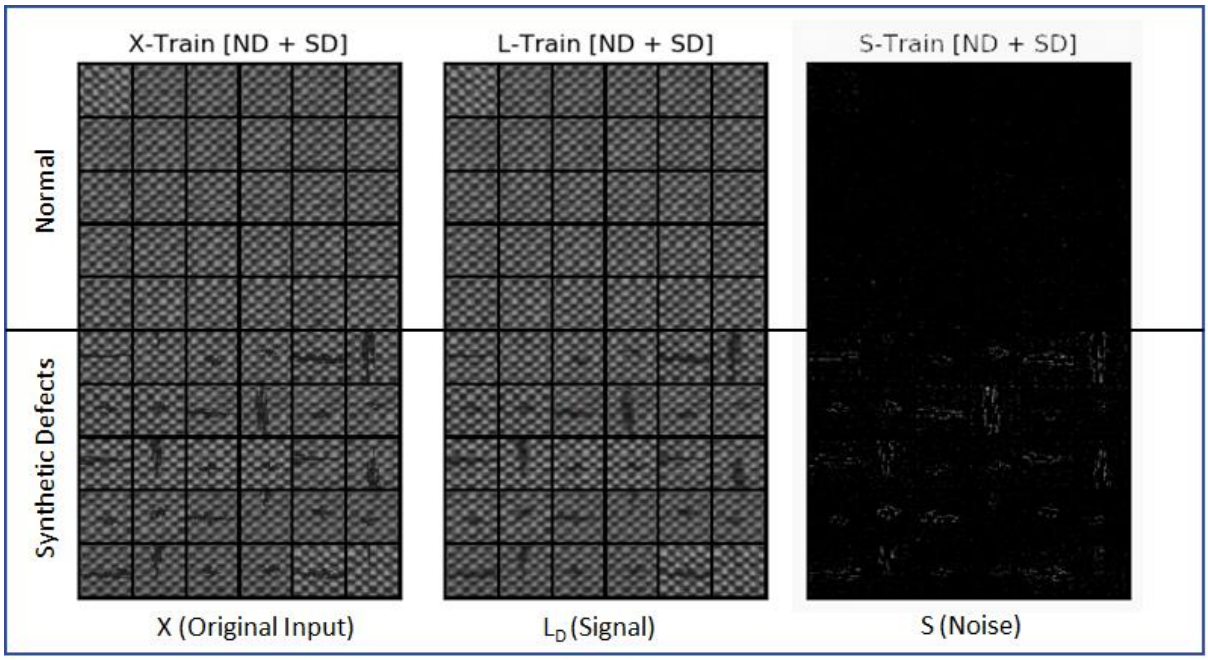
Another important observation during our experiments was the effect of the image patch size (i.e. the size of the input image given to the autoencoder in pixels) on the detection performance of the autoencoder. As indicated in Table 2, smaller patch size leads to much better performance. In fact, smaller patch sizes are also better for more precise defect localization especially for small defects. One drawback of small patch sizes, however, may emerge when the defect (on the actual surface being inspected) is much larger compared to the patch size, which would entail a certain level of additional processing for higher level interpretation and classification of the defect. We also investigated the effect of using synthetic defects as opposed to random noise for pseudo supervised training. The results are illustrated in Table 3. Both in Table 2 and in Table 3 *noise ratio* refers to the percentage of the noisy samples with respect to the total number of samples in the training dataset. Note that using different defect types (i.e. structured or random noise) leads to different behavior of the AE during the optimization loop, which in turn causes our conversion logic to result in varying iteration numbers for the training. Depending on the fabric type, this may have dramatic effects on the performance of RAE during the test phase. To ensure fair comparison of the two types of defect modeling, we fixed the number of training loops to a total of 630 (including regularization iterations as well as AE training epochs) for this particular exercise. Notice that, by observing accuracy and F1 scores in addition to AUC scores, it can be concluded that for all dataset types synthetic defects perform better, and that for some fabric types (e.g. fabric_00, fabric_04, fabric_06 and KSSD2) the improvement induced by using

synthetic structured defects as opposed to random noise is more significant. In all experiments the convergence criteria used to terminate the iterations is the same, as explained in Section 3.1.

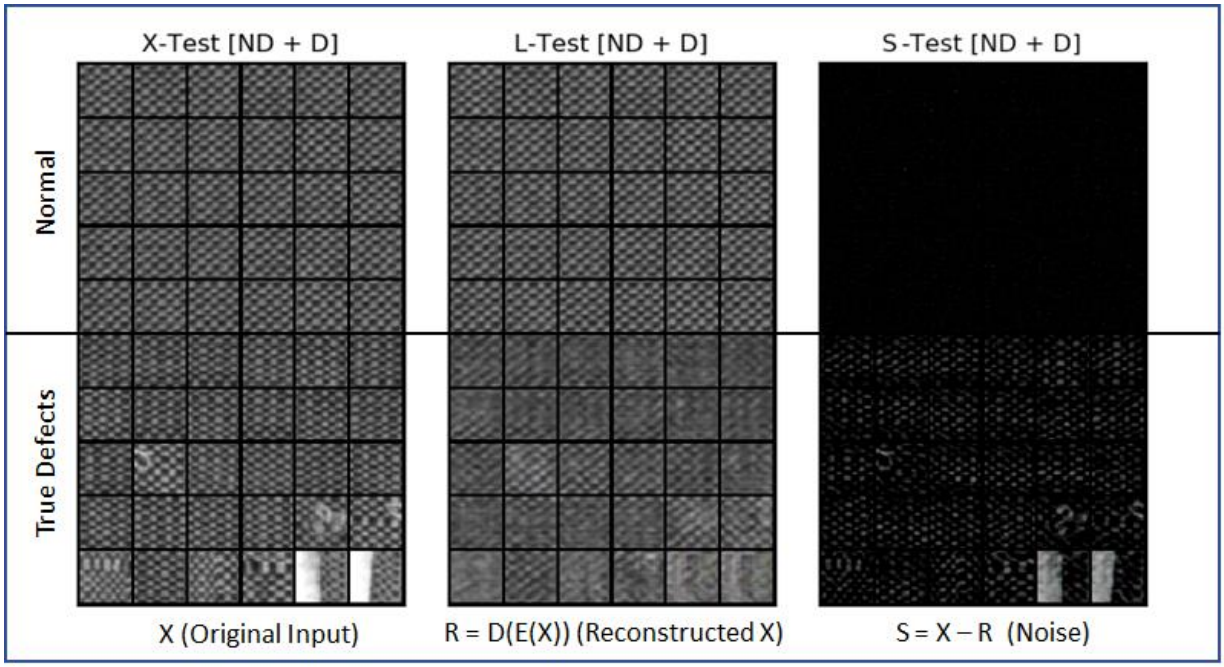
Table 3 Comparison of the Results When Using synthetic Defects (Structured Noise) vs. Random (Gaussian) Noise.

Dataset	Noise Type	Noise Ratio	RDCAE RAUC	F1 score	Accuracy
$fabric\_00$	struct.	5	$\mathbf{0.991 \pm 0.009}$	$\mathbf{0.701 \pm 0.278}$	$\mathbf{0.846 \pm 0.149}$
	random	5	0.618 ± 0.137	0.228 ± 0.024	0.228 ± 0.114
$fabric\_01$	struct.	5	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.996 \pm 0.004}$	$\mathbf{0.995 \pm 0.005}$
	random	5	0.999 ± 0.000	0.963 ± 0.025	0.959 ± 0.028
$fabric\_02$	struct.	5	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
	random	5	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
$fabric\_03$	struct.	5	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.998 \pm 0.002}$	$\mathbf{0.997 \pm 0.003}$
	random	5	0.996 ± 0.002	0.883 ± 0.034	0.863 ± 0.036
$fabric\_04$	struct.	5	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.763 \pm 0.223}$	$\mathbf{0.925 \pm 0.072}$
	random	5	0.976 ± 0.004	0.333 ± 0.004	0.659 ± 0.001
$fabric\_06$	struct.	5	$\mathbf{0.999 \pm 0.000}$	$\mathbf{0.840 \pm 0.017}$	$\mathbf{0.992 \pm 0.000}$
	random	5	0.983 ± 0.005	0.414 ± 0.014	0.977 ± 0.001
$KSDD2$	struct.	5	$\mathbf{0.953 \pm 0.032}$	$\mathbf{0.924 \pm 0.047}$	$\mathbf{0.885 \pm 0.067}$
	random	5	0.943 ± 0.006	0.813 ± 0.008	0.739 ± 0.010

5.2 Illustration of the Outputs



(a) Training samples



(b) Test samples

Figure 4 An example set of L (signal) and S (noise) matrices of 32x32 patches corresponding to fabric_03, illustrating samples extracted from both training (a) and test (b) sets.

We further exemplify the performance of the RDCAE, in Figure 4, by illustrating the visual outputs of our method for signal (background) and noise (defect) samples of one of the datasets (i.e. fabric_03). Figure 4 (a) depicts the original synthetic defects, the L (signal) part and S (noise) part corresponding to the same patches at the end of the training process. Note that the L matrices preserve the overall background pattern except where the defects were located. In contrast, the S matrices capture the defect areas. The figure includes both normal and defect images to highlight the difference. Observe that S images corresponding to the normal samples do not contain any pattern or pixels, indicating that there was no defect to separate out. Figure 4 (b) illustrates the same matrices obtained after the training of the AE, this time using the *test samples* that include the actual *real world* defects. This time instead of L_D we have the reconstructed input (i.e. $R = D(E(X))$) in the middle of the Figure. Notice how the background (R) of the defect images is distorted, and the defects are reflected in the S images.

Figure 5 on the other hand presents the final reconstruction error scores of our RDCAE plotted against a Structural SIMilarity (SSIM) index [46] calculated by comparing the input and output of the AE. The Structural SIMilarity (SSIM) index is a method for measuring the similarity between two images. In contrast to RDCAE reconstruction error scores, with SSIM we expect to have high scores (close to 1) for normal images, and relatively low scores for defect images. For the SSIM algorithm implementation we used the compare_ssim function provided by the scikit-image [47] library. Note that the normal samples of both training and test datasets are clustered closely in a relatively dense area and that the defect samples included in test datasets (i.e. actual real-world defects) are distributed away from the non-defect samples. Also, it can be seen that the synthetic defects are noticeably separated away from non-defect samples of both training and test datasets. This indicates that the RDCAE has learned an efficient representation of the normal samples exceptionally well, such that the RDCAE reconstruction error now behaves as a sound discriminator.

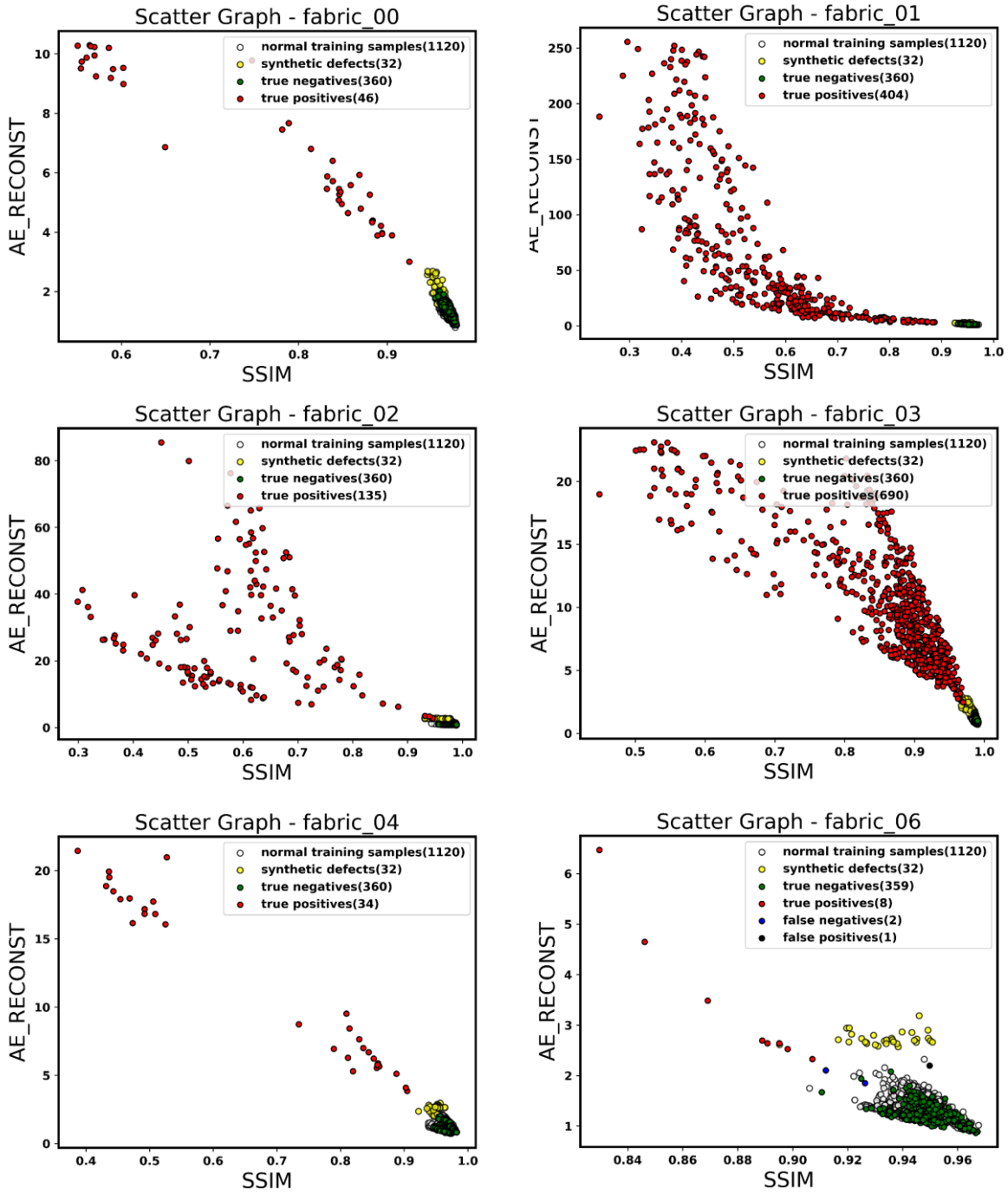


Figure 5 Scatter graphs illustrating a 2D distribution of RDCAE reconstruction errors plotted against structural similarity scores, corresponding to anomalous samples and normal samples of test datasets. Normal samples and synthetic defects used during training

Each graph includes a legend indicating the number of normal training samples, synthetic defects, true positives, true negatives, false positives (if any) and false negatives (if any). The results show that our RDCAE has a remarkable performance for detecting defects in regard with fabric types provided in an industrial grade dataset. Note that throughout the performance tests reported up to this point, the dataset $fabric\_06$ performs relatively poorly. This is likely to be due to its challenging features with regard to two aspects:

1. The normal samples are relatively difficult to differentiate from the anomalous ones since they share more common features compared to other datasets. We note that this could be regarded as an illustrative example of challenges that may arise in real world industrial applications.

2. The number of anomalous samples is much less compared to other datasets (i.e. only 10 samples, compared to 46 for $fabric\_00$, 404 for $fabric\_01$, 135 for $fabric\_02$, 690 for $fabric\_03$ and 34 for $fabri\_04$). Therefore, metric scores (F1 in particular) are much more sensitive to the number of false positives and false negatives.

Note also that, there are some other elements that lead to exceptionally good results for other datasets. First, each of our datasets contains samples from a specific fabric type. So we perform training for a particular fabric type, and then carry out defect detection test for the same fabric type. This implies that in a real world industrial application we would need to train the visual inspection system for each fabric type. This should not be an important predicament, since training times are not huge (in the order of a couple of minutes) for even modest hardware specification we used, and using embedded software would dramatically improve the performance. Moreover, large-volume fabric production lines can tolerate such initial setup phases. However, this is something that should be recognized when interpreting the results. Second, combining AE optimization with pixel wise regularization in a single framework to achieve signal/noise separation and feature learning in a progressive way, seem to work particularly well for fabric defect detection.

5.3 Comparison to Other Anomaly Detection Methods

Table 4 Comparison of Area Under ROC Curve (AUC) Metric Results of Robust Deep Convolutional Autoencoder (RDCAE) to That of Other Methods

Dataset	Robust DCAE	DCAE	Deep SVDD	OC-SVM-LREP	OC-SVM-AERE	ISOF-LREP	ISOF-AERE
$fabric\_00$	$\mathbf{1.000 \pm 0.000}$	0.973 ± 0.026	0.944 ± 0.030	0.676 ± 0.134	0.828 ± 0.028	0.669 ± 0.148	0.825 ± 0.042
$fabric\_01$	$\mathbf{1.000 \pm 0.000}$	0.994 ± 0.002	0.842 ± 0.111	0.762 ± 0.015	0.911 ± 0.011	0.781 ± 0.027	0.902 ± 0.017
$fabric\_02$	$\mathbf{1.000 \pm 0.000}$	1.000 ± 0.000	0.982 ± 0.012	0.665 ± 0.028	0.857 ± 0.009	0.664 ± 0.021	0.886 ± 0.008
$fabric\_03$	$\mathbf{1.000 \pm 0.000}$	1.000 ± 0.001	0.818 ± 0.092	0.476 ± 0.057	0.875 ± 0.007	0.505 ± 0.031	0.873 ± 0.011
$fabric\_04$	$\mathbf{1.000 \pm 0.000}$	0.978 ± 0.006	0.774 ± 0.109	0.488 ± 0.065	0.865 ± 0.019	0.485 ± 0.024	0.857 ± 0.019
$fabric\_06$	$\mathbf{0.993 \pm 0.003}$	0.955 ± 0.019	0.739 ± 0.127	0.384 ± 0.008	0.737 ± 0.063	0.468 ± 0.003	0.777 ± 0.074
$KSSD2$	$\mathbf{0.972 \pm 0.012}$	0.787 ± 0.013	0.807 ± 0.002	0.704 ± 0.010	0.712 ± 0.003	0.661 ± 0.017	0.722 ± 0.003

We compare the performance of our method with four different state of the art methods:

1. **Deep Convolutional Autoencoders (DCAE) [26, 27]**. We employed exactly the same neural network architecture as our Robust DCAE except that the optimization method is based only on minimizing the reconstruction error as in standard AEs.
2. **Isolation Forests (IF) [20].** We used the latest stable version of the widely-used python machine learning library scikit-learn [43]. The IsolationForest function provided by the library is an implementation of the algorithm presented in [20].
3. **One-Class SVM [19].** We employ the implementation provided by scikit-learn [43] as we did for the Isolation Forests. This is an implementation of the algorithm presented in [19]. Once again, we used the latent representation of the DCAE above as input to the algorithm.
4. **One-Class Support Vector Data Description (OC-SVDD).** We adapt the method provided by [23]. We used the PyTorch implementation available from their github repository at https://github.com/lukasruff/Deep-SVDD-PyTorch. To facilitate a meaningful comparison, we ensured that our RDCAE architecture and the AE architecture used in their method are exactly the same, and also used the encoder part in the corresponding one-class SVDD network as suggested by their method.

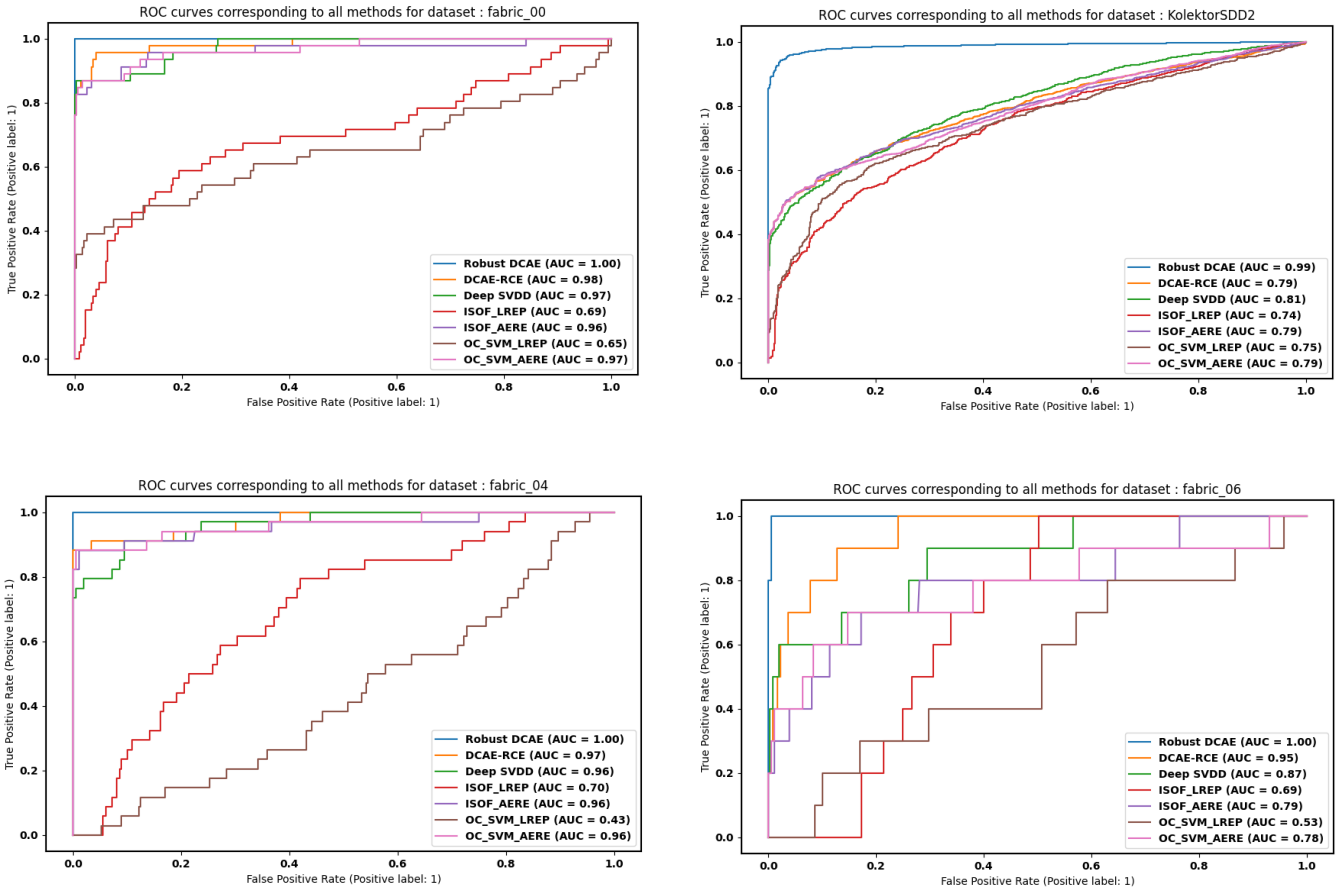


Figure 6 The graphs above illustrate the Receiver Operating Characteristic (ROC) curves generated by all the methods listed in Table 4, pertaining to datasets that are relatively more challenging.

For DCAE and Deep SVDD we trained the AEs using a comparable epoch number. As our Robust DCAE uses a specific convergence criteria, the number of total training iterations is not fixed. However, for DCAE and Deep SVDD a pre-determined epoch number has to be set. So, for a fair comparison we used the average training epoch number obtained from our replicated runs and used that number as the epoch number for DCAE and Deep SVDD. For OC-SVM and Isolation Forest we used the recommended hyper-parameters in their documentation. To establish a well-founded basis for comparison with these two methods we used two different inputs to their algorithm: 1 - the latent representation of the AE (in DCAE) mentioned above which is a feature set of size 256 for each image. This is indicated by the suffix "-LREP" at the end of corresponding column label in Table 4 (e.g. OC-SVM-LREP), 2 - the reconstruction error obtained from the AE, which is a single figure. This is indicated by the suffix "-AERE" at the end of corresponding column label (e.g. ISOF-AERE). In addition to AUC scores given in Table 4, we provide Receiver Operating Characteristic (ROC) curves generated by those methods in Figure 6, to illustrate a more comprehensive and discernible performance comparison of the methods. The ROC curve shows the trade-off between sensitivity (or True Positive Rate (TPR)) and specificity (1 – False Positive Rate(FPR)). For all methods, normalized raw scores are used (rather than predicted labels) to obtain smoother and more indicative curves. The four graphs in the figure pertain to relatively more challenging datasets, to further articulate the level of improvement achieved for those datasets. It can be observed that our method, Robust DCAE, outperforms all of the methods, in some cases with a significant margin. This can be attributed to the ability of the robust convolutional AEs to efficiently learn the stable representative features, through the separation of signal and noise during its training.

6. Concluding Remarks

In this paper, we illustrated the use of Robust Deep Convolutional Autoencoders (RDCAE) for defect detection via two recently introduced industrial quality datasets and we presented some improvements to the training process of RDCAE, that enable us to more reliably manage the convergence of the training. We have illustrated the use of synthetic simulated defects (or structured noise), so that a robust convergence criteria can be settled without compromising the detection performance of the method. We believe that this introduces a plausible and efficient solution to the defect detection process *in the absence of true (real life) abnormal samples*.

Our experiment results are a clear manifestation of the theoretical argument stating the competency of robust deep convolutional AEs in signal-noise separation and thus their ability to more effectively learn the common, stable features as opposed to subtle and inconsistent features that are exhibited by outliers. In other words, compared to plain AEs (e.g DCAE) and many other anomaly detection methods, robust AEs are more adept to learn the boundary between the normal and abnormal samples. There are many application areas, such as automated visual surface inspection systems, that can benefit from the strengths of this method.

It is also worth noting that the clear separation of the noise (i.e. defects) into a separate image (e.g. S images in Figure 4) makes the method more amenable to either further image processing (as a post processing step), or various custom neural network architecture designs for further feature extraction to enable defect identification and classification.

7. Acknowledgments

1. This work was partially supported by TUBİTAK BİLGEM we are grateful for that support.
2. The author declares that he has no known competing interests or personal relationships that could have appeared to influence the work reported in this paper.

8. Availability of data and materials

There are two datasets used in the experiments conducted during our work:

1. The first one is AITEX Fabric Image Database which is publicly available from The Textile Industry Research Association ‐ AITEX of Spain, and is downloadable from https://www.aitex.es/afid/
2. The second one is the Kolektor Surface-Defect Dataset 2 (KolektorSDD2/KSDD2) which is publicly available from The Visual Cognitive Systems Laboratory of Slovenia, and can be downloaded via https://www.vicos.si/resources/kolektorsdd2/

References

[1] V. J. Hodge, J. Austin, A survey of outlier detection methodologies, Artificial Intelligence Review 22 (2) (2004) 85–126. doi:10.1007/s10462-004-4304-y .
[2] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv. 41 (3) (Jul. 2009).
[3] M. A. Pimentel, D. A. Clifton, L. Clifton, L. Tarassenko, A review of novelty detection, Signal Processing 99 (2014) 215–249. doi:10.1016/j.sigpro.2013.12.026 .
[4] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–44. doi:10.1038/nature14539 .
[5] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition (2015). arXiv:1512.03385 .
[6] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, D. Song, Anomalous instance detection in deep learning: A survey (2020). arXiv:2003.06979 .

[7] R. Chalapathy, S. Chawla, Deep learning for anomaly detection: A survey (2019). arXiv:1901.03407 .
[8] D. Bank, N. Koenigstein, R. Giryes, Autoencoders (2020). arXiv:2003.05991 .
[9] M. Schreyer, T. Sattarov, D. Borth, A. Dengel, B. Reimer, Detection of anomalies in large scale accounting data using deep autoencoder networks, CoRR abs/1709.05254 (2017). arXiv:1709.05254 .
[10] D. Zimmerer, F. Isensee, J. Petersen, S. Kohl, K. Maier-Hein, Unsupervised anomaly localization using variational auto-encoders, in: D. Shen, T. Liu, T. M. Peters, L. H. Staib, C. Essert, S. Zhou, P.-T. Yap, A. Khan (Eds.), Medical Image Computing and Computer Assisted Intervention – MICCAI 2019, Springer International Publishing, Cham, 2019, pp. 289–297.
[11] C. Zhou, R. C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, ACM, New York, NY, USA, 2017, pp. 665–674. doi:10.1145/3097983.3098052 .
[12] R. Chalapathy, A. K. Menon, S. Chawla, Robust, deep and inductive anomaly detection, in: M. Ceci, J. Hollmén, L. Todorovski, C. Vens, S. Džeroski (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer International Publishing, Cham, 2017, pp. 36–51.
[13] M. Ribeiro, A. E. Lazzaretti, H. S. Lopes, A study of deep convolutional auto-encoders for anomaly detection in videos , Pattern Recognition Letters 105 (2018) 13 – 22, machine Learning and Applications in Artificial Intelligence. doi:https://doi.org/10.1016/j.patrec.2017.07.016 .
[14] H. Y. Ngan, G. K. Pang, N. H. Yung, Automated fabric defect detection?a review , Image and Vision Computing 29 (7) (2011) 442–458. doi:https://doi.org/10.1016/j.imavis.2011.02.002 .
[15] K. Hanbay, M. F. Talu, Ömer Faruk Özgüven, Fabric defect detection systems and methods?a systematic literature review, Optik 127 (24) (2016) 11960–11973.
[16] M. F. Nisha, P. Vasuki, S. M. M. Roomi, Survey on various defect detection and classification methods in fabric images, Journal of Environmental Nano Technology (JENT) 6 (2) (2017) 20–29.
[17] C. Li, J. Li, Y. Li, L. He, X. Fu, J. Chen, Fabric defect detection in textile manufacturing: A survey of the state of the art, Security and Communication Networks 2021 (2021) 9948808.
[18] T. Czimmermann, G. Ciuti, M. Milazzo, M. Chiurazzi, S. Roccella, C. M. Oddo, P. Dario, Visual-based defect detection and classification approaches for industrial applications?a survey, Sensors 20 (5) (2020).
[19] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, R. C. Williamson, Estimating the support of a high-dimensional distribution, Neural Comput. 13 (7) (2001) 1443–1471.
[20] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, ACM Trans. Knowl. Discov. Data 6 (1) (Mar. 2012). doi:10.1145/2133360.2133363 .
[21] S. Hariri, M. Carrasco Kind, R. J. Brunner, Extended isolation forest, IEEE Transactions on Knowledge and Data Engineering (2019) 1–1 doi:10.1109/TKDE.2019.2947676 .
[22] D. M. Tax, R. P. Duin, Support vector data description, Machine Learning 54 (1) (2004) 45–66.
[23] L. Ruff, R. A. Vandermeulen, N. Görnitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, M. Kloft, Deep one-class classification, in: Proceedings of the 35th International Conference on Machine Learning, Vol. 80, 2018, pp. 4393–4402.
[24] R. Chalapathy, A. K. Menon, S. Chawla, Anomaly detection using one-class neural networks (2018). arXiv:1802.06360 .
[25] E. Parzen, On estimation of a probability density function and mode , The Annals of Mathematical Statistics 33 (3) (1962) 1065–1076. URL http://www.jstor.org/stable/2237880
[26] D. Gong, L. Liu, V. Le, B. Saha, M. Mansour, S. Venkatesh, A. Hengel, Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection (10 2019). doi:10.1109/ICCV.2019.00179 .
[27] S. Edwards, M. S. Lee, Using convolutional neural network autoencoders to understand unlabeled data, in: T. Pham (Ed.), Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications, Vol. 11006, International Society for Optics and Photonics, SPIE, 2019, pp. 444 – 458. doi:10.1117/12.2518459 .
[28] P. Wu, J. Liu, F. Shen, A deep one-class neural network for anomalous event detection in complex scenes, IEEE Transactions on Neural Networks and Learning Systems 31 (7) (2020) 2609–2622.

[29] P. Schlachter, Y. Liao, B. Yang, Deep one-class classification using intra-class splitting , 2019 IEEE Data Science Workshop (DSW) (Jun 2019). doi:10.1109/dsw.2019.8755576 .

[30] E. Plaut, From principal subspaces to principal components with linear autoencoders (2018). arXiv:1804.10253 .

[31] E. J. Candès, X. Li, Y. Ma, J. Wright, Robust principal component analysis, J. ACM 58 (3) (Jun. 2011).

[32] F. De la Torre, M. J. Black, Robust principal component analysis for computer vision, in: Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, Vol. 1, 2001, pp. 362–369 vol.1.

[33] H. Akrami, A. A. Joshi, J. Li, S. Aydore, R. M. Leahy, Robust variational autoencoder (2019). arXiv:1905.09961 .

[34] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders , in: Proceedings of the 25th International Conference on Machine Learning, ICML ?08, Association for Computing Machinery, New York, NY, USA, 2008, p. 1096?1103. doi:10.1145/1390156.1390294 .

[35] A. Collin, C. D. Vleeschouwer, Improved anomaly detection by training an autoencoder with skip connections on images corrupted with stain-shaped noise, 2020 25th International Conference on Pattern Recognition (ICPR) (2021) 7915–7922.

[36] P. Liznerski, L. Ruff, R. A. Vandermeulen, B. J. Franks, M. Kloft, K.-R. Müller, Explainable deep one-class classification , in: International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=A5VV3UyIQz

[37] S. Mosci, L. Rosasco, M. Santoro, A. Verri, S. Villa, Solving structured sparsity regularization with proximal methods, in: J. L. Balcázar, F. Bonchi, A. Gionis, M. Sebag (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 418–433.

[38] A. Douady, Julia Sets and the Mandelbrot Set, Springer, Berlin, Heidelberg, 1986, pp. 161–174. doi:10.1007/978-3-642-61717-1_13 .

[39] G.-L. Zhang, M.-Q. Cai, X.-L. He, X.-Z. Gao, M.-D. Zhao, D. Wang, Y. Li, C. Tu, H.-T. Wangrmark, Pseudo-topological property of julia fractal vector optical fields, Opt Express 27 (9) (2019) 13263–13279.

[40] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980 .

[41] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. doi:10.1109/5.726791 .

[42] A. Paszke, et al. Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[44] J. Silvestre-Blanes, T. Albero-Albero, I. Miralles, R. Pérez-Llorens, J. Moreno, A public fabric database for defect detection methods and results, Autex Research Journal 19 (4) (2019) 363 – 374. doi:https://doi.org/10.2478/aut-2019-0035 .

[45] J. Božič, D. Tabernik, D. Skočaj, Mixed supervision for surface-defect detection: from weakly to fully supervised learning, Computers in Industry (2021).

[46] Zhou Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Transactions on Image Processing 13 (4) (2004) 600–612.

[47] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, scikit-image: image processing in python, PeerJ 2 (2014) e453.