POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Pixelwise Anomaly Detection exploiting Steerable Filters based methods

Supervisor: Prof. Giacomo Boracchi
Co-supervisor: M.Sc. Luca Frittoli

M.Sc. Thesis by:
Andrea Bionda,
matriculation number 921082

Academic Year 2019-2020

To my beloved family.

Abstract

In the last few years, automatic detection of anomalies in images has become very popular in the industrial quality control, where the goal is to check the presence of unexpected defects in final products. In this thesis, we address the problem of the automatic pixelwise identification and segmentation of anomalies in textured images. In this scenario, abnormal areas can be visually identified because they highly deviate in terms of intensity, structure and contrast from the rest of the image, which follows a specific periodic pattern.

We propose two different approaches to tackle the aforementioned problem. Both are based on Semi-Supervised Machine Learning techniques trained only with normal patches, i.e., portions of images which do not contain anomalies. The idea is to learn the pattern of normal textures and deem as abnormal the patches that are not alike. The first method uses the *Structural Texture Similarity* (STSIM), an algorithm based on the *Steerable filters*, to extract the features which belong to the normal set. Then, we learn a density model that characterizes these features, and finally we classify each test patch calculating the probability of belonging to the same distribution. The second solution exploits a Deep Learning model called *autoencoder* to learn how to extract normal features. In this case, the model learns how to compress the input data in a small set of features, called latent representation. The autoencoder is then trained to reconstruct the compressed features to be as similar as possible to the input data. In this scenario, we propose to use the *Complex Wavelet Structural Similarity* (CW-SSIM) as loss function. CW-SSIM is a similarity metrics based on the Steerable filters, which in our case it is deployed to assess the similarity between the input and the reconstructed patch during the training. The rationale is that the model is trained to reconstruct only normal textures and should fail to do so for anomalous regions. We finally reconstruct each test image using the trained autoencoder, and we label as anomalies the portions of reconstructed image which present high visual dissimilarity from the original one.

We propose again to use CW-SSIM as anomaly metrics to automatically identify the differences between these two images.

This thesis provides the theoretical and practical basics behind the two Anomaly Detection solutions proposed, which introduce the concept of Steerable filters in this environment, exploring two completely different approaches. In the last, we compare the performance achieved by our solutions with the State of the Art techniques on a challenging real-world images dataset of nanofibrous materials, acquired by an electronic microscope. In particular, the method based on CW-SSIM achieves significant performance gains in pixelwise defect identification.

Sommario

Negli ultimi anni, il riconoscimento automatico di anomalie all'interno di immagini digitali è divenuto un elemento critico in molte attività industriali, al fine di controllare la qualità dei prodotti finiti. In questa tesi, affronteremo il problema dell'identificazione e segmetnazione puntuale di anomalie all'interno di immagini di tipo texture. In questo ambiente, le aree anomale sono distinguibili in quanto differenti in termini di intensità, struttura e contrasto dal resto dell'immagine, il quale segue uno schema specifico e periodico.

Al fine di affrontare questo problema, nel seguente elaborato proponiamo due approcci sostanzialmente differenti. Entrambi sono basati su tecniche di Semi-Supervised Machine Learning, le quali vengono addestrate unicamente con porzioni di immagini che non contengono anomalie. L'idea generale su cui si basano queste tecniche è far imparare alla macchina lo schema periodico delle porzioni di immagini normali e successivamente, evidenziare le aree che maggiormente deviano da esso. Il primo metodo proposto usa *Structural Texture Similarity* (STSIM), algoritmo basato sugli *Steerable filters*, per estrarre dalle porzioni di immagini normali le caratteristiche che le contraddistinguono. Successivamente viene appreso il loro modello di densità e infine ogni porzione di test viene classificata in base alla probabilità di appartenere alla medesima distribuzione. Il secondo metodo è basato sull'uso di *autoencoder*, un modello di Deep Learning che impara come estrarre le caratteristiche delle porzioni normali. In questo caso, il modello impara a comprimere i dati in input in un ridotto set di caratteristiche, chiamato in gergo tecnico rappresentazione latente. L'autoencoder viene quindi allenato a ricostruire le caratteristiche compresse, generando un output il più somigliante possibile all'input. In questo scenario, proponiamo l'utilizzo di *Complex Wavelet Structural Similarity* (CW-SSIM) come loss function. CW-SSIM è una metrica che quantifica la similarità fra due immagini basata sull'uso degli Steerable filters, che nel nostro caso viene usata per stimare la somiglianza tra l'immagine in input e quella ricostruita durante la fase di addestramento del modello. Visto che durante questa fase il modello apprende le caratteristiche unicamente delle immagini normali, è possibile presupporre che esso sia incapace di ricostruire le anomalie. Infine, ogni immagine di test viene ricostruita utilizzando l'autoencoder precedentemente allenato e vengono etichettate come anomalie le porzioni dell'immagine ricostruita che presentato una grossa dissimilarità rispetto alla loro versione originale. Proponiamo ancora l'uso di CW-SSIM come metrica per identificare automaticamente le differenze tra queste due immagini.

Nel seguente elaborato, vengono presentati i concetti teorici e pratici su cui sono basate le due tecniche di riconoscimento delle anomalie proposte, le quali introducono il concetto di Steerable filters seguendo due differenti approcci. Nella sezione finale della tesi, mettiamo a confronto le performance delle nostre soluzioni con quelle già presenti in letteratura utilizzando un dataset di immagini sulle nanofibre, acquisite con un microscopio elettronico. In particolare, il metodo basato su CW-SSIM raggiunge un significativo incremento di prestazioni nell'identificazione puntuale dei difetti.

Acknowledgements

I would like to thank my parents, Mario and Monica, for the support given in these academic years, and Natalia which helped me in the revision of this thesis. I want also to thank Prof. Giacomo Boracchi and M.Sc. Luca Frittoli for having encouraged me to deeply investigate this topic, exploring it through different perspectives.

Contents

Chapter 1

Introduction

Computer Vision is the field of *Information Technology* (IT) which addresses image-related problems such as classification and Anomaly Detection. The mainstream approach for solving visual understanding problems consists in two parts: the first is called *Feature Extraction*, able to extract interesting information from the input data, while the second leverages this information to solve the given problem. In the Computer Vision field, when we are dealing with images, usually it is not possible to directly apply Anomaly Detection algorithms on raw inputs. This would require a large number of parameters and several computing resources to process, due to the high dimension and complexity of images. For these reasons, it is necessary to implement a first step that is able to collect meaningful information and reduce data dimension. In recent years, *Machine Learning* (ML) has revolutionized the way in which the second part is tackled. ML algorithms learn the parameters of a mathematical model from several training data, to provide for each input the correct outcome, e.g. the class label in the classification. *Deep Learning* (DL), a specific branch of ML, extends the above concepts, training a model that is also able to extract the most interesting features for the assigned problem, combining the two Computer Vision tasks in an end-to-end learning procedure. All the Machine Learning techniques (including DL ones) need a large amount of data to be properly initialized; in principle, the more they are the better the model is able to solve the given problem.

In particular, this thesis is focused on Computer Vision *Anomaly Detection*, based on Machine Learning techniques. We are interested in the design and assessment of ML models capable of learning the distribution of regular inputs and identifying as anomalous the areas of evaluation images which are less explicable in a probabilistic sense [27]. The Anomaly Detection problem, also called Novelty Detection, refers to the identification of inputs that do not have the expected characteristics. Actually, Anomaly Detection can be seen as a two-class classification problem, in which the goal is to label each input with the correct class, namely *Normal* and *Anomalous*. The straightforward approach to tackle the problem is *Supervised Anomaly Detection*. In this framework, the models are trained from a dataset of annotated images and their correct labels, learning a traditional two-class classification model. The structure of a Supervised anomaly detection procedure can be found in Figure 1.1. These solutions are particularly effective when all the possible kinds of anomalies are entirely contained in the training set and a large amount of annotated data is provided, which is not always possible.

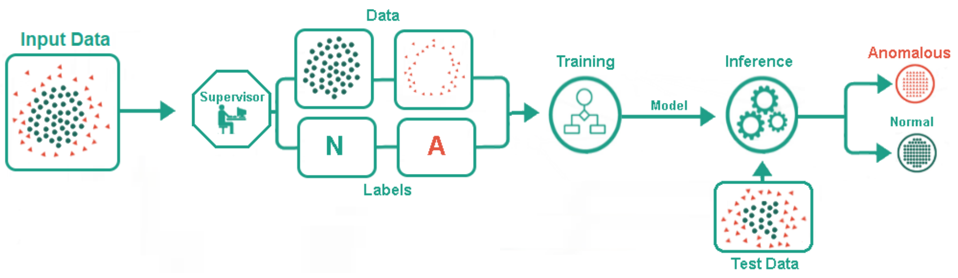

Figure 1.1: Supervised Anomaly Detection.

In real world scenarios, such as industrial applications, having a complete dataset of all the kinds of anomalies is very rare. For example, we can think about a foundry in which every day many castings are produced and the controller has to perform quality inspection, maybe using some cameras or X-ray scans which are able to retrieve information also inside the semifinished product. Anomaly Detection instruments are really helpful in this context, since they reduce the human work increasing the productivity. To implement a Supervised method, we need the foundry to provide us all the types of defects that it could encounter during the production, which is of course very difficult. Normal data are usually easier to collect, since they represent the largest part of the daily production, while anomalies should be very rare. In these contexts, *Semi-Supervised Anomaly Detection* approaches require only part of the given input images to be labeled. The models belonging to this framework are trained using only data labeled as normal, which means that they learn how normal inputs are shaped. The structure of a Semi-Supervised anomaly detection procedure can be found in Figure 1.2. All these approaches are based on the idea that outliers, i.e. anomalies, can be identified, because their distribution differs from the one of the ordinary data. For the aforementioned reasons, Semi-Supervised Anomaly Detection is the straightforward framework in these contexts.

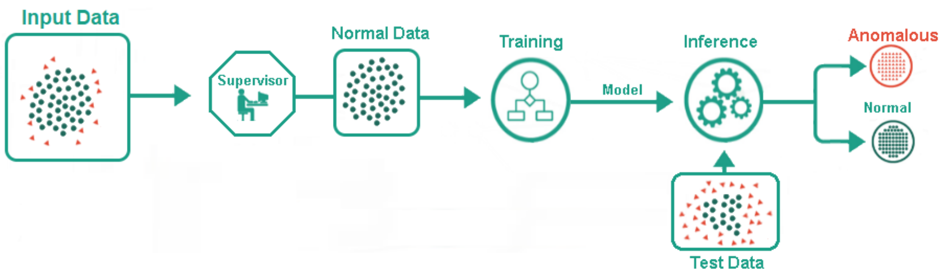

Figure 1.2: Semi-Supervised Anomaly Detection.

The segmentation of anomalies in textured images is very important in real world applications in order to automatize human works, e.g. in quality control for inline industrial production, or support it as in medical scenarios when a diagnosis based on images evaluations, such as a CAT (Computerized Axial Tomography), has to be made. It is a very challenging Computer Vision problem since images lie in a very wide and complex space and thus the recognition of small structural changes, which could or could not represent anomalies, is not an easy task. The goal of the thesis is to propose two Semi-Supervised ML approaches applied to Anomaly Detection, which use two different image similarity metrics named *Structural Texture Similarity* (STSIM) [41] and *Complex Wavelet Structural Similarity* (CW-SSIM) [29], based on the Steerable filters decomposition [10]. We will refer to them as the *Anomaly Detection method based on STSIM* and *Anomaly Detection method based on CW-SSIM applied to Autoencoders*. We choose to build our methods around these two metrics because Steerable filters are able to capture important features from textured images. In the first solution, we adapt STSIM to extract the features that belong to the normal set in order to fit them to a density model. Once the parameters which characterize the training patches are learned, we classify each test patch calculating the probability of belonging to the anomalous-free distribution. In the second procedure, instead, we propose to use CW-SSIM as loss function of an autoecoder. This model learns to compress the normal patches into a vector of features and to expand this latent representation to be as similar as possible to the input data. The CW-SSIM is deployed in the training procedure to assess the similarity between the input and the reconstructed patch. Since the autoencoder is trained to reconstruct only normal textures, the test images will not be perfectly reconstructed in the anomalous areas, presenting distorted patterns. In order to localize these areas, we propose to apply again CW-SSIM between the original test image and the reconstructed one, which will present higher dissimilarity in the abnormal locations. In our experiments, we will demonstrate the practical effectiveness of our choices using a dataset of nanofibrous materials (which can be approximated to a textured one, apart from anomalies). In particular, the second solution outperforms the methods present in literature in all the evaluation metrics proposed.

1.1 Problem Formulation

Let I be an image, i.e. a matrix, of size $w \times h \times c$ of values $I(i,j) \in \mathbb{N}$, where h defines the height, w the width and c represents the number of color channels. The latter in our case is set to one, since we are dealing only with gray scale images. Each element, or pixel, of the image $I(i,j)$ at position (i,j) is a single value that can range from 0 to $2^r - 1$, where r is the color depth (usually referred to as bit depth). For each image I, we define the binary mask of anomalies Ω_I as a matrix of size $w \times h$ such that:

$$\Omega_I(i,j) = \begin{cases} 1, & \text{if } I(i,j) \text{ is an anomalous element of image I,} \\ 0, & \text{otherwise.} \end{cases} \tag{1.1}$$

Given an image I, the Anomaly Detection problem requests to automatically find the binary mask $\hat{\Omega}_I$ that best approximates the reference mask of anomalies Ω_I, also referred to as the ground truth.

1.2 Structure of the thesis

We organize the thesis in the following chapters:

- Chapter 2 introduces the basic concepts about the followed approaches and present the State of the Art techniques that address our problem.
- Chapter 3 is chosen to explain in detail important methodologies that will be useful in the presentation of our proposed methods. In particular, here the Steerable filters are presented.
- Chapter 4 is where our contributions are explained. Here we describe all the steps that are part of our proposed solutions.

- Chapter 5 lists all the implementation details behind our proposed techniques such as the parameters' choice.
- Chapter 6 presents the experiment results and discusses the differences between our solutions and the ones present in the State of the Art.
- Chapter 7 shows the conclusions and future work.

Chapter 2

State of the Art

In this section, we introduce the principles of our two proposed solutions: the Anomaly Detection method based on STSIM, which follows a *Hand-Crafted Feature-based* approach and the one based on CW-SSIM, which follows a *Data-Driven* approach. In Sections 2.1 and 2.2, we also report some examples of real applications in the Anomaly Detection environment in which the two approaches are followed respectively. The main difference between the two is how they intend the feature extraction procedure. We recall that feature extraction is a process of dimensionality reduction by which raw images are compressed to a more manageable set of features. Feature extraction effectively reduces the amount of data that has to be processed, decreases data redundancy, while still accurately and completely describes the original dataset. In some cases, the entire image is reduced in sub-portions called patches before applying feature extraction. This is usually done when it is necessary to capture local image characteristics or when dealing with too large images.

2.1 Hand-Crafted Feature-based approach

The Hand-Crafted Feature-based approach is based upon the application of manually designed models that are able to extract specific types of features which are thought fundamental to solve the assigned problem. This type of feature extraction is usually referred to as *Hand-Crafted feature extraction*, since these characteristics are manually engineered by the data scientists. These features are then used to evaluate whether the input data belong to the normal or anomalous class, using an Anomaly Detection model.

2.1.1 Automated surface inspection using Gabor filters

Among Hand-Crafted Feature-based methods, we can cite the use of *Gabor filters*, as a Hand-Crafted feature extraction algorithm in automated surface inspection [35]. *Tsai et al.* demonstrate that it is possible to extract the energy response from a textured image applying several convolutions with a specific set of Gabor filters. In this context, a pixel with zero response $E(i,j) = 0$ can be logically considered belonging to a homogeneous surface, while a pixel with a large response is probably in a non-homogeneous area. Therefore, all pixels associated with the homogeneous texture will be sifted out, and only those pixels corresponding to a local non-homogeneity will be considered. The proposed evaluation model is based on a statistical evaluation: as first step the authors compute the mean μ_N and the standard deviation σ_N of the energy responses coming from the application of Gabor filters to anomaly free images. Then, they calculate the binary mask of anomalies evaluating each pixel highlighted in the anomalous image by comparing its energy with the previously extracted features:

$$\hat{\Omega}_I = \begin{cases} 1, & \text{if } E(i,j) > \mu_N + C \cdot \sigma_N, \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

where $\hat{\Omega}_I$ is the mask of anomalies and C is a manually tuned parameter which controls the sensibility of the evaluation.

2.1.2 Steerable pyramid for directional structures detection

Another Hand-Crafted Feature-based technique is the application of Steerable filters [10] for the diagnosis of breast cancer [8]. Steerable filters are directional derivative operations, which can vary in scale and orientation, in a way to provide multi-scale and multi-orientation analysis. The process is divided in two steps: in the first step, Steerable filters decomposition is used as an oriented contrast enhancement over the image, allowing to highlight the possible presence of anomalies. A threshold is then applied over the enhanced images to remove non interesting pixels (certainly non-cancer zones). Then, in the second step, the Steerable filters decomposition is used again but applied only over portions of the image above the threshold. This step allows to detect the presence of directional patterns, which are most probably tumor masses, by subtraction of the filtered patch with the original one. Each pixel position is considered anomalous if the local difference between the two images is above a second manually tuned threshold.

2.2 Data-Driven approach

Data-Driven approach is based upon the idea that the machine is trained to decide which features are important in a specific problem. These approaches require to define a model composed by a set of parameters, that will be tuned during training. In this phase, the model deployed will incrementally learn, through examples, how to extract the features that best describe input data. These features, as in the previous Section 2.1, are then used to evaluate whether the input data belong to the normal or anomalous class. In some cases, this approach allows that the feature extraction model and the Anomaly Detection one are trained together in an end-to-end fashion.

2.2.1 Sparse Dictionary Learning

Sparse coding is a learning method which aims at finding a sparse representation of the input data (also known as sparse coding) in the form of a linear combination of basic elements. These elements are called atoms and they compose a dictionary. One of the key principles of dictionary learning is that the dictionary has to be inferred from the input data.

Defect Detection by Learning a Dictionary of Normal Data

In the study by *Boracchi et al.* [7], the Anomaly Detection problem is tackled by learning a model providing sparse representations of patches extracted from normal images. This model can be expressed in terms of a dictionary $D \in \mathbb{R}^{p,n}$ providing sparse approximation of all normal patches $s_c \in \mathbb{R}^p$

$$s_c \approx Dx_c = \sum_{i=1}^{n} d_i x_{c,i}, \tag{2.2}$$

where $x_c \in \mathbb{R}^n$ is the coefficient vector assumed to be sparse and d_i identify each dictionary atom. The dictionary learning problem actually corresponds to learning both the dictionary D and the sparse representation $x_c \in \mathbb{R}^{n,m}$ for a given training set $S \in \mathbb{R}^{p,m}$ containing m normal patches. Dictionary learning is then formulated as the following optimization problem:

$$[\hat{D}, \hat{X}] = \arg\min_{D,X} \frac{1}{2} \|DX - S\|_2^2 + \lambda \|X\|^1 , \tag{2.3}$$

where the second term is the regularization term. The minimization algorithm used is the Alternating Direction Method of Multipliers (ADMM) [6].

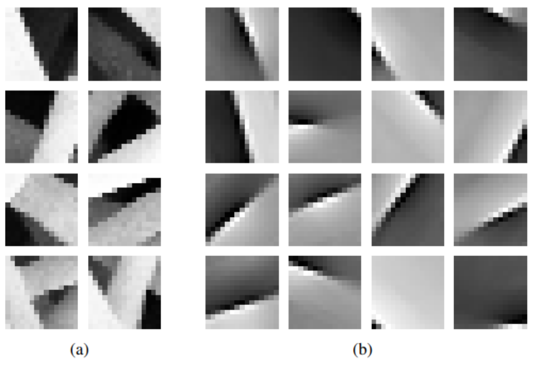

Figure 2.1: a) Examples of training patches. b) Elements of the dictionary learned by the ADMM algorithm [6]. This image is from [7].

After the training step, each patch S_c in a test image is independently analyzed to determine whether it admits a sparse representation with respect to D. For this purpose, the authors compute for each test patch s_c the corresponding sparse coding x_c by minimizing:

$$x_c = \arg\min_x \frac{1}{2} \|Dx - s_c\|_2^2 + \lambda \|x\|^1 , \tag{2.4}$$

to generate the features vector $f(c)$:

$$f(c) = \left[\left\| \hat{D}x_c - s_c \right\|_2^2 , \|x_c\|^1 \right] . \tag{2.5}$$

Using the features extracted from normal patches, the authors model the distribution ϕ, of the features extracted from normal patches, by Kernel Density Estimation (KDE) [23], adopting a kernel based on linear diffusion. Then a patch s_c is considered anomalous when $f(c)$ falls in a low-density region of ϕ.

Boracchi et al. in [4] also propose a similar sparse approach in which they consider anomalous a patch when its features vector $f(c)$ is above a certain threshold γ:

$$\sqrt{(f(c) - \mu)^T \Sigma^{-1} (f(c) - \mu)} > \gamma, \tag{2.6}$$

where μ and Σ are the expectation and the covariance matrix of distribution ϕ, estimated from normal patches. Once all the patches coming from the test image are evaluated, they are merged to create a full resolution map in which anomalies are highlighted with respect to the rest of the image.

2.2.2 Convolutional Neural Network

Another popular Data-Driven approach consists in training a Deep Neural Network for feature extraction. First, we introduce the concept of *Feed-Forward Neural Network* (FFNN) [13] that will be useful in the discussion of *Convolutional Neural Network* (CNN) [33]. Traditional FFNNs comprise an input layer, one or more hidden layers, an output layer and a set of weights called the network's parameters. Each layer is composed by a number of nodes called neurons. The inputs are initially multiplied by their respective weights, summed and passed through an activation function, which determines the output of the neuron. If that output exceeds a given threshold, it activates the node, passing data to the next layer in the network. This process is iterated for each hidden layer.

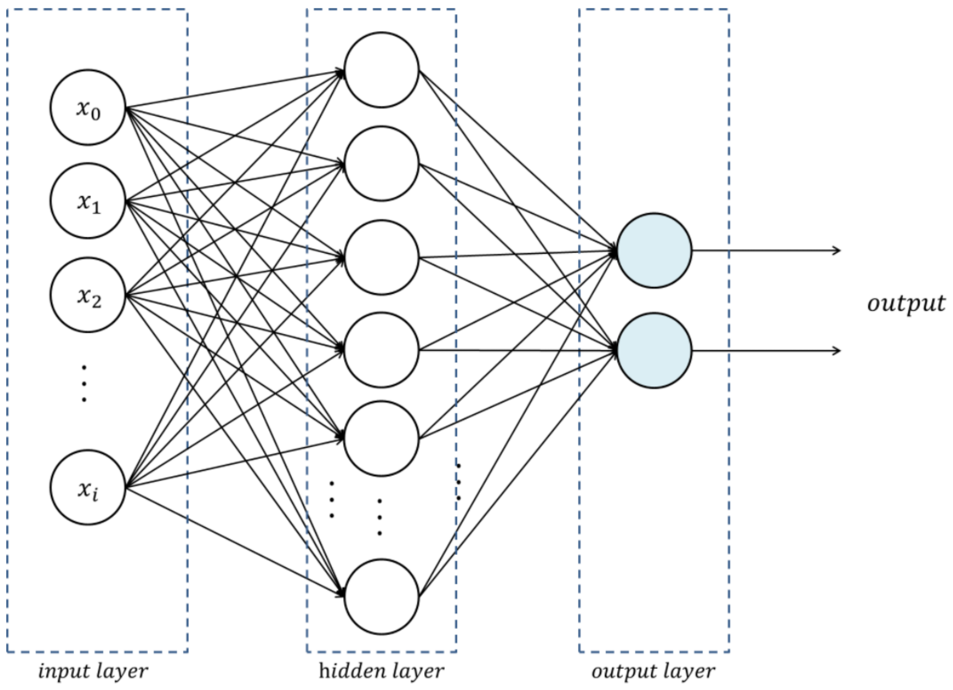

Figure 2.2: Feed Forward Neural Network.

The training is performed using a dedicated set of data called training set, which will not be used to evaluate the method. The goal of this procedure is to find the set of weights which minimizes the distance to the target value, i.e. the expected output for each input in the network. The metrics used to calculate the distance between the output of the FFNN and the target value is called loss function. The optimization algorithm has the objective to minimize the loss function reaching its point of convergence, or local minimum. This is achieved by adjusting the weights of the network. The process in which the algorithm adjusts its weights is through gradient descent, allowing the model to determine the direction to take in order to reduce the loss in the weights space. [13].

In recent years, Convolutional Neural Networks have obtained great results in the Computer Vision field, thanks to the large development of Deep Learning techniques. Applications using this approach often require less expert analysis and fine-tuning and exploit the tremendous amount of video data available in today's systems. Another advantage of the CNN framework is that it can be re-trained using a custom dataset for almost any use case, contraries to algorithms based on Hand-Crafted features which tend to be more domain-specific [22]. CNNs are Feed Forward Neural Networks specifically designed to extract interesting features in the images domain. There are different types of layers, namely *Convolutional Layer*, *Pooling Layer* and *Fully-Connected Layer*.

Convolutional Layer: The Convolutional Layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input channels.

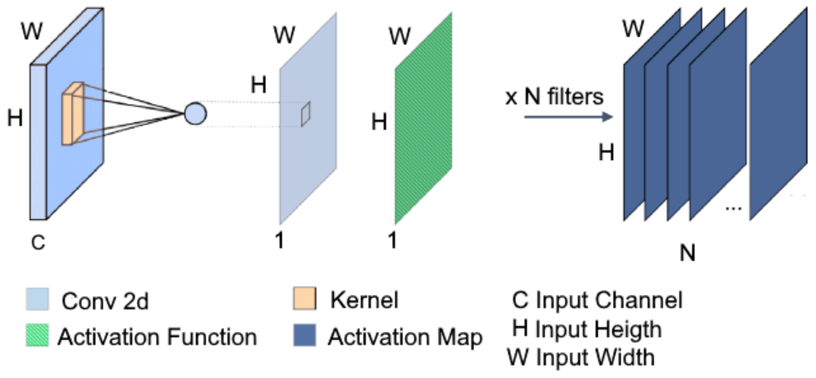

Figure 2.3: Convolutional layer. This image is from [33].

The convolution consists in sliding the filter across width and height of the input volume, and computing a dot product between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume, we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some types of visual features, such as edges, particular blobs or patterns. We can apply several filters in each Convolutional layer, and each will produce a separate 2-dimensional activation map. These activation maps are finally stacked along the depth (channel) dimension and produce the output volume. The three hyperparameters which control the size of the output volume are the depth, the stride and the padding. The depth of the output volume corresponds to the number of filters we would like to use. The stride and the padding, instead, are used to control the spatial dimension of the output [33].

Pooling Layer It is common to periodically insert a Pooling layer in-between successive Convolutional layers. Its purpose is to progressively reduce the spatial size of the features representation, allowing the research of interesting patterns at different image scales, reducing the amount of parameters and computation in the network, and hence also to control overfitting.

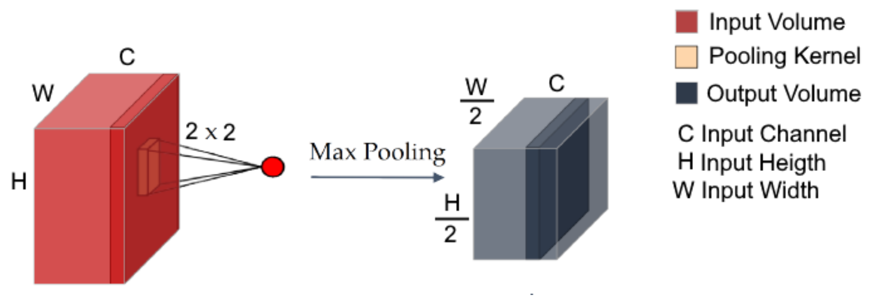

Figure 2.4: Pooling layer. This image is from [33].

The Pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a Max Pooling layer with filters of size 2×2 with a stride of 2: it downsamples every depth slice in the input by a factor of 2 along both width and height. Every 2×2 region is then replaced by its maximum value. The depth dimension remains unchanged [33]. Many other Pooling layers exist, which apply other operations such as minimum and average.

Fully-Connected Layer Neurons in a Fully Connected layer have full connections to all activations in the previous layer, as seen in Feed-Forward Neural Networks.

These layers are placed at the end of the network to process the output of the previous layers in order to produce the expected output [33]. In Anomaly Detection, for example, the possible output is a 2-dimensional

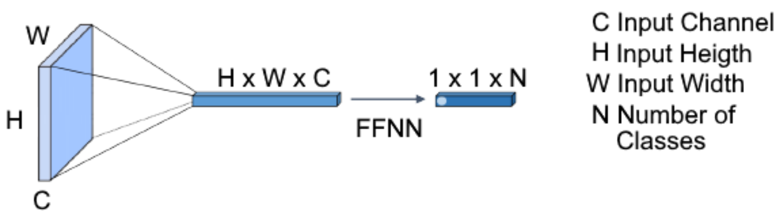

Figure 2.5: Fully Connected layer. This image is from [33].

vector containing the probability of belonging to the two classes named Anomalous and Normal.

Loss function The loss function is the measure that tells how much the output is far from the target value. This measure is used to learn the parameters that minimize it. Clearly, it depends on the task that the model will tackle. For example in supervised binary classification, in which we want to learn the label to assign to each input image, it is common practice to define the loss function as the binary cross-entropy function:

$$\mathcal{L}(w) = -\sum_{i=1}^{N} t_i \log(\phi(x_i, w)) + (1 - t_i) \log(1 - \phi(x_i, w)), \tag{2.7}$$

where x_i and t_i correspond to the input and the target label of the i^{th} sample over N, $\phi(x_i, w)$ is the Convolutional Neural Network that takes in input x_i, and produces in output the probability of the correct class label using parameters w.

Deep One-Class classification

A remarkable example of a CNN employed in Anomaly Detection framework is the Deep Support Vector Data Description (Deep SVDD) by *Ruff et al.* [28]. In contrast to the other methods presented here, it does not produce a full resolution anomaly map, but an anomaly score that says how much the whole image is far from the normal distribution. In principle, it could be possible to revisit this approach to produce an anomaly map. One could train Depp SVDD patch-wise, then evaluate a test image by extracting the anomaly score of each single patch and upsamling the scores to the original patch size.

The idea is to train a Convolutional Neural Network to produce a compact representation of the input images in the latent space $F \in \mathbb{R}^p$, while minimizing the volume of a hypersphere that encloses the representations of normal data. This method forces the network to extract a compact data representation, since the network must closely map the data points to the center of the sphere. The authors define Deep SVDD optimization problem as:

$$\arg\min_{\hat{w}} \frac{1}{n} \sum_{i=1}^{N} \|\phi(x_i;\hat{w}) - C\|^2 + \frac{\lambda}{2} \sum_{l=1}^{L} \|\hat{w}_l\|_F^2 \,, \tag{2.8}$$

where $\phi(x_i; w) \in F$ is the compact representation of input x given by the network ϕ with parameters w. The second term is a network weight decay regularizer with hyperparameter $\lambda > 0$. This minimization aims to find a hypersphere of minimum volume with center $C \in F$, minimizing the mean distance of all data representations to the center.

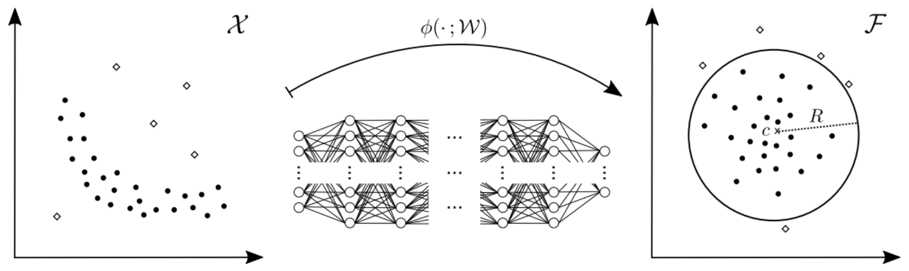

Figure 2.6: Deep SVDD learns the parameters of a CNN while it attempts to map most of the data network representations into a hypersphere characterized by center C of minimum volume. Normal examples should fall within, whereas anomalies should fall outside the hypersphere. This Figure is from [28].

For a given test point x, the anomaly score s is defined as the distance of the point to the center of the hypersphere:

$$s(x) = \|\phi(x_i;\hat{w}) - C\|^2 \,, \tag{2.9}$$

where $\hat{w}$ denotes the learned parameters during training. The center of the hypersphere C can be found empirically as the mean of the network representations, that results from performing an initial forward pass on some training data samples.

Anomaly Detection by CNN-based self-similarity

Another method that applies CNN to Anomaly Detection is [20] by *Napoletano et al.*, which propose a Semi-Supervised method for automatic detection and localization of anomalies on images of nanofibrous materials. This is a region-based method: it measures how much a region of the test image is anomalous with respect to a dictionary composed of “normal” patches taken from the training images. The goal is to segment (localize) the region in the image which is anomalous, building the corresponding anomaly map. This approach exploits the feature extraction properties of the Convolutional Neural Networks into a dictionary learning approach.

In contrast to Deep SVDD [28], in this case this is not an end-to-end learning approach, but the features vectors are extracted exploiting a pre-trained Convolutional Neural Network. In particular, the network architecture adopted is based on the ResNet-18 architecture, pre-trained on the set of images defined by the ILSVRC 2015 challenge, which, as demonstrated in [20], works particularly well as a feature extraction model on texture images similar to the visual appearance of the SEM images.

The features vectors extracted by the CNN from normal patches are first reduced in dimension, applying the *Principal Component Analysis* (PCA) [39] and then grouped by subregions into K clusters, using the *K-Means* algorithm [2], forming the dictionary of normal features.

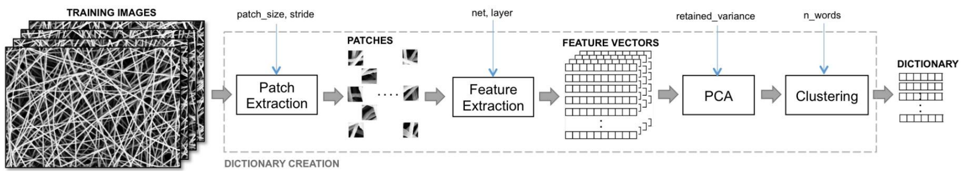

Figure 2.7: Dictionary creation during the training phase. This figure is from [20].

The degree of abnormality of a patch is obtained by processing it as in the training phase (feature extraction and PCA) and then averaging the Euclidean distances between its features and the m most similar subregions of the dictionary. Once all the patches coming from the test image are evaluated, they are merged to create a full resolution map in which anomalies are highlighted with respect to the rest of the image.

2.2.3 Convolutional Autoencoder

Convolutional Autoencoders (CAE) belong to the family of Convolutional Neural Networks used for data reconstruction. The original aim was to deal with dimensionality reduction problems [36] or feature extraction in an unsupervised fashion.

The general structure of an autoencoder is composed by two sub-networks: *Encoder* and *Decoder*. The encoder $E(\cdot)$ is a classical CNN which takes in input the image x and generates a compressed features vector called latent representation $z = E(x)$ (also referred as bottleneck). The decoder $D(\cdot)$ is another network that, starting from the latent representation z, upscales the feature vectors to the resolution of the input image $y = D(z) = D(E(x))$. The scheme of the network can be found in Figure 2.8.

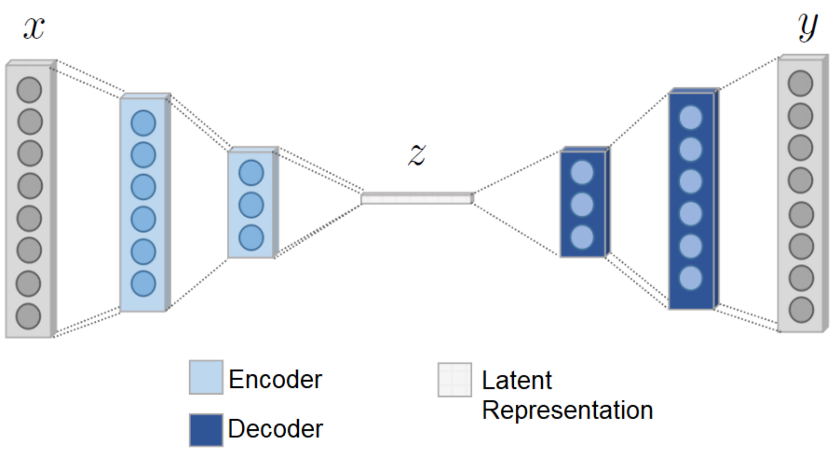

Figure 2.8: Convolutional autoencoder architecture.

The training of a convolutional autoencoder takes place in an unsupervised fashion, since it is trained to reconstruct its input, and thus it does not need any supervised label. The most common used loss function is the Mean Squared Error (MSE):

$$\mathcal{L}_{AE}(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2, \tag{2.10}$$

where x and y are respectively the input image and the reconstructed one and N is the number of pixels involved in the comparison. This point-by-point metrics simply calculates the l^2 distance between the corresponding pixels in the two images and averages the distance.

The idea behind CAE is to compress the data into the latent representation z, which represents the features of the input data. Without this compression-decompression approach, the model will learn the trivial identity function. The Convolutional term indicates that all the layers presented, both in the encoder and in the decoder, are Convolutional and Pooling layers. Clearly, in the encoder the Pooling layers reduce the spatial dimension of the input, while in the decoder they increase it.

In the test phase, the autoencoder will not be able to perfectly reconstruct the potential anomalies, since it is trained only with normal images. For this reason, comparing the input image and the reconstructed one, it is possible to generate a pixel-wise anomaly map, using as anomaly metrics the function defined in (2.10). Finally, defining a threshold α, we can label all the pixels below α as normal (since the l^2 distance is low) and all the others as anomalous.

Improving Unsupervised Defect Segmentation by Applying Structural Similarity To Autoencoders

The main issue of the above Anomaly Detection approach is that the l^2 distance yields high residuals in locations where the reconstruction is only slightly inaccurate (e.g. due to small localization imprecisions of edges) and fails to detect structural differences between the input and reconstructed images, when the respective pixels' color values are roughly consistent. To alleviate the aforementioned problems, *Bergmann et al.* in [3] propose to measure the reconstruction accuracy using the structural similarity (SSIM) metrics [38].

The primary motivation in the development of the Structural Similarity (SSIM) index is to allow non-structural contrast and intensity changes, as well as small translations, rotations, and scaling changes, which are detectable but do not affect the perceived content of an image [41]. The main approach to accomplish this goal is to compare local image statistics in corresponding sliding windows (for example, 7×7) in the two images and to pool the results of such comparisons. The sliding window procedure follows a regular grid sampling strategy with stride 1. It is possible to define μ_x^i and σ_x^i as the mean and the variance values of image x at window position i. Then, σ_{xy}^i as the covariance between image x and y, at the same window location i.

To evaluate the similarity metrics, the system separates the task into three comparisons: *Luminance*, *Contrast* and *Structure*. Notice that the three components are relatively independent. The Luminance component is defined as:

$$l^i(x,y) = \frac{2\mu_x^i \mu_y^i + C_1}{(\mu_x^i)^2 + (\mu_y^i)^2 + C_1}, \tag{2.11}$$

where C_1 is a small constant to avoid instability when the denominator tends to zero. The Contrast comparison takes a similar form:

$$c^i(x,y) = \frac{2\sigma_x^i \sigma_y^i + C_2}{(\sigma_x^i)^2 + (\sigma_y^i)^2 + C_2}. \tag{2.12}$$

Thus, the Structure term is defined as:

$$s^i(x,y) = \frac{\sigma_{xy}^i + C_3}{\sigma_x^i \sigma_y^i + C_3}. \tag{2.13}$$

At this point, the SSIM index can be written as a function of the three previously presented terms:

$$\text{SSIM}^i(x,y) = [l^i(x,y)]^\alpha [c^i(x,y)]^\beta [s^i(x,y)]^\gamma. \tag{2.14}$$

That if we set $\alpha = \beta = \gamma = 1$, it results in a specific form of the SSIM index:

$$\text{SSIM}^i(x,y) = \frac{(2\mu_x^i \mu_y^i + C_1)(2\sigma_{x,y}^i + C_2)}{((\mu_x^i)^2 + (\mu_y^i)^2 + C_1)(\sigma_x^i)^2 + (\sigma_y^i)^2 + C_2}. \tag{2.15}$$

The final SSIM score is calculated averaging the contributions from all the window locations:

$$\text{SSIM}(x,y) = \frac{1}{N}\sum_{i=1}^{N} \text{SSIM}^i(x,y). \tag{2.16}$$

The index ranges between [-1, 1], where 1 indicates maximum similarity.

The training procedure proposed by *Bergmann et al.* in [3] requires in input patches coming from images that do not contain anomalies. The model is a classical convolutional autoencoder trained using SSIM as loss function between patches. This choice results in an improved reconstruction quality with respect to the MSE loss function in (2.10).

The anomaly score is calculated applying SSIM in the classical way, but then the scores across all the sliding windows are not averaged, but assigned to the center of the corresponding window. This results in an anomaly map in which pixels with higher structural dissimilarity are highlighted.

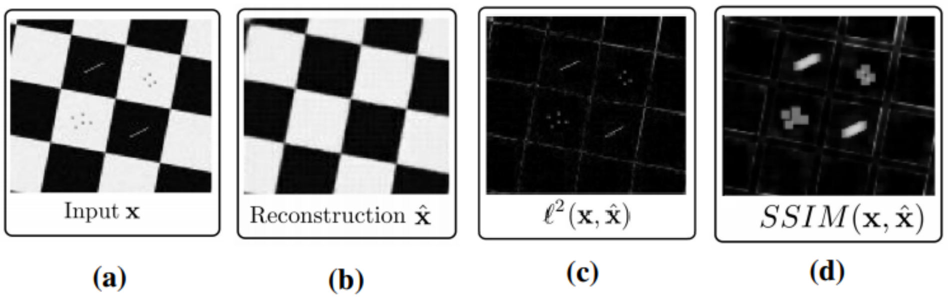
Input $\mathbf{x}$
Reconstruction $\hat{\mathbf{x}}$
$\ell^2(\mathbf{x}, \hat{\mathbf{x}})$
$SSIM(\mathbf{x}, \hat{\mathbf{x}})$
(a)
(b)
(c)
(d)

Figure 2.9: A toy example from [3], illustrating the advantages of SSIM over l^2 for the segmentation of defects. (a) 128 $\times$ 128 checkerboard pattern with gray strokes and dots that simulate defects. (b) Output reconstruction $\hat{x}$ of the input image x trained with an l^2 autoencoder. The defects have been removed by the autoencoder. (c) l^2 residual map. Brighter colors indicate larger dissimilarity between input and reconstruction. (d) SSIM residual map. In contrast to the l^2 error map, SSIM gives more importance to the main visual differences than to the slight inaccuracies around reconstructed edges.

2.2.4 Variational Autoencoder

A variational autoencoder (VAE) is a probabilistic model whose posterior distribution is approximated by a neural network, forming an autoencoder-like architecture [15]. Using the same notation of Section 2.2.3, the encoder $E(\cdot)$ is designed to reproduce in output data which belong to the variational approximate posterior $p_E(z|x)$, that can be defined as multivariate Gaussian with a diagonal covariance structure:

$$p_E(z|x) \sim \mathcal{N}(z; \mu, \sigma^2 I), \tag{2.17}$$

where μ_i and σ_i are the mean and the standard deviation of the outputs of the encoding function $E(x)$. This distribution will be trained to be close to the prior distribution defined as $\mathcal{N}(z; 0, I)$, which is the centered isotropic multivariate Gaussian, working as a regularization term.

The decoder $D(\cdot)$, instead, will produce in output data that belong to the distribution $p_D(x|z)$, namely the likelihood of the input data x, given the latent variable z sampled from the encoder distribution $z \sim p_E(z|x)$. The distribution $p_D(x|z)$ depends on the nature of the data. If the data are binary a Bernoulli distribution is used. If the data are in continuous form, a Multivariate Gaussian is a possible choice.

The loss function is defined as the Kullback-Leibler divergence (KL) [16] between the latent distribution and the prior, plus the variational lower-

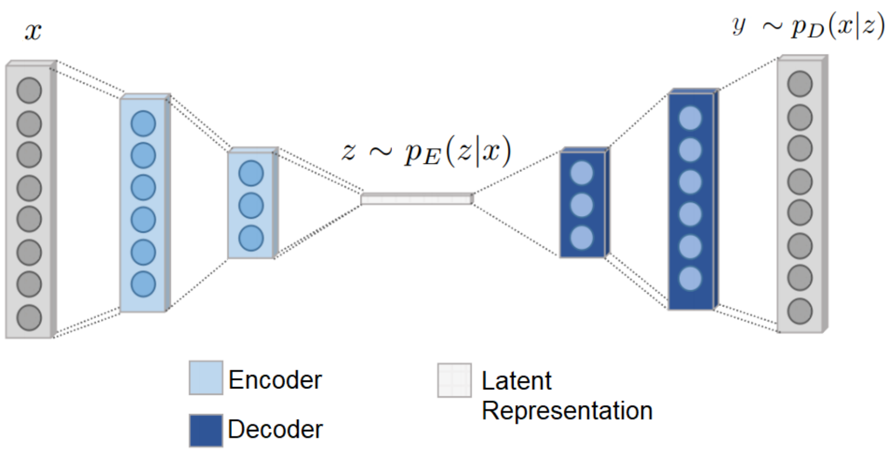

Figure 2.10: Variational Autoencoder architecture.

bound on the marginal likelihood with respect to x:

$$\mathcal{L}_{VAE}(x, y) = -KL(p_E(z|x), \mathcal{N}(z; 0, I)) + \mathbb{E}_{p_E(z|x)}[\log(p_D(x, z))]. \quad (2.18)$$

The first term of (2.18) is the KL divergence between the approximate posterior and the prior of the latent variable z. This term forces the posterior distribution to be similar to the prior one, working as a regularization term. The second term of (2.18) can be rewritten as:

$$\mathbb{E}_{p_E(z|x)}[log(p_D(x, z)) = \frac{1}{L}\sum_{l=1}^{L} log(p_D(x|z_l)), \quad (2.19)$$

where L samples from the latent distribution are drawn to evaluate the likelihood distribution. After the training phase, the VAE can be employed to compute an anomaly score in two different ways:

- **Using the Latent Space Representation.** One option is to compute the KL divergence $KL(p_e(z|x), \mathcal{N}(z; 0, I))$, and to indicate defects for large deviations from the prior distribution [32]. However, to use this approach for the pixel-accurate segmentation of anomalies, a separate upsampling for each pixel of the input image has to be performed.

- **Using the Reconstruction Probability.** *Choi et al.* [1] propose a different approach: for each sample from the encoder z_l, the mean μ_l^y

and the standard deviation σ_l^y of the decoder output are calculated in order to build the likelihood probability:

$$p_D(x|z_l) = p_D(x|\mu_l^y, \sigma_l^y). \tag{2.20}$$

Then, these likelihoods are averaged over all the L samples. This pixel-wise probability is then used to determine weather a pixel belongs to the normal class or not by a threshold.

2.2.5 Generative Adversarial Network

Generative Adversarial Networks (GANs) are Convolutional Neural Networks able to generate new unseen data belonging to the same unknown distribution of the training data [12]. They are composed by two sub-networks: the generator $G()$ is trained to learn the input distribution, while the discriminator $D(\cdot)$ learns how to distinguish data coming from $G()$ from data belonging to the input dataset. We remark that the generator has the same structure of the encoder $E(\cdot)$ presented in the two previous sections.

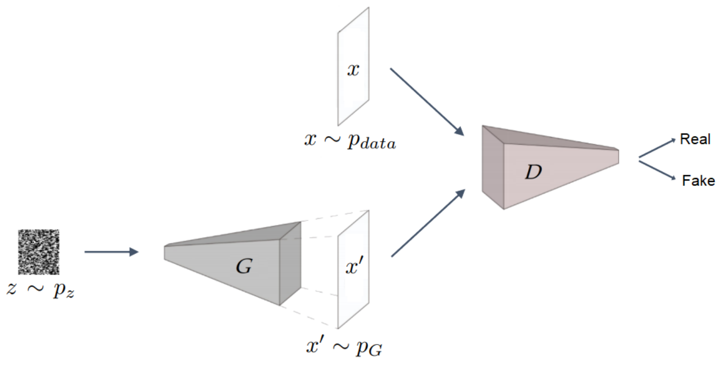

Figure 2.11: Generative Adversarial Netowrk architecture.

To learn the generator's distribution $x' \sim p_G$ over input data $x \sim p_{data}$, a prior distribution on input noise variables $z \sim p_z$ (white noise) is defined. The generator $G(z)$ is used to map the input space to the data one. $D(x)$ represents the probability that x came from the data distribution rather than p_G. $D(\cdot)$ is trained to maximize the probability of assigning the correct label to both training examples x and samples from $G(z)$. Simultaneously, $G()$ is trained to minimize $log(1 - D(G(z)))$. $D(\cdot)$ and $G()$ play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}}[log(D(x))] + \mathbb{E}_{z \sim p_z}[1 - log(D(G(z)))]. \tag{2.21}$$

Bringing it to the image world, the idea is that while the generator attempts to produce realistic images in order to fool the discriminator, the latter learns how to distinguish images generated from realistic ones. The term Adversarial in GAN comes exactly from the training procedure: the two networks compete in this setting having two opposite objectives.

Unsupervised Anomaly Detection with Generative Adversarial Networks

As an Anomaly Detection problem, the goal is to produce an anomaly score of test data over a pre-trained architecture. *Schlegl et al.* propose *AnoGAN* [30], a GAN network which is trained only with normal images. The authors define a loss function $\mathcal{L}$ for the mapping of new images to the latent space that comprises two components, a *residual loss* $\mathcal{L}_R$ and a *discrimination loss* $\mathcal{L}_D$:

$$\begin{aligned}\mathcal{L}_D &= \lambda \left| f(x) - f(G(z)) \right|, \\ \mathcal{L}_R &= \left| x - G(z) \right|, \\ \mathcal{L}(z) &= (1-\lambda)\mathcal{L}_R(z) + \lambda \mathcal{L}_D(z).\end{aligned} \tag{2.22}$$

The residual loss enforces the visual similarity between the generated image $G(z)$ and the query image x. The discrimination loss, instead, constrain the generated image $G(z)$ to follow the learned distribution of input data, where $f(x)$ is the output of an intermediate layer of the discriminator. Therefore, both components of the trained GAN are used to adapt the network's weights via backpropagation. The summation in the loss formulae means that the score over all the pixels has been summed. The discriminator loss is inspired by the feature matching technique. Instead of optimizing the parameters of the generator by maximizing the discriminator's output on generated examples as in (2.21), the generator is forced to produce data similar to the training one, i.e. whose intermediate feature representation is similar to those of real images. This way, feature matching addresses the instability of GANs due to over-training discriminator response. The anomaly map over test data will be calculated as:

$$S(x,z) = (1-\lambda) \left| x - G(z) \right| + \lambda \left| f(x) - f(G(z)) \right|. \tag{2.23}$$

Chapter 3

Background

In this chapter, we introduce some important concepts that will be useful in the description of our methods. In Section 3.1, we present the *Steerable Filters*, which we employ in our solution to extract interesting features from the input images. Then, in Section 3.2, we present the *Multi Scale Structural Similarity* (MS-SSIM), an improved version of the Structural Texture Similarity (presented in Section 2.2.3), which we use as a baseline method in our experiments. Finally, in Sections 3.3 and 3.4, we describe in detail an anomaly metrics named *Mahalanobis Distance* and an Anomaly Detection algorithm called *Kernel Density Estimation*, which we will use in our first solution based on STSIM as feature extraction algorithm.

3.1 Complex Steerable Filters

Complex Steerable filters are based upon the *Discrete Wavelets Transform*, which is able to cut up data into different frequency components, and then studies each component with a resolution matched to its scale [34]. In two dimensions, this transform has been proven to be able to represent the image with a set of basis functions which are related to translation and dilation operations applied to a common kernel [34].

Freeman and Adelson [10] develop a technique to synthesize kernels of arbitrary orientation from a linear combination of a fixed set of kernels (Steerable Basis), describing the conditions under which a set of rotated basis functions is steerable. *Wang et al.* [31] also design a two dimensional transform that is jointly shiftable in orientation (multi-orientation property) and position (multi-scale property), where the basis functions are translations, dilations, and rotations of a single kernel. The transform is then constructed as a recursive pyramid called *Steerable Pyramid Decomposition* [31]. A pyramid in Computer Vision is composed by the repetition of the same original image with a decreasing spatial resolution at each level of the pyramid. Figure 3.1 illustrates an example of the orientation-shiftable basis functions at one scale of the pyramid.

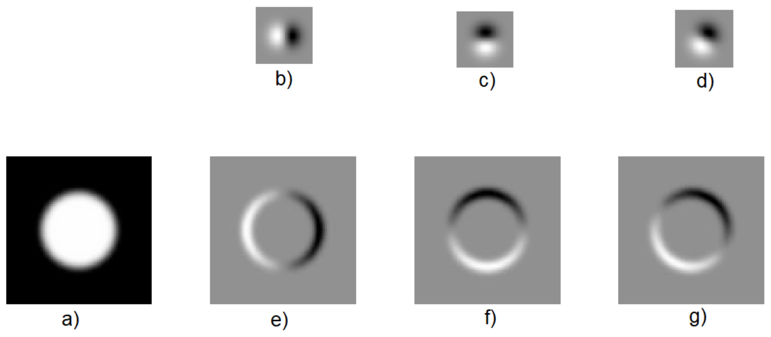

Figure 3.1: Example of Steerable filters applied to an image at a single scale of the pyramid. a) Original image. b)-c)-d) are the Steerable oriented kernels at orientation 0°,90° and 30° respectively. e)-f)-g) are the subband results coming from the convolution between the original image and the kernels respectively. This image is from [10].

The Steerable pyramid algorithm is based on recursive application of filtering and subsampling operations. The input image is initially convolved with a band-pass filter $B(w)$ which, varying the frequency parameters w, applies a number of orientated convolutions at each stage of the pyramid. Then, before subsampling the image, a low-pass filter $L_1(w)$ is applied to remove the most aliased components, which are high frequency, i.e. noise, in the subsampled domain. The subsampled image iteratively passes through the above procedure until the last scale of the pyramid, where another low-pass filter $L_0(w)$ is applied. The last scale of the pyramid is not convolved with oriented filters. The output comprehends also the input image convolved with a high pass filter $H(w)$. The block diagram defining this procedure can be found in Figure 3.2.

This procedure decomposes the input image into M different subbands. Each subband represents the response of the convolutions between the original image and O oriented kernels. These convolutions are repeated for the number of scales defined by S, where for each scale the image resolution

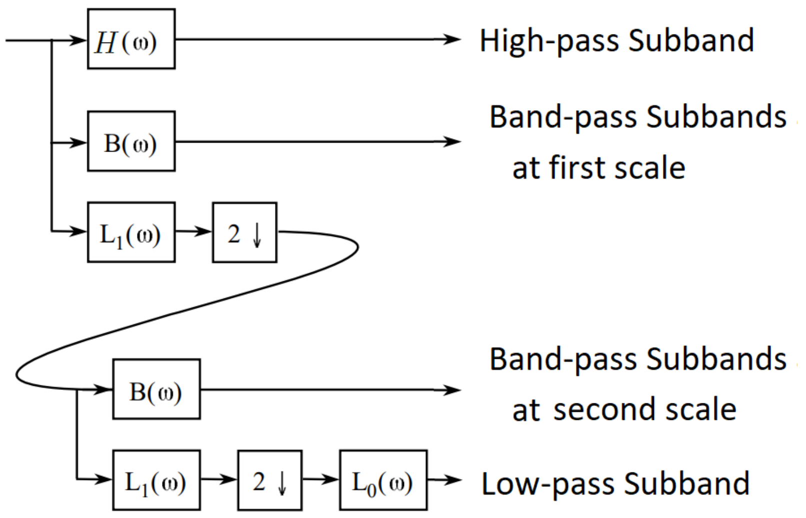

Figure 3.2: Illustration of a two-stage recursive cascade. This figure is inspired from [31].

reduces by a factor of 2. In the end we have

$$M = O(S-2) + 2 \tag{3.1}$$

subbands, where the terms "2" indicates that on the first and last scales, which represent respectively the outputs of $H(w)$ and $L_0(w)$, the oriented convolutions are not applied. The design of the filters and more implementation details can be found in [31] and [10]. An example of Steerable filters decomposition with 5 scales 4 orientations can be found in Figure 3.3.

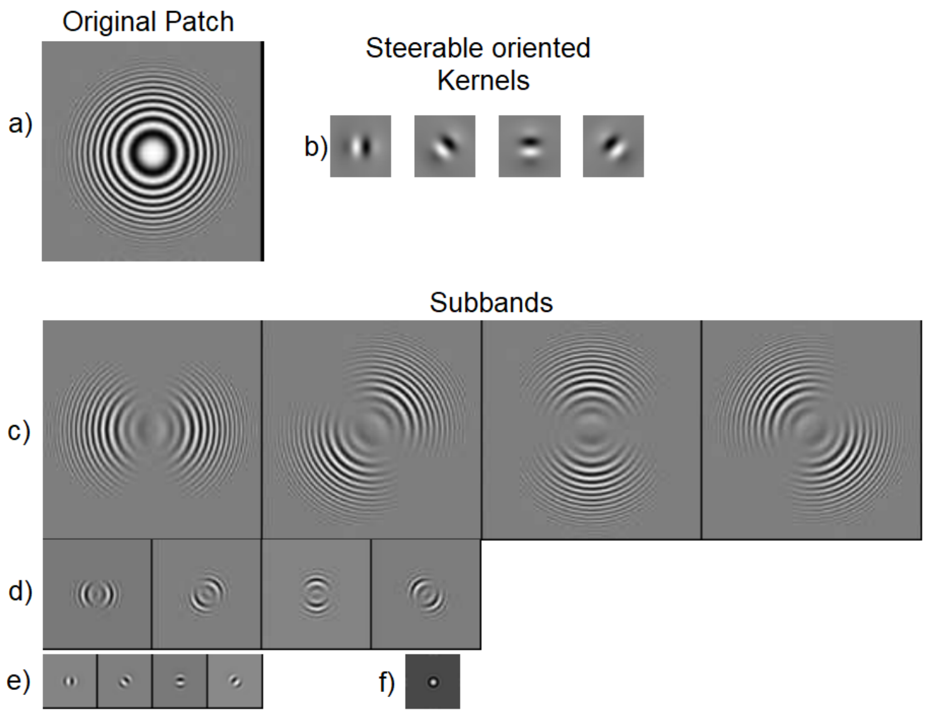

Figure 3.3: Example of 5 scales 4 orientations steerable filters decomposition. a) Original patch. b) Steerable oriented kernels at 0°, 45°, 90° and 130°. c)-e) Band-pass coefficients in a multi-scale pyramid representation. (f) Low-pass image. This image is from [10].

3.2 Multi-Scale Structural Similarity

The Multi-Scale Structural Similarity (MS-SSIM) [37] was introduced as an extension of SSIM [38]. The subjective evaluation of a given image and the perceivability of details depend on many factors: the image signal, the distance from the image plane to the observer and the perceptual capability of the observer. Because of that, a multi-scale approach is more convenient when we want to incorporate image details at different resolutions [37]. MS-SSIM takes two images in input, iteratively applies a low-pass filter and downsamples the filtered images by a factor of 2. The scheme of the algorithm can be found in Figure 3.4.

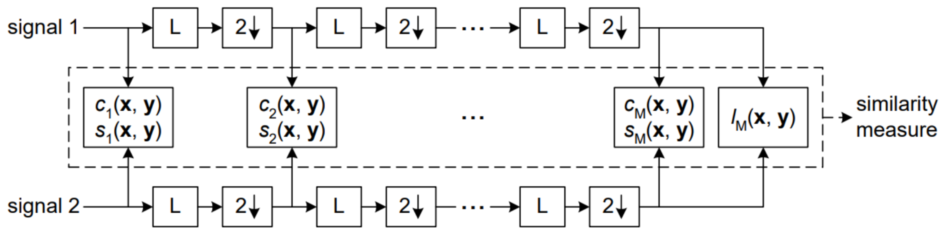

Figure 3.4: Multi-scale structural similarity measurement system. L: low-pass filtering; 2 ↓: downsampling by 2. This image is from [37].

This procedure generates two pyramids, one for each image, where at each level the original image is halved in the spatial dimension. In these pyramids, the original image is at scale one, while the last one is at scale M. For each level of the pyramid, are computed the contrast term $c_j(x, y)$ (2.12) and structure term $s_j(x, y)$ (2.13), where j indicates that the comparison is performed at level j, while the luminance term $l_M(x, y)$ (2.11) is calculated only at scale M. The overall MS-SSIM evaluation is obtained by combining the measurement at different scales as follows:

$$\text{MS-SSIM}(x, y) = [l_M(x, y)]^{\alpha_M} \sum_{j=1}^{M} [c_j(x, y)]^{\beta_j} [s_j(x, y)]^{\gamma_j}, \tag{3.2}$$

where the exponents $\alpha_M, \beta_j, \gamma_j$ are used to adjust the relative importance of different components, similarly to (2.14). In the performance evaluation Section 6.4, where we will present our experimental results using this metrics as loss function, we will refer to the coefficient configuration provided by [37]. Both MS-SSIM and SSIM work with local statistics and range between [-1, 1], where 1 indicates maximum similarity.

3.3 Mahalanobis Distance

The Mahalanobis distance is a distance-based metrics introduced by P.C. Mahalanobis [18]. It is a scale-invariant metrics and provides a measure of distance, also referred as likelihood, between a point $x \in \mathbb{R}^p$ and a $p-$variate probability distribution function (pdf) $f(X)$ with mean $\mu = \mathbb{E}(X)$ and covariance matrix $\Sigma = \mathrm{Cov}(X)$ of the distribution. The Mahalanobis distance is defined by:

$$M(x, \mu, \sigma) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}. \tag{3.3}$$

If the covariance matrix is the identity matrix, the Mahalanobis distance reduces to the Euclidean distance. All values of x such that $M(x, \mu) = c$, for any specified constant value c, have equal likelihood. As we can notice from Figure 3.5, the level sets produced by this metrics are ellipsoids. We consider x an outlier with respect to f if it belongs to a region outside the ellipsoid defined by a certain threshold η, i.e., $M(x, \mu) > \eta$.

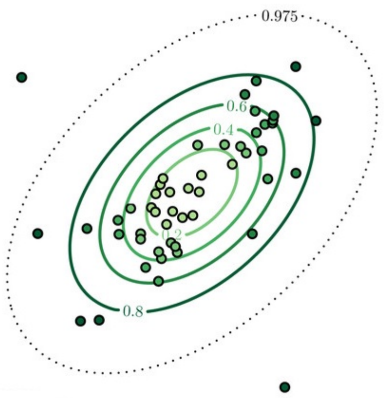

Figure 3.5: Mahalanobis distance 2D representation at $c = 0.4$, 0.6, 0.8 *and* 0.975.

3.4 Kernel Density Estimation

The Kernel Density Estimation (KDE) is a kernel-based method that produces a smooth estimation of the probability density function (pdf) of the input data $\{x_1, ..., x_n\}$ [23]. The goal is to learn the parameters θ to model the pdf of the normal data $f_\theta(X)$ using a kernel function K. The formal definition of KDE is:

$$\hat{f}_\theta(x) = \frac{1}{nb} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{3.4}$$

where $\hat{f}_\theta$ is the kernel density estimator of the function f_θ. $K(x)$ is the kernel function, that is generally a smooth symmetric function, such as a Gaussian, and $b > 0$ is called the smoothing bandwidth that controls the amount of smoothing. Basically, the KDE smoothes each data point x_i into a small density bump and then sums all these small bumps together to obtain the final density estimate [23], as we can see in Figure 3.6.

The anomaly score is defined as the log probability of having an incoming data y belonging to the distribution f_θ, i.e., the log-likelihood of y with respect to the estimator $\hat{f}_\theta$:

$$S = \log(\hat{f}_\theta(y)) = \log(\hat{f}(y|\theta)). \tag{3.5}$$

The more the likelihood is towards zero, the more y belongs to the same distribution of $f_\theta(X)$.

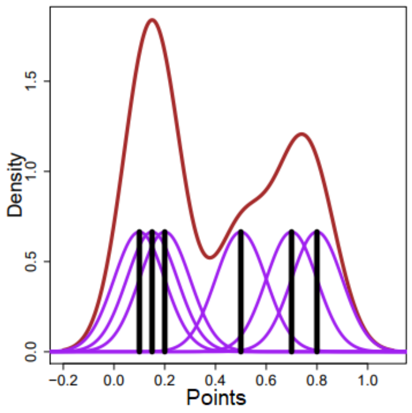

Figure 3.6: KDE first smoothes each data point (at 0.1, 0.2, 0.5, 0.7, 0.8, 0.15) into a purple density bump and then sums them up to obtain the final density estimate – the brown density curve. The image is from [23].

Chapter 4

Our Contribution

In this chapter, we present two novel methods in the Semi-Supervised Anomaly Detection framework. In particular, both of them are based on the Steerable filters decomposition that is used to extract interesting features from textured images. The first method, named *Anomaly Detection based on STSIM*, follows the Hand-Crafted Feature-based approach, where the Steerable filters are directly used over normal training patches to extract handcrafted features. The feature extraction algorithm is based on the Structural Texture Similarity Metrics (STSIM) [41], which is a widely used metrics for content-based retrieval. The idea is that all the normal features are distributed in a particular way with respect to the feature space. The goal is to learn their posterior distribution, and detect as anomalous the patches that fall in low density regions.

The second method, named *Anomaly Detection based on CW-SSIM applied to Autencoders*, follows the Data-Driven approach, where the features to extract are learned through the use of an autoencoder. The novelty is that we propose *Complex Wavelet Structural Similarity* (CW-SSIM) [29] as loss function, which is based on the Steerable filters. We remark that this similarity metrics was never used as loss function before. The rationale behind this procedure is that the autoencoder during the training learns the features that characterize normal patches, and thus will not be able to perfectly generate anomalies in evaluation. The reconstruction based Anomaly Detection takes advantage of this fact calculating the structural similarity, using CW-SSIM, between the reconstructed and the original image, as in [3]. The locations where the autoencoder fails to reconstruct the anomalies should present higher dissimilarity with respect to other regions.

The first Section 4.1 proposes again the problem formulation reported in the Introduction, focusing also into the requirements of our solutions. The second Section 4.2 introduces the general outline of the patch-wise methodology, which is used by our two methods and also commonly deployed in this framework. Finally, in Sections 4.3 and 4.4 we explain in details the two proposed strategies, based on STSIM and CW-SSIM respectively.

4.1 Problem Formulation and Requirements

Let I be an image, i.e. a matrix, of size $w \times h \times c$ of values $I(i,j) \in \mathbb{N}$, where h defines the height, w the width and c represents the number of color channels. The latter in our case is set to one, since we are dealing only with gray scale images. Each element, or pixel, of the image $I(i,j)$ at position (i,j) is a single value that can range from 0 to $2^r - 1$, where r is the color depth (usually referred to as bit depth). For each image I, we define the binary mask of anomalies Ω_I as a matrix of size $w \times h$ such that:

$$\Omega_I(i,j) = \begin{cases} 1, & \text{if } I(i,j) \text{ is an anomalous element of image I,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

Given an image I, the Anomaly Detection problem requests to automatically find the binary mask $\hat{\Omega}_I$ that best approximates the reference mask of anomalies Ω_I, also referred to as the ground truth. Our goals are: (a) to identify all the anomalous regions in the image I; (b) to cover the largest number of anomaly pixels that belong to the identified anomalous regions; (c) to label as normal all the other pixels that do not present anomalous patterns.

4.2 Outline of our patch-wise solutions

The proposed solutions belong to the Semi-Supervised Anomaly Detection framework, in which only normal images are used during the training stage. The patches forming the training set $T = \{x_i \in \mathbb{R}^{h_t,w_t} | i \in \{1,...,N_t\}\}$ are extracted from normal images at random positions in such a way that they have the same spatial dimension. We can define h_t and w_t as the height and the width of the training patches. For simplicity, we use only squared patches for both the methods. The training phase ends, producing a model able to statistically distinguish a normal patch from an anomalous one. For each test image I belonging to the test set, the patches in $E_I = \{x_j \in \mathbb{R}^{h_t,w_w} | i \in \{1,...,N_e\}\}$ are extracted following a regular grid sampling strategy with stride s. Each of them is firstly evaluated, computing its degree of abnormality, and then each anomaly score is combined to create the anomaly map ω_I. The output of both the strategies is the pixel-wise classification of test images, also referred as the estimated binary mask of anomalies $\hat{\Omega}_I$, generated applying the threshold γ to the anomaly map.

4.3 Anomaly Detection method based on STSIM

We propose the use of Structural Texture Similarity Metrics (STSIM) [41], as a feature extraction technique that relies upon Steerable filters decomposition. STSIM was initially designed by *Pappas et al.* for image analysis and content-based retrieval (CBR), in particular on the recovery of textures that are identical to a query one, in the sense that they could be patches from a large perceptually uniform texture even if they present important pixel-wise differences. An example can be found in Figure 4.1. This approach is particularly effective when we are dealing with textured images, as the one in Figure 4.1 or the nanofibrous dataset [19], which we will use to evaluate our performances.

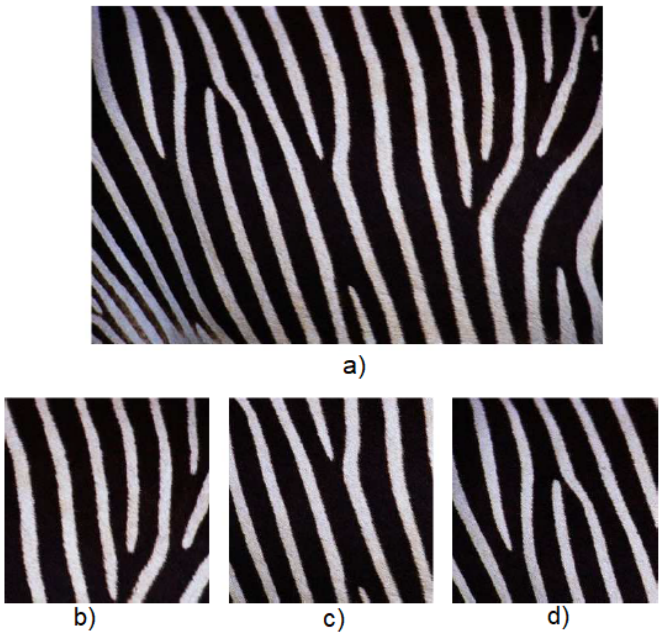

Figure 4.1: Example of textured patches b)-c)-d) which are perceptually identical to the query image a). This image is from [41].

The features extracted with STSIM, belonging to normal patches, are consequently used to train an Anomaly Detection model. In particular, we propose the use of the Mahalanobis distance and the Kernel Density Estimation. These metrics are able to tell, given a test patch, if it belongs to the same distribution of the training set, i.e. if they are perceptually identical (same texture images). Once the anomaly score is produced for each test patch, they are upscaled and merged to create the anomaly map ω_I.

4.3.1 Preprocessing

The training set is filled with normal patches only if their median is above a certain value $\epsilon > 0$. This is done to avoid that entire dark patches pollute the training set, since they could decrease the effectiveness of recognizing normal textures.

$$T = \{x_i | median(x_i) > \epsilon\}. \tag{4.2}$$

It is worth mentioning that nanofibrous materials, which have too large holes, might yield porosity defects. However, this sort of anomalies can be detected by straightforward morphological operations over the whole image [7].

4.3.2 Subband Decomposition

The first step is to apply Steerable filters decomposition to each training patch $x_i \in T$. Steerable filters are directional derivative operations, which can vary in scale and orientation, in a way to provide multi-scale and multi-orientation analysis. The motivation comes from the work of *Portilla and Simoncelli* on texture analysis/synthesis [24], who show that a broad class of textures can be synthesized using a set of statistics that characterize the coefficients of a multi-scale frequency decomposition (Steerable filters).

This procedure decomposes each patch x_i into M different subbands. Each patch is initially brought to the frequency domain through the *Discrete Fast Fourier Transform* (FFT). At each scale of the pyramid, a sequence of filters in the frequency domain is convolved with the original image in order to produce M Steerable subbands x_i^m with $m = \{1, ..., M\}$. Finally, each subband is taken back to the image domain with the *Inverse Discrete Fast Fourier Transform* (IFFT). We recall that at each scale the subband is a complex matrix, where its spatial dimension reduces by a factor of 2 each time the subsampling operation is applied. The subbands are defined as $x_i^m \in \mathbb{C}^{w_s,h_s}$, where $w_s = h_s$ (in our case) and $m = \{1, ..., M\}$. This procedure is fully described in the Background chapter, in Section 3.1.

The number M is a manually tuned parameter that depends on many factors. We recall from (3.1) that the parameter M depends on the number of scales S, and on the number of oriented filters O. The parameter S is intrinsically connected to the patch dimension, since it determines the number of subsampling operation that the pyramid will produce. In principle, the upper bound in the number of scales is:

$$S \leq \log_2(w_t), \tag{4.3}$$

since going further means to reduce the subband dimension to a single value. This limit, in practice, is more restrictive since, as we have discussed before, too small patches (and so also subbands) would not lead to capture the features that characterize normal textures. Regarding the parameter O, there is not a particular limitation, but it can be tuned according to the characteristics of the patches which we are dealing with. Focusing on the nanofibrous texture, we can notice that normal patches are characterized by high differences in gradient in almost any direction. An example can be found in Figure 4.2. This fact is due to the presence of the filamentous elements that produce high variations with respect to the dark background. The band-pass convolutions are exactly deployed to capture these types of changes in all the O directions. Anomalous patterns, instead, are characterized by uniform agglomerates that have almost the same response in any direction. We can assume that, even applying a few number of oriented convolutions, these differences can be captured by the Steerable filters. An example of the Steerable filters decomposition applied to the nanofibrous dataset can be found in Figure 4.2. All the images representing the nanofibrous dataset are in gray-scale, but we decide to show them with different colors (the Virdis[1] heatmap) based on their gray level. This choice is made to enable a better interpretation by the viewer and will be maintained in the following pictures regarding the examples from this dataset.

4.3.3 STSIM Feature Extraction

This step allows to effectively generate the set of features that we will use to distinguish normal patches from anomalous ones. These features are based

[1]https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

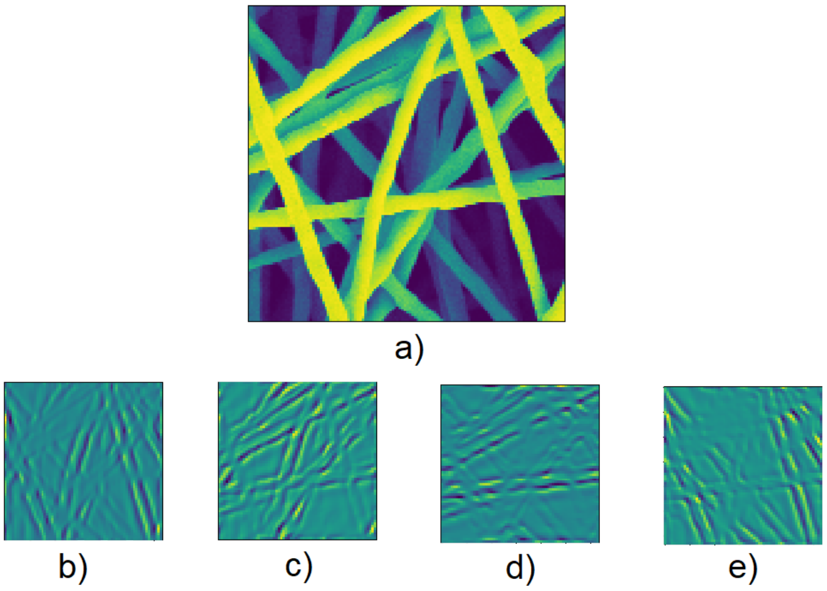
Figure 4.2: Steerable filters decomposition at one scale of the pyramid. a) Original patch. b)-e) Steerable subbands at second scale of the pyramid with 4 orientations.

on statistical measurements from each subband and among different ones. The first step to apply is to bring all the subband coefficients to the real domain, calculating the modulus:

$$|x_i^m| = \sqrt{(\mathcal{R}(x_i^m))^2 + (\mathcal{I}(x_i^m))^2}, \tag{4.4}$$

where $\mathcal{R}$ and $\mathcal{I}$ represent the real and the imaginary part of x_i^m respectively. This step is applied because we need a real statistical value $|x_i^m| \in \mathbb{R}^{w_s,h_s}$.

This procedure extracts 4 single-band statistics from each subband x_i^m. The first two statistics are the mean value:

$$\mu_{|x|}^m = \mathbb{E}\left[|x^m(i,j)|\right], \tag{4.5}$$

and the variance:

$$(\sigma_{|x|}^m)^2 = \mathbb{E}\left[\left(|x^m(i,j)| - \mu_{|x|}^m\right)\left(|x^m(i,j)| - \mu_{|x|}^m\right)^*\right], \tag{4.6}$$

where (i,j) represents the indexing of each single pixel. Then, the first-order autocorrelation coefficients can be in the horizontal direction as:

$$\rho_{|x|}^m(0,1) = \frac{\mathbb{E}\left[\left(|x^m(i,j)| - \mu_{|x|}^m\right)\left(|x^m(i,j+1)| - \mu_{|x|}^m\right)^*\right]}{(\sigma_{|x|}^m)^2}, \tag{4.7}$$

and in the vertical direction as:

$$\rho^m_{|x|}(1,0) = \frac{\mathbb{E}\left[\left(|x^m(i,j)| - \mu^m_{|x|}\right)\left(|x^m(i+1,j)| - \mu^m_{|x|}\right)^*\right]}{(\sigma^m_{|x|})^2}. \tag{4.8}$$

The first-order autocorrelations of adjacent subband coefficients provide structural and directional information. In particular, these terms are achieved by calculating the cross-correlation between the same subband shifted of one pixel in the horizontal or vertical direction. We specify that there is no need to consider higher order autocorrelations, because this would be equivalent to compute first-order autocorrelations of subsamppled images. This operation is effectively done when we compute the first-order autocorrelations of lower frequency subbands. Thus, by computing (4.7) and (4.8) on a multi-scale frequency decomposition, we are effectively computing higher-order autocorrelations [41].

The last step extracts $O(S-3)$ cross-band correlations between subbands at adjacent scales for a given orientation, and $(S-2)\binom{O}{2}$ correlations between all orientations for a given scale:

$$\rho^m_{|x|}(0,0) = \frac{\mathbb{E}\left[\left(|x^m(i,j)| - \mu^m_{|x|}\right)\left(|x^n(i,j)| - \mu^n_{|x|}\right)\right]}{(\sigma^m_{|x|})(\sigma^n_{|x|})}, \tag{4.9}$$

where m and n are the two subbands involved in the calculation. These features are used because the magnitudes of the wavelet coefficients are not statistically independent: large magnitudes in subbands of natural images tend to occur at the same spatial locations in subbands at adjacent scales and orientations. The intuitive explanation that [41] gives for this fact is that the "visual" features of natural images rise to large local neighborhood spatial correlations, as well as large scale and orientation correlations. We remark that since the first and the last layer of the pyramid are not convolved with the oriented filters, this correlation cannot be applied to them.

In the end, the total number of single-band statistics is:

$$N_i = 4(O(S-2)+2) = 4M, \tag{4.10}$$

while the number of cross-band statistics is:

$$N_c = (S-2)\binom{O}{2} + O(S-3), \tag{4.11}$$

for a total of N_f features:

$$N_f = N_i + N_c. \tag{4.12}$$

At this point, the procedure ends since we have extracted from each training patch $x_i \in T$ the corresponding features vector $f_i \in \mathbb{R}^{N_f}$. The full procedure can be found in Algorithm 1.

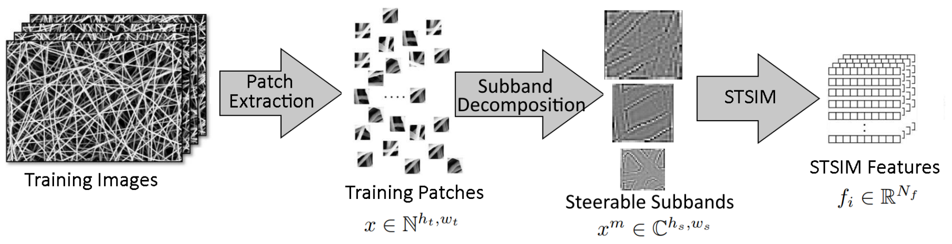

Figure 4.3: Complete feature extraction procedure. The template schema is taken from [20].

```
Algorithm 1: ST-SIM Feature Extraction
  Input: Set of normal images.
  Output: Set of features vectors $F$.
    1. Extract patches randomly from the training images: $x_i$.
    2. Apply the preprocessing step (4.2).
    3. Decompose each patch into the corresponding Steerable subbands $x_i^m$
       (as in Section 3.1).
    4. Get the modulus of each subband $|x_i^m|$ (4.4).
    5. Extract the mean $\mu^m_{|x_i|}$ (4.5), the variance $(\sigma^m_{|x_i|})^2$ (4.6), the horizontal
       autocorrelation $\rho^m_{|x_i|}(0,1)$ (4.7), the vertical autocorrelation $\rho^m_{|x_i|}(1,0)$
       (4.8) and the cross-band correlation $\rho^m_{|x|}(0,0)$ (4.9).
    6. Pack all the statistical features in the STSIM features vector $f_i$.
```

4.3.4 Learning Normal Features' Distribution

Once all the features vectors of the training patches $f_i \in \mathbb{R}^{N_f}$ with $i \in \{1, ..., N_t\}$ are extracted, we need to learn the parameters of a density model that characterize their distribution. The idea is that all the normal patches, even in the test set, belong to the same posterior distribution in the feature space, while anomalous ones have lower density with respect to this normal distribution. The training procedure has to learn a model which represents the distribution of normal features. In our solution, we propose two different Anomaly Detection models: the first based on Mahalanobis Distance and the second on Kernel Density Estimation.

Before effectively learning these parameters, we need to split the training features into two sets: we can call the first one T_f, which is effectively used to train the Anomaly Detection algorithms, and V_f, which is used to calculate empirically the mean density of normal patches μ_v over the learned distribution. This value will be used as the anomaly score on too dark test patches.

Mahalanobis Distance is a distance-based method that, in our case, provides a measure of distance between a single features vector $f_i \in \mathbb{R}^{N_f}$ and the mean $\mu_t \in \mathbb{R}^{N_f}$ of the normal distribution of features. This is equivalent to considering as anomalous any features vector having an indicator falling outside a confidence region around μ_t, defined by the Chebyshev inequality.

The Mahalanobis model is composed by two parameters: the mean features vector $\mu_t \in \mathbb{R}^{N_f}$, and the covariance matrix $\Sigma_t \in \mathbb{R}^{N_f,N_f}$ over the training set T_f. We calculate the mean features vector as:

$$\mu_t = \mathbb{E}\left[T_f\right]. \tag{4.13}$$

The covariance matrix, instead, is defined as:

$$\Sigma_t = \mathbb{E}\left[(T_f - \mu_t)(T_f - \mu_t)^T\right], \tag{4.14}$$

where the symbol T over the second term indicates the transpose of a matrix. Finally, the mean density of normal patches μ_v is calculated averaging the Mahalanobis distance (4.17) over the set V_f:

$$\mu_v = \mathbb{E}\left[\sqrt{(V_f - \mu_t)^T \Sigma_t^{-1} (V_f - \mu_t)}\right]. \tag{4.15}$$

Kernel Density Estimation is a kernel-based method that produces a smooth estimation of the input data's probability density function (pdf). The goal is to learn the model characterized by the parameters θ of the distribution ϕ_θ, which the training features vectors belong to: $f_i \sim \phi_\theta : \forall x_i \in T_f$. In our experiments, we model the density estimate (3.4) using a Gaussian Kernel with bandwith b equal to 0,5. This means that the parameters θ of the learned model represent a Gaussian distribution in the feature space.

The last step is to calculate the mean density of normal patches μ_v, by averaging the log likelihood (4.18) over the set V_f:

$$\mu_v = \mathbb{E}\left[\log(\phi_\theta(V_f))\right]. \tag{4.16}$$

The full training procedure can be found in Algorithm 2.

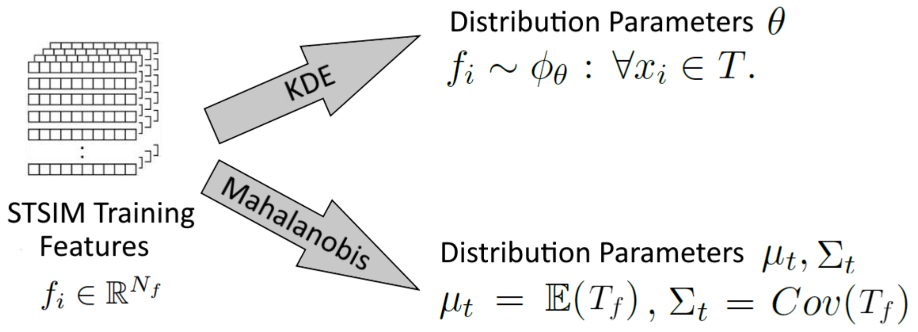
Distribution Parameters θ
$f_i \sim \phi_\theta : \forall x_i \in T.$
KDE
STSIM Training Features
$f_i \in \mathbb{R}^{N_f}$
Mahalanobis
Distribution Parameters μ_t, Σ_t
$\mu_t = \mathbb{E}(T_f), \Sigma_t = Cov(T_f)$

Figure 4.4: Normal distribution learning procedure. For each Anomaly Detection model, we show the parameters learned.

```
Algorithm 2: Normal Distribution Learning
  Input: Set of features vectors $F$.
  Output: Normal distribution parameters.
    1. Split the feature set $F$ in $T_f$ and $V_f$.
    2. If algorithm = Mahalanobis
           Calculate the distribution parameters $\mu_t$ (4.13) and $\Sigma_t$ (4.14).
           Calculate the mean distance of normal features $\mu_f$ (4.15).
    3. If algorithm = KDE
           Estimate the distribution parameters $\theta$.
           Calculate the mean density of normal features $\mu_f$ (4.16).
```

4.3.5 Distance-based Anomaly Detection

The rationale behind the proposed method is that, in order to detect anomalies within an image, we have to estimate how much a subregion of the image is far from being normal. We can split the test dataset in two sets: the validation set, used to decide the threshold value γ, and the test set, upon which the evaluation metrics are calculated. We postpone the discussion about the threshold value in a separate section (see Section 6.2), since it refers also to the other proposed method, based on CW-SSIM.

All the test patches $x_j \in E_I$, which respect the same brightness condition of the training patches (4.2), are passed through the same feature extraction procedure, explained in Section 4.3.3. Then, their anomaly score S_j is calculated depending on the Anomaly Detection procedure chosen. For the **Mahalanobis Distance**:

$$S_J = \sqrt{(f_j - \mu_t)^T \Sigma_t^{-1} (f_j - \mu_t)}; \tag{4.17}$$

and for the **Kernel Density Estimation** as the log-likelihood of the test data:

$$S_J = \log(\phi_\theta(f_j)). \tag{4.18}$$

For the patches which do not respect the brightness condition, the assigned anomaly score S_j is the mean density of the normal patches μ_v. An example of the score provided by KDE can be found in Figure 4.5.

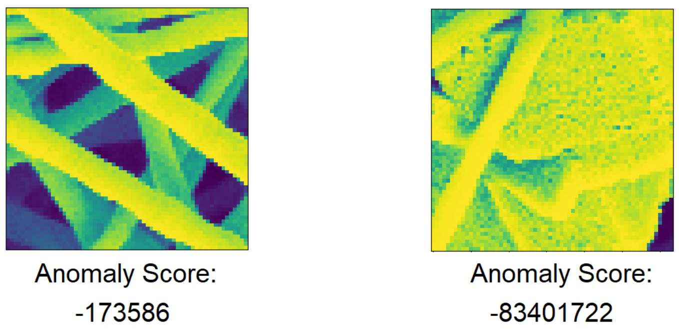

Figure 4.5: Score provided by the KDE: the more it is near to zero the more the input belongs to the normal distribution.

Once the corresponding anomaly score S_j is calculated for all the patches x_j, the goal is to generate the anomaly map ω_i. Each anomaly score is first upsampled to the patch dimension, then the degree of abnormality of each pixel is obtained by averaging the anomaly score of each corresponding subregion. The stride upon which the anomaly map is built is s, the same used during patch extraction. An example of this can be found in Figure 4.6. We remark that in this method it is fundamental to average the anomaly score coming from overlapping subregions, in order to produce high resolution anomaly maps. The smaller the stride is, the more the evaluated patches are, and so the higher the anomaly resolution but also the evaluation time.

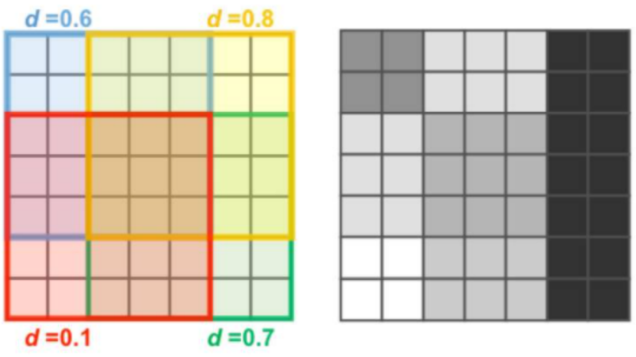

Figure 4.6: Simulated example of the anomaly map generated by 4 upsampled anomaly scores d. In this example the patch size is 5×5 with stride $s = 3$. This image is from [20].

The last step is to produce the binary mask of anomalies, which can be achieved by applying the threshold γ to each pixel in the anomaly map:

$$\hat{\Omega}_I(i,j) = \begin{cases} 1, & \text{if } \omega_I(i,j) \geq \gamma, \\ 0, & \text{otherwise.} \end{cases} \tag{4.19}$$

Since the KDE estimation outputs a density metrics, the sign in the above equation (4.19) is inverted. For the Mahalanobis procedure, instead, it is correct since it provides a distance metrics. The complete evaluation procedure can be found in Algorithm 3.

Algorithm 3: Anomaly Detection Evaluation

```
Input: Test image $I$. Normal distribution parameters.
Output: Binary mask of anomalies $\hat{\Omega}_I$.

 1. Extract the test patches $x_j \in E_I$ from the image $I$.

 2. For each $x_j \in E_I$
        If $median(x_j) > \epsilon$
            Calculate the features vector $f_j$.
            If algorithm = Mahalanobis
                $S_J = \sqrt{(f_j - \mu_t)^T \Sigma_t^{-1} (f_j - \mu_t)}$ (4.17).
            If algorithm = KDE
                $S_J = log(\phi_\theta(f_j))$ (4.18).
        Else
            $S_j = \mu_f$.

 3. Generate the anomaly map $\omega_I$.

 4. Apply the threshold $\gamma$ to compute the binary mask of anomalies $\hat{\Omega}_I$.
```

4.4 Anomaly Detection method based on CW-SSIM applied to Autoencoders

In our second method, we propose Complex Wavelet Structural Similarity (CW-SSIM) [29] as loss function to train the convolutional autoencoder. The idea comes as an extension of the work of *Bergmann et al.* [3], which use the Structural Similarity Metrics (SSIM) [38] as loss function (see Section 2.2.3). The use of CW-SSIM allows to improve all the important contributions of SSIM, such as to avoid yielding high residuals in locations where the reconstruction is only slightly inaccurate. CW-SSIM allows non-structural contrast and intensity changes, which are detectable but do not affect the perceived quality of an image, while it increases the tolerance to translation, scaling and rotation with respect to SSIM.

Once the training phase has ended, we evaluate the abnormality of each test patch with a reconstruction-based approach. We calculate the visual similarity distance between the input and the output of the decoder (also called reconstructed input) as anomaly map ω_I. This calculation is done applying CW-SSIM [29]. The rationale behind this solution is that the autoencoder is not able to correctly reconstruct anomalies, since it is trained only with normal images, and in these areas the CW-SSIM distance will be large. The great advantage of this evaluation phase with respect to the previous one (4.3.5) is that the anomaly scores do not need to be upscaled.

4.4.1 Preprocessing

The preprocessing step consists in rescaling all the patches between 0 and 1: $\{x_i \in \mathbb{R}^{h_t, w_t} | 0 \leq x_i \leq 1\}$. This is done to improve the training convergence.

4.4.2 Autoencoder

The autoencoder is used in this method to extract the features of normal textures in a Semi-Supervised fashion. We deploy a fully convolutional autoencoder (made only by convolution operations), since we want to provide in input images of any dimension, without modifying the structure. We define $E(\cdot)$ the encoder network that takes in input the image x and generates the compressed latent representation, also referred to as bottleneck, $z = E(x)$. Then, $D(\cdot)$ denotes the decoder network that, starting from the latent representation z, upscales the features vectors to the resolution of the input image $y = D(z) = D(E(x))$. The latent representation $z \in \mathbb{R}^{N_l}$ is a features vector with the same dimension N_l of the autoencoder's bottleneck.

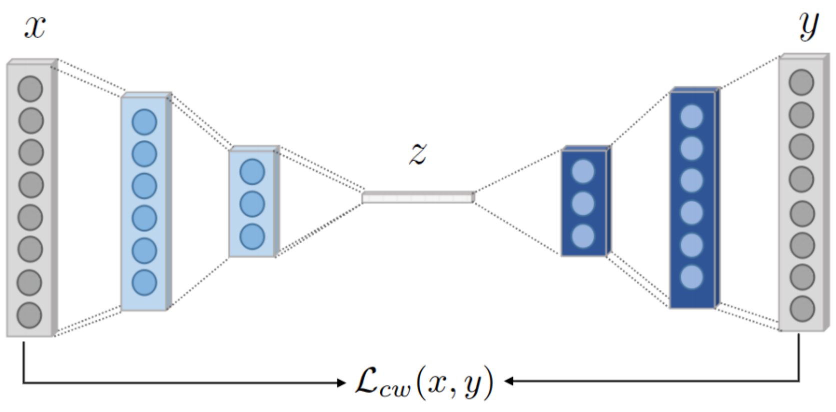

Figure 4.7: Autoencoder training scheme. The figure shows the encoder $E(x)$ on the left, the latent representation z in the center and the decoder $D(z)$ on the right.

4.4.3 Loss Function

Complex Wavelet Similarity Metrics (CW-SSIM) is inspired by the impressive pattern recognition capabilities of the human visual system, since it was discovered that neurons in the primary visual cortex are well-modeled localized multi-scale bandpass oriented filters (referred as *wavelets*), that decompose images into multiple visual channels [5]. CW-SSIM attempts to emulate these features in order to design a measurement that is insensitive, up to a few pixels, to non-structured geometric image distortions [29].

The first operation is to decompose x and y into M subbands using the Complex Steerable filters [31], as we have presented above in Section 4.3.2. We recall that the number of subbands M depends on the scale S and orientation O parameters. Regarding the number of scales S, we decide to deploy a pyramid which allows us to achieve almost the same spatial dimension in the last (low-pass) subband, with respect to the previous proposed method. Concerning the parameter O, instead, we can follow the same reasoning of the previous Section 4.3.2, assuming that, even applying a few number of oriented convolutions, the differences between anomalous and normal textures can be captured by the Steerable filters. This procedure decomposes each patch x and y into M different subbands: we can define x^m and y^m

with $m = \{1, ..., M\}$ the subband extracted at m^{th} position in the input and the reconstructed patch respectively.

The comparison takes place locally: this means that each subband is divided in small windows of size $L \times L$ following a regular grid sampling strategy with stride 1. We can define $C_{x^m} = \{c^i_{x^m} | m = 1, ..., M; i = 1, ..., L^2\}$ and $C_{y^m} = \{c^i_{y^m} | m = 1, ..., M; i = 1, ..., L^2\}$ the two sets of coefficients extracted at the same pyramid's subband m and the local window position i. The CW-SSIM index between two windows at the same spatial location i is defined as:

$$\mathrm{CW}(c^i_x, c^i_y) = \frac{2\left|\sum_{m=1}^{M} c^i_{x^m} (c^i_{y^m})^*\right| + K}{\sum_{m=1}^{M} \left|(c^i_{x^m})^2\right| + \sum_{m=1}^{M} \left|(c^i_{y^m})^2\right| + K}, \tag{4.20}$$

where c^* denotes the complex conjugate and K is a small constant that avoids the denominator to tend to zero [29]. In order to produce the loss function, we need a single score for the entire patch comparison. For this purpose, the CW-SSIM index is averaged over all the windows:

$$\mathcal{L}_{cw}(x, y) = 1 - \frac{1}{L^2} \sum_{i=1}^{L^2} \mathrm{CW}(c^i_x, c^i_y). \tag{4.21}$$

The loss value ranges between [0, 2], where 0 indicates maximum similarity.

In CW-SSIM (as in SSIM), the processing is done on a sliding window basis. This is essential when the goal is to ignore point-wise differences, but we want to make sure that only local variations on the scale of the window size are penalized by the metrics. Note that the window size determines the texture scale relevant to our problem. Thus, if the window is large enough to include several repetitions of the basic pattern of the texture, then the statistical features extracted represent the image textures [41]. The main reason why STSIM in Section 4.3.3 was applied on the entire patch is that the goal was the overall similarity of two texture patches, where the assumption was that they constitute uniform (homogeneous) textures. In those settings, the global window produces more robust statistics, unaffected by local variations [41]. The training Algorithm is described in 4.

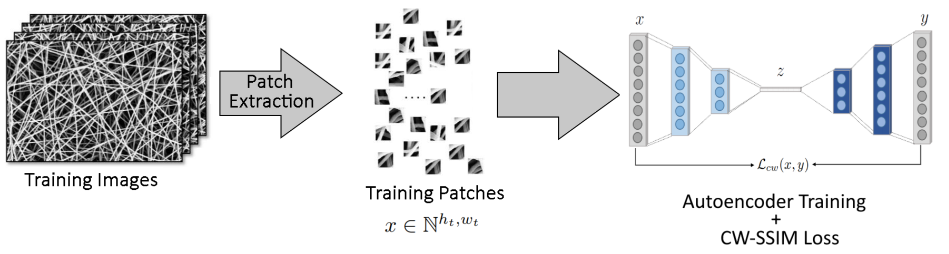

Figure 4.8: Data-Driven training procedure.

```
Algorithm 4: Autoencoder Feature Extraction
  Input: Set of normal images.
  Output: Trained autoencoder.
    1. Extract patches randomly from the training images: $x_i \in T$.
    2. Apply the preprocessing by rescaling each image between 0 and 1.
    3. Train the autoencoder only with normal patches $T$, using
       $\mathcal{L}_{cw}(x, D(E(x)))$ as loss function (4.21).
```

4.4.4 Reconstruction-Based Anomaly Detection

The rationale behind the proposed method is to extract the features of normal patches in a Semi-Supervised approach. When these features, included in the latent space z, flow through the decoder, they are reconstructed in such a way that they become patches very similar to the input ones. When, after the training phase, a patch which contains an anomalous blob is provided in input, the encoder E is able to extract only the features that correspond to normal patterns. In the end, what the decoder reconstructs in the place of the anomalous blob is the most similar normal pattern. At this point, we propose to compare the original and reconstructed patch using CW-SSIM, annotating as anomalies the pixels which present higher dissimilarity. We can split the test dataset in two sets: the validation set, used to decide the threshold value γ, and the test set upon which the evaluation metrics are calculated. The full reconstruction-based Algorithm can be found in 5. As before, we postpone the discussion about the threshold value in a separate Section 6.2.

For each test image I belonging to the test set, the patches $x_j \in E_I$ are extracted following a regular grid sampling strategy with stride s. Each test patch is then processed by the autoencoder generating the corresponding reconstruction patch y_j. These patches are finally used to generate the full resolution reconstructed image I', by averaging the reconstructed pixel values coming from different patches. An example of a reconstructed patch using the SEM dataset can be found in Figure 4.9.

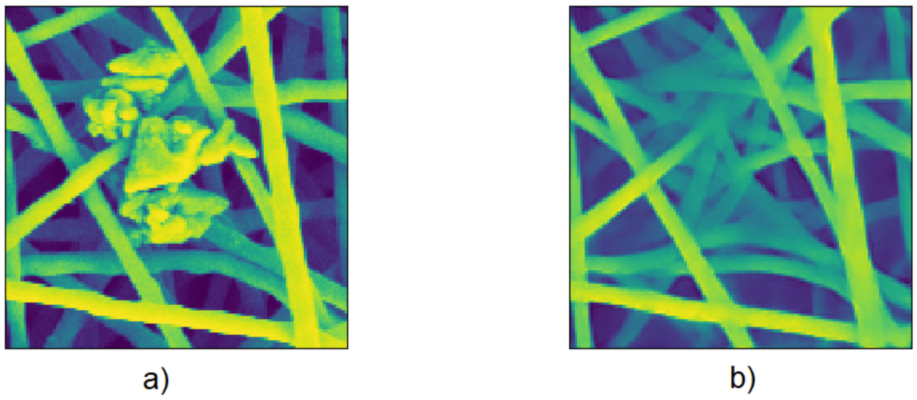

Figure 4.9: Anomalous pattern from a 128×128 patch, reconstructed by the autoencoder. a) The original patch. b) The patch reconstructed by the autoencoder. It is possible to notice that what the decoder has reconstructed is the tubular pattern in place of the anomaly.

In order to produce the anomaly map ω_I, the reconstruction-based approach compares the original image I and the reconstructed one I'. In contrast to the l^2 distance (2.10), which is common practice to use, we propose the use of CW-SSIM which gives more importance to the salient differences than to slight inaccuracies around reconstructed edges. Another motivation is that, in our case, the point-wise differences in anomalous areas are not so large (since the gray level of an anomaly is similar to the tubular one), thus a metrics that penalizes differences in a wider region is essential, as we will demonstrate in the evaluation Section 6.4.

To compute the anomaly score between an entire image I and its reconstruction I', we first apply the subband decomposition on both of them, upsampling each subband to the original patch size. At this point, we apply the CW-SSIM equation in each window position (4.20), but then we do not average all the anomaly scores, as we have done in the loss function. In the end, the degree of abnormality of each pixel is obtained by averaging the degree of abnormality of each corresponding window location on different scales. In order to produce more robust results, we decide to average the anomaly maps coming from different CW-SSIM scores, which are built using subband decompositions at various scales. An example of an anomaly map coming from a test patch can be found in Figure 4.10.

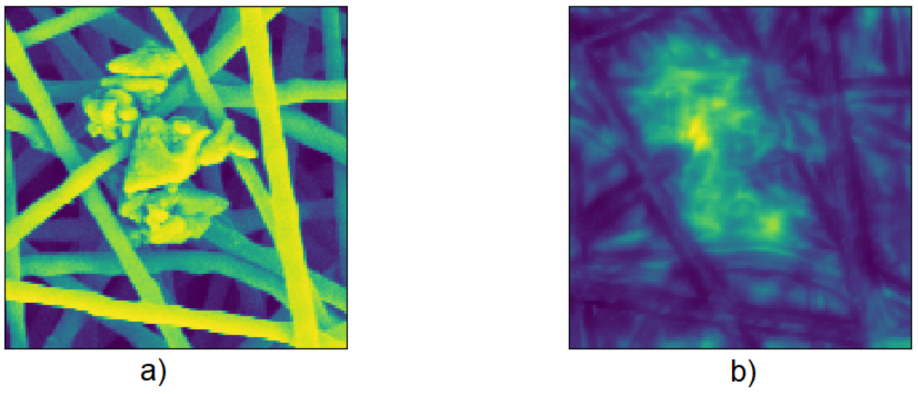

Figure 4.10: Anomalous pattern from a 128×128 *patch and the corresponding anomaly map. a) The original patch. b) The anomaly map of the patch using CW-SSIM index. The anomalous area is highlighted in the anomaly map.*

The final step is to generate the binary mask of anomalies $\hat{\Omega}_I$ applying the threshold γ:

$$\hat{\Omega}_I(i,j) = \begin{cases} 1, & \text{if } \omega_I(i,j) \geq \gamma, \\ 0, & \text{otherwise.} \end{cases} \tag{4.22}$$

This procedure requires also two post-processing steps; the first one allows us to eliminate, from $\hat{\Omega}_I$, anomaly values that refer to dark pixels in the original image I:

$$\hat{\Omega}_I(i,j) = \begin{cases} 1, & \text{if } I(i,j) \geq \delta, \\ 0, & \text{otherwise.} \end{cases} \tag{4.23}$$

Then, as the second post-processing step, a circular structuring element is applied as a morphological operator to delete outlier regions that are only a few pixels wide.

Algorithm 5: Reconstruction-Based Novelty Detection

```
Input: Test image $I$. Trained autoencoder.
Output: Binary mask of anomalies $\Omega_I$.
  1. Extract the test patches $x_j \in E$ from the image $I$.
  2. For each $x_j \in E$
         Calculate the reconstructed patch $y_j$.
  3. Generate the full reconstructed image $I'$.
  4. Produce the anomaly map $\omega_I$ by measuring the CW-SSIM
     dissimilarity between the original image $I$ and the reconstructed one
     $I'$.
  5. Calculate the binary mask of anomalies (4.19).
  6. Apply the threshold for dark pixels (4.23) and the morphological
     operator.
```

Chapter 5

Implementation Details

In this chapter, we present all the implementation details that are behind the proposed solutions. We explain all the parameters' choices that we made focusing on the effective implementation of the algorithms presented before. In particular, we discuss about the patch dimension that we have chosen in the two methods, the subband decomposition parameters and the different architectures of the two training procedures. This chapter is structured in two main sections: in the first one 5.1 we present the implementation details about the method based on STSIM, while in the second part 5.2 the details about the method based on CW-SSIM.

5.1 Implementation of the STSIM based method

First of all, we make a brief recap of the Anomaly Detection method based on STSIM. If we recall the scheme in Figure 5.1, the feature extraction procedure is composed by three different steps. In the first one, the training patches are extracted from anomaly-free images. Once the extraction has ended, the training patches are convolved with the Steerable filters in order to produce their set of Steerable subbands. Finally, the STSIM statistics are computed from each subband and will be used to create the features vector of each training patch.

The Anomaly Detection training procedures will be deployed to learn the distribution of the features, coming from normal patches, in the feature space. The trained model, then, will be used to evaluate the anomaly score of test images in a patch-wise fashion.

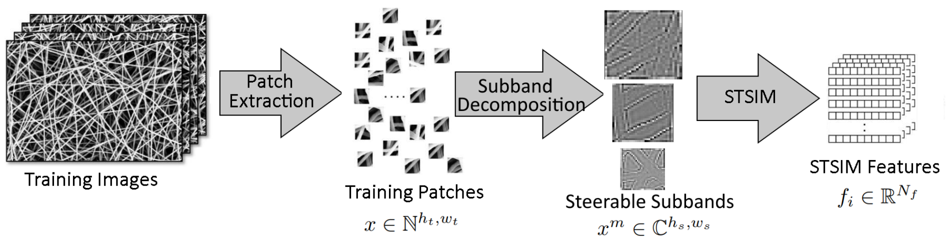

Figure 5.1: Complete feature extraction procedure. The template scheme is taken from [20].

5.1.1 Patch Extraction and Preprocessing

The patch dimension ($w_t = h_t$) is a critical choice because it determines the anomaly map resolution, since each test patch provides a single anomaly score that must be upsampled to the patch dimension. This means that too large patches would lead to have less resolution in the anomaly map. This choice also determines the dimension upon which the features are captured: too small patches would not lead to capture the features that characterize normal textures, while too large ones force the feature extraction procedure to be less sensitive to small anomalous regions. For the aforementioned reasons, the performances of our method based on STSIM highly depend on this parameter. The research for the best trade-off in the patch dimension is fundamental. We empirically choose $w_t = 20px$ for the implementation that uses the KDE, while $w_t = 18px$ for the one that uses the Mahalanobis distance as anomaly metrics. The choice of these values will be fully discussed in the evaluation Section 6.3.

The parameter ϵ defines the minimum median value of each training patch (4.2). It is a manually tuned parameter that we set to 30 (out of 255, the maximum gray-scale value). During the training phase, we deploy $N_t = 500$ patches for the Mahalanobis implementation and $N_t = 2500$ for the KDE one. The number of patches extracted will clearly affect the accuracy of the Anomaly Detection procedures, thus we use a higher number of training patches for the KDE algorithm, since it is based on a more complex model.

5.1.2 Subband Decomposition

We recall that the number of subbands M strictly depends on two factors: the patch texture and its dimension. In our tests, the best configuration is achieved with the number of scales $S = 2$ or $S = 3$ and the number of orientations $O = 4$. We decide to try at maximum 3 scales, since going further means treating subband sizes of less than 9 or 10 pixels, that we consider the lower bound in the subband dimension. Experimental results in the evaluation Section 6.3 confirm that using lower sizes would not lead to capture the features that characterize normal textures. Regarding the parameter O, we can notice that normal patches are characterized by high differences in gradient in any direction, whereas anomalous patterns are characterized by uniform agglomerates that have almost the same response in any direction. The experimental results prove that, even applying a few number of oriented convolutions, these differences can be captured. An example of a 3 scales, 4 orientations decomposition applied to a 32×32 input patch can be found in Figure 5.2.

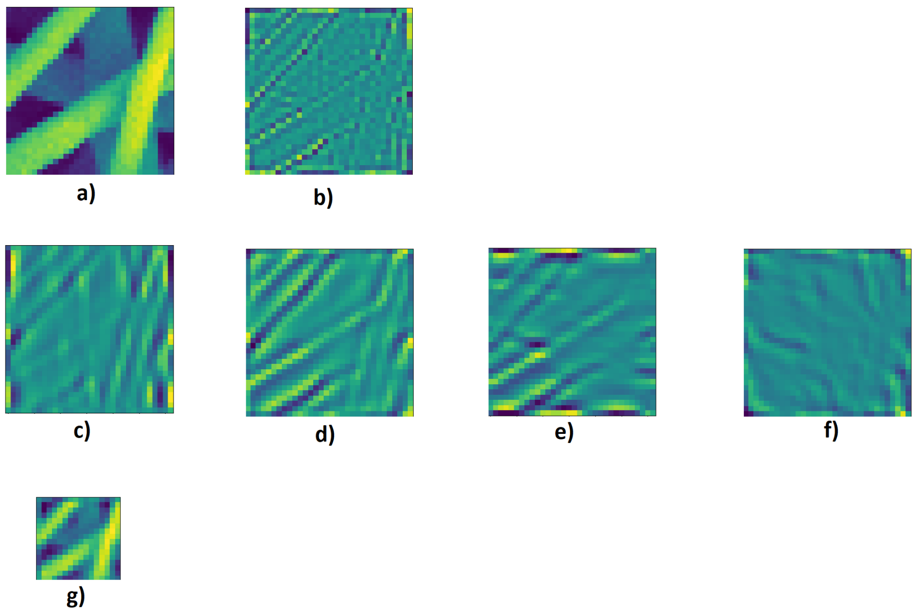

Figure 5.2: Steerable subband decomposition with $S = 3$ and $O = 4$, applied to a 32×32 patch. We decide to show a larger patch, with respect to the ones that we really use, for better interpretability. a) Original patch. b) High-pass subband. c)-f) Band-pass subband at scale 2. g) Low-pass subband. The oriented subbands (c-f) show us that they present high differences in gradient and thus the probability of belonging to normal textures is greater.

5.1.3 Training Parameters

Once all the features vectors of the training patches are extracted, we need to learn the parameters that characterize their distribution. In our experiments, we fix the split of the training dataset explained in Section 4.3.4 as 80% (T_f) - 20% (V_f). Regarding the KDE training, we can mention that we model the density estimate (3.4), using a Gaussian Kernel with bandwidth b equal to 0.5.

5.1.4 Anomaly Detection

The Anomaly Detection evaluation needs that each test image I is divided in patches. We choose to maintain the same patch size of the training set following a regular grid sampling strategy with stride $s = 1$. We believe that it is fundamental to keep the same size of the training patches in order to extract the same type of features. We finally choose to have the minimum possible stride in order to have the best possible accuracy in the anomaly map ω_I. The value of each pixel in the anomaly map is the mean value of the anomaly score of all the patches over that position. An example of an anomaly map computed using the aforementioned procedure can be found in Figure 5.3.

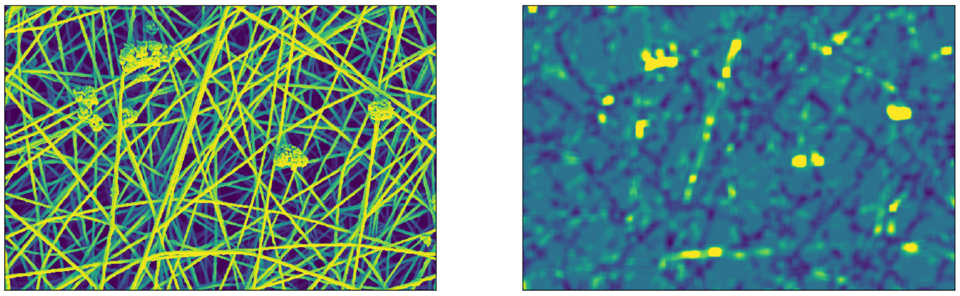

Figure 5.3: Test image I and its anomaly map ω_I computed using Mahalanobis distance. We can notice that anomalous blobs are highlighted in the anomaly map. This is due to the fact that the distance is greater in the anomalous areas.

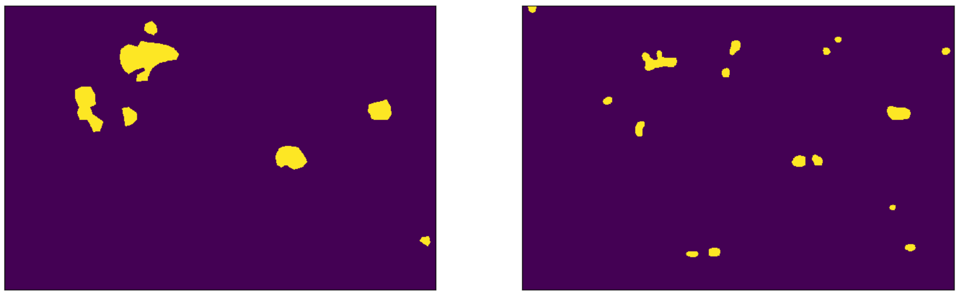

Figure 5.4: Reference binary mask Ω_I and the one generated by our solution $\hat{\Omega}_I$ from the anomaly map defined in Figure 5.3.

5.2 Implementation of the CW-SSIM based method

In the Anomaly Detection method based on CW-SSIM applied to autoencoders, we extract patches from normal images in order to train an autoencoder to reconstruct its input. The objective of the network is to minimize the perceptual similarity between the input and the output of the autoencoder. During this stage, the model learns how to extract features that characterize normal patches.

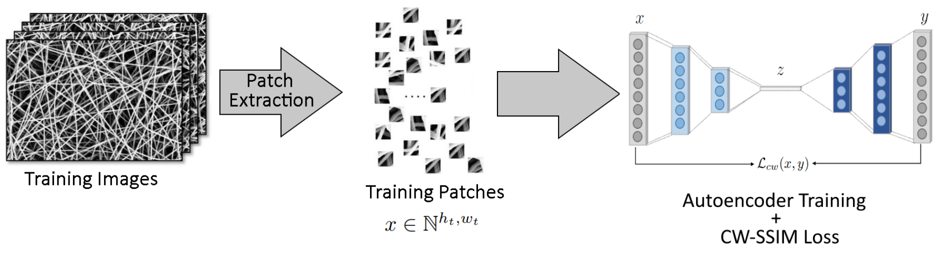

Figure 5.5: Data-Driven training procedure.

This solution exploits the idea that the autoencoder is not able to correctly reconstruct anomalies, since it is trained only with normal patches. In principle, the anomalous areas will be reconstructed with textures as similar as possible to the normal ones, thus the CW-SSIM distance between the original and the reconstructed patches will we be large in those regions.

5.2.1 Patch Extraction and Preprocessing

We choose to select a larger patch dimension with respect to the previous method. This decision is motivated by two factors: the first one is that we do not extract a single anomaly score from each patch during anomaly evaluation, but one for each window. We recall that in the previous method it was fundamental to choose a small patch size in order to increase the resolution of the anomaly map. The second motivation is that CW-SSIM works better with particularly large patches, as described in its original paper [29]. In any case, too large patches would require higher computational effort and could lead to lower precision in the global anomaly score; for this reason we manually tune the patch dimension to $w_t = h_t = 128px$.

5.2.2 Autoencoder Architecture

The encoder $E(\cdot)$ is built by 5 convolutional layers, which reduce the input space dimension to the latent dimension. They are all characterized by a kernel dimension equal to 4. We select small filters in order to capture tiny features from the beginning. The unique parameter that we vary between different layers is the stride which indicates how many pixels skip after each convolution: for example stride equal to 2 means that the output of the convolution will be halved in spatial dimension. It also changes the filter size, called also channel dimension, which indicates how many different 4×4 kernels are applied. We can notice that, from the structure that you can find in Figure 5.6, as we approach to the latent space the number of filters increases, while the spatial dimension decreases. This is a common procedure able to capture features at different resolutions, until the latent space $E(x) = z$. In particular, selecting $w_t = h_t = 128px$, we have that the z is a one-dimensional vector of 512 features. The architecture of the decoder $D(\cdot)$ is exactly the reverse of the encoder one, but instead of deploying classical convolution operations, they are transposed convolutions which are able to increase the spatial dimension. The upscale factor is tuned by the stride parameter. The output of the decoder $y = D(E(x))$ has the same spatial and color dimension of the input patch.

Each convolutional layer, both in encoder and decoder, is followed by a Leaky Rectified Linear Unit (Leaky ReLU) activation function [17] with slope 0.3, which introduces non-linearity in the model. The last convolutional layer in the decoder, instead, is followed by a linear activation function. Since we train the model from scratch, i.e. do not apply transfer learning, we initialize the weights with the Glorot initialization [11], which allows faster training convergence in this setting. This architecture counts 5,573,057 number of parameters.

5.2.3 Loss Function

The autoencoder deployed is trained to minimize the perceptual similarity between the input image and the reconstructed one using the CW-SSIM index. This index is based on the complex Steerable decomposition, which is implemented here as explained in its dedicated Section 4.3.2. Regarding the number of scales S, we decide to deploy a pyramid which allows us to achieve the spatial dimension of 16 pixels in the last (low-pass) subband, which is similar to the one achieved in the STSIM-based method. Concerning the parameter O, we can make the same assumption of the STSIM-based

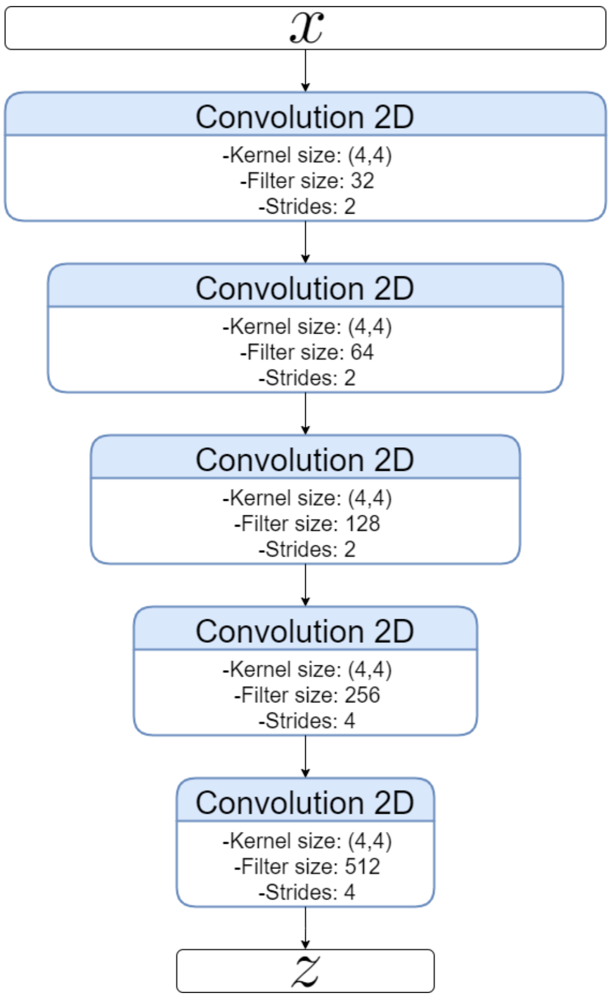

Figure 5.6: General outline for the encoder architecture. The decoder has exactly the reverse structure, but with Transpose Convolution 2D layers.

solution that, even applying a few number of oriented convolutions, the differences between anomalous and normal textures can be captured by the Steerable filters. For the aforementioned reasons, we empirically choose $S = 5$ and $O = 5$. Once the subbands, both from the input and the reconstructed patch, are generated, the CW-SSIM index can be calculated as in (4.21). We recall that this metrics is computed locally with a sliding window of size $L \times L$. We empirically choose $L = 7$ as the best window size.

5.2.4 Training Parameters

The autoencoder architecture is trained on 50,000 defect free patches of size 128×128, divided in batches of 256 patches in order to fit in memory. We train the network for 200 epochs, using the ADAM [14] optimizer with an initial learning rate of 1×10^{-3} and a weight decay set to 0.5 every 20 epochs.

5.2.5 Anomaly Detection

At the end of the training procedure, we will have a network able to extract the features that characterize only normal patches. This solution uses the CW-SSIM similarity metrics between the original and the reconstructed patches in order to generate the heat-map of the anomalies. In principle, anomalous areas are very different in the two patches and thus the similarity score will be lower. The evaluation is performed by striding over the test image I and reconstructing patches of size 512×512 using the trained autoencoder. We cannot fit the whole image to the autoencoder because it is not squared. In principle, it would be possible to set the horizontal and vertical stride to 512. However, at different spatial locations, the autoencoder produces slightly different reconstructions of the same data, which leads to some striding artifacts. Therefore, we decrease the stride to 16 pixels and average the reconstructed pixel values [3]. In the end, we have produced the full reconstructed image I'. An example of a reconstructed test image and its original version can be found in Figure 5.7.

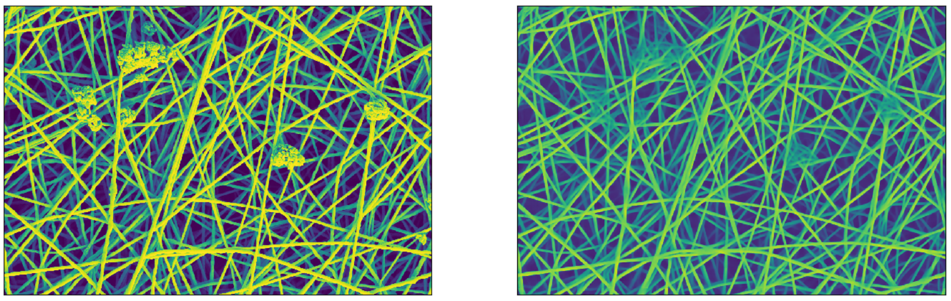

Figure 5.7: Test image I and its version I' after the reconstruction procedure. We can notice that anomalous blobs are reconstructed with filamentous textures.

The anomaly map is generated by computing the CW-SSIM similarity between the original image I and the reconstructed one I'. In our implementation, we average the scores of 3 CW-SSIM indexes coming from different Steerable configurations at $S = 7, 8, 9$ and $O = 6$, all of them using a window size of $L = 7$. These metrics, working locally, could produce false indications near the image sides; for this reason we decide to suppress the anomaly score over these pixels. In our configuration, we choose to halve the values of all the scores which present a distance from the nearest border of 5 pixels at the most. An example of an anomaly map computed using the aforementioned procedure can be found in Figure 5.8.

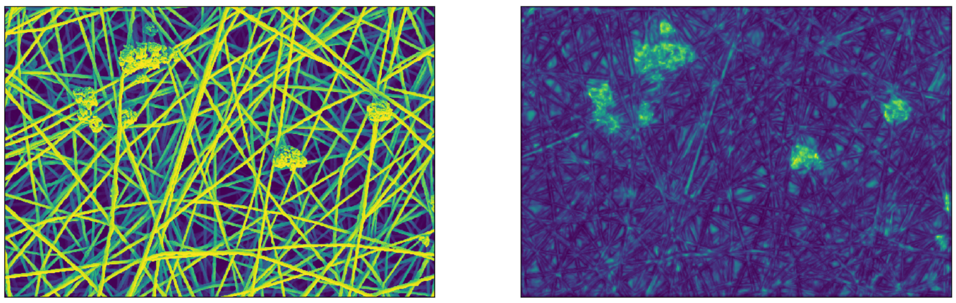
Figure 5.8: Test image I and its anomaly map ω_I. We can notice that anomalous blobs are higlighted in the anomaly map. This is due to the fact that the CW-SSIM index is lower in anomalous areas (the colormap is inverted for better interpretability).

In the end, the binary mask of anomalies $\hat{\Omega}_I$ is generated applying the threshold γ to the anomaly map ω_I. The post-processing step is composed of two operations: the first one allows us to eliminate, from $\hat{\Omega}_I$, anomaly values that refer to dark pixels in the original image I. To do this, we manually choose the dark pixels threshold parameter $\delta = 30$. Then, as the second post-processing step, a circular morphological operator with diameter 10 is applied to delete outlier regions that are only a few pixels wide. An example of binary mask of anomalies after the post-processing steps can be found in Figure 5.9. From the resulting figures, we can qualitatively notice that the anomaly detection capability is greater with respect to the previous method, without loosing resolution. The motivation behind this differences in the results will be fully discussed in the performance evaluation Section dedicated to this method 6.4.

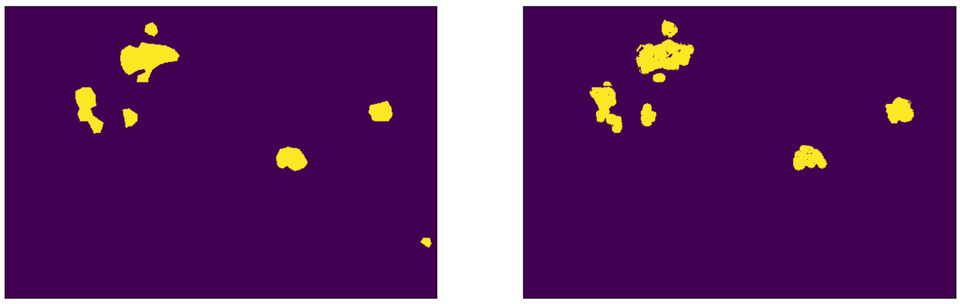
Figure 5.9: Reference binary mask Ω_I and the one generated by our solution $\hat{\Omega}_I$ from the anomaly map defined in Figure 5.8.

Chapter 6

Experiments and Performance Evaluation

In this chapter, we present the experimental evaluation of our solution. In the first Section 6.1, we introduce the dataset upon which we base our experiments. In the second Section 6.2, we provide which are the metrics that we use to evaluate our methods. In the following two Sections 6.3 and 6.4, we present the performance of the methods based on STSIM and CW-SSIM respectively, with an in-depth discussion about the results. Then, in Section 6.5, we compare our results with three State of the Art techniques, which use our same dataset: the one based on Sparse Dictionary Learning [7], presented in Section 2.2.1; CNN-Based Self Similarity [20], presented in Section 2.2.2; and finally the one that uses SSIM as loss function of an autoencoder [3], presented in Section 2.2.3. In this section, we also report the evaluation time analysis, regarding both the training and the evaluation phase of our methods compared with the ones in literature. In the last Section 6.6, we report some experiments in which we modify the CW-SSIM method, using a latent based anomaly score in place of the reconstruction error to generate the anomaly map. We decide not to add this solution among the proposed ones, since it achieves very poor results in our experiments.

6.1 Dataset

The dataset used in this work is composed of images of nanofibrous materials acquired with the FE-SEM (Carl Zeiss Sigma NTS, Gmbh Oberkochen, Germany). The Field Emission Scanning Electron Microscopy (FE-SEM) is an electron microscope which provides topographical and elemental information with a minimum spatial resolution of $1nm$. All images are acquired in the same conditions and using the same parameters. The external appearance of the basic material can be seen as a non-periodic continuous texture of filamentous elements that look like bright wires [20]. The anomalous areas, instead, are composed by agglomerates of filamentous material which can appear in any position of the image. Some examples with and without anomalies are in Figure 6.1.

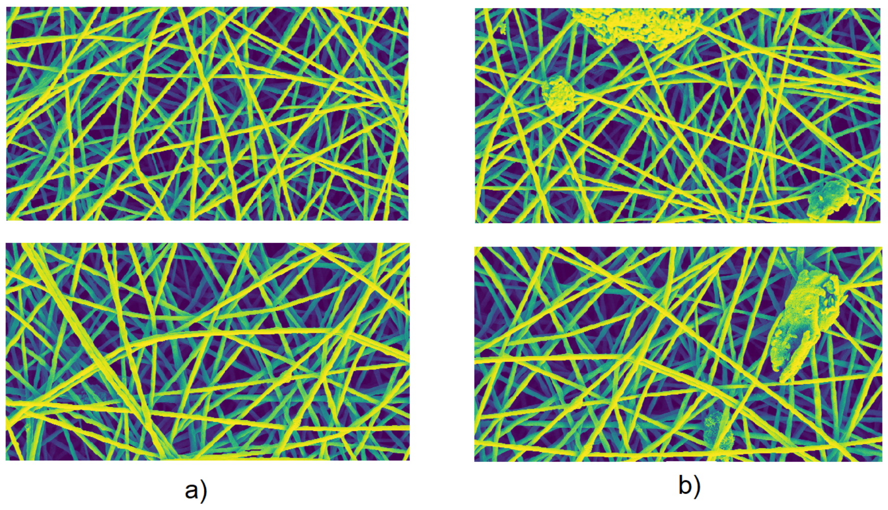

Figure 6.1: Portions of normal and anomalous images. a) Portion of normal images. b) Portion of anomalous images.

The dataset is composed of 45 images, 5 of them are normal (do not contain any anomaly) and the other 40 are anomalous images (contain at least an anomaly). In order to make a fair comparison with the State of the Art, we use the same database splitting between training, validation and test sets present in literature. The training set is composed by all the 5 normal images. The other 40 images are divided in the test set, which contains 35 images, and the validation set, which contains the other 5 images. For sake of comparison with the State of the Art, we fix the validation set with images at the following positions: 8, 15, 27, 31, 35. All the defects have been manually annotated. The dataset and the defect annotations are publicly available at [19]. Each image I is a 8-bit gray-scale of size 696×1024. The annotation associated with each anomalous image is the map $\hat{\Omega}_I \in \{0, 1\}$. Figure 6.2 shows an anomalous image along with its defect annotation. The training set will be used by our two methods to learn the model characterized by normal data. The validation set will be used to determine the threshold γ, while the test set, instead, will be used to assess the quality of the proposed solutions with respect to the current State of the Art techniques. Finally, we remark that the overall defects in anomalous images are very small: on average they cover 1.3% of the image, and only the 0.5% of the anomalies exceeds the 2% of the image size [7].

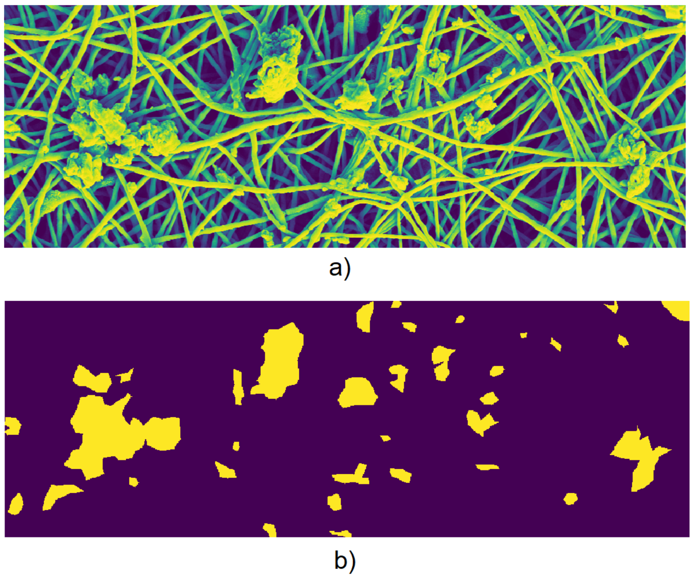

Figure 6.2: Portion of an anomalous image and the corresponding anomaly mask. a) Portion of a normal image I. b) Portion of the reference anomaly mask Ω_I.

6.2 Evaluation Metrics

In this section, we present the metrics that we will compute to evaluate the performance of our proposed solutions. We decide to propose the same metrics used by the selected reference methods in literature, in order to have a fair comparison with them. In particular, we are going to present the *Area Under Curve* (AUC), which provides an aggregate measure of performance across all possible classification thresholds, and the *Defect Coverage* as the percentage of pixels covered by the binary mask at a certain threshold. Moreover, we provide another metrics called *Intersection over Union* (IoU), used to measure the accuracy of the model's prediction with respect to the ground-truth, which is not reported in the reference methods.

6.2.1 Area Under Curve

The AUC index is based on the generation of the *Receiver Operating Characteristic* curve (ROC curve), that is a graph showing the performance of a classification model at all classification thresholds. It is a two dimensional graph composed of the *True Positive Rate* (TPR) in the vertical axis and the *False Positive Rate* in the horizontal one. The TPR (also called as recall) is defined as:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{6.1}$$

where TP is the number of pixels correctly classified as anomalous, and FN the number of pixels that are not recognized as anomalous by the model. The FPR is defined as follows:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}, \tag{6.2}$$

where FP is the number of normal pixels that are wrongly classified as anomalous (they are normal), and TN the number of pixels correctly classified as normal. Each point in the ROC curve indicates the intersection between the TPR and FPR at different thresholds, thus the line from (0,0) to (1,1) indicates the random classification. The Figure 6.3 shows an example of two ROC curves.

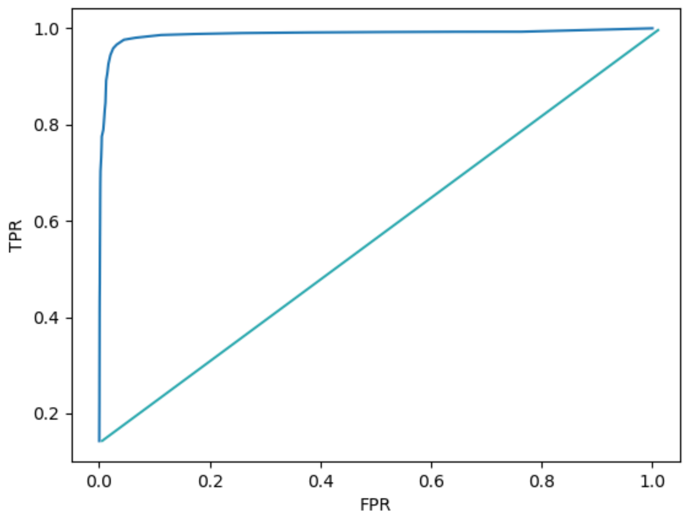

Figure 6.3: ROC curve of the solution based on CW-SSIM (blue) vs. random classifier (aquamarine).

AUC measures the entire two-dimensional area underneath the entire ROC curve (integral from (0,0) to (1,1) of the curve). AUC provides an aggregate measure of the performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a randomly chosen positive example more highly than a random negative one [9].

6.2.2 Defect Coverage

The second metrics that we want to use is the one proposed in [7]. This metrics represents the coverage percentage of defects at a certain level of threshold. We use the same configurations of all the other solutions in literature, choosing a threshold value γ such that FPR is about 5% over the validation set. To compute the Defect Coverage percentage, we detect each defect within each reference map Ω_I, by finding each connected component $cc_j \in \Omega_I$. Each cc_j with $j = \{1, ...D\}$ represents a single anomalous blob, out of D, in Ω_I. For each of these defects j, we calculate the Coverage Factor as follows:

$$\text{Coverage Factor}_j = \frac{\text{TP}_j}{\text{TP}_j + \text{FN}_j}. \tag{6.3}$$

Therefore, the Coverage Factor of a connected component is the number of correctly detected pixels over its area. The Defect Coverage metrics reports the minimum Coverage Factor over more than 50% anomalous blobs in the test set.

6.2.3 Intersection over Union

The IoU index is one of the most popular metrics in the Semantic Segmentation field. It is used to measure the accuracy of the model's predicted binary mask with respect to the reference one, penalizing the wrong classification more than the previous two indexes. It is defined as the area of overlap between the predicted segmentation and the ground truth, divided by the area of union between the predicted segmentation and the ground truth:

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}}. \tag{6.4}$$

This index is particularly penalizing when, as in our case, we have an important unbalance between the two classes (we recall that on average the anomalies cover 1.3% of the test images). The threshold γ is chosen for completeness as we have done before, in such a way that FPR is about 5% over the validation set. In IoU this threshold results too low, producing many false indications. For this reason, we also report the performances at FPR of 3%, only for the best configuration.

6.3 Evaluation of the STSIM based method

In this section, we report the performance evaluation of the Anomaly Detection method based on Structural Texture Similarity. Initially, we discuss about the possible parameters' choice, such as the patch dimension, and how they affect the results. Here we also report the performance evaluation achieved by the best configuration. In the end, we add some graphic results, correlated by some considerations about its strengths and limitations.

6.3.1 Algorithm configuration performance comparison

In order to search the best configuration of our first solution and present its results, we tune the three main parameters: the patch dimension (that we can call simply w_t, since we use only squared patches), the subband decomposition number of scales S and orientations O. We recall that the number of scales S strictly depends on w_t while, for the parameter O, we have more freedom in the choice since it does not depend on any other parameter. The complete performance evaluation using KDE as anomaly metrics can be found in Table 6.1, and using Mahalanobis distance in Table 6.2. The research for the best configurations follows a greedy approach: in the beginning, we focus mostly on the patch size, starting from 16 pixels and increasing it until we realize that the performance starts to decrease. This procedure is applied with $S = 2$ and $O = 4$, which we empirically consider good configurations. The configurations that reach the best performance are investigated further by tuning initially S and then O, looking for different feature characteristics.

From the evaluation tables, we can notice that this method achieves the best performances with a patch size of $18px$ using Mahalanobis and $20px$ using KDE anomaly metrics. The difference between the two solutions is very small and can be related to the choice of the KDE's bandwidth b. Tables 6.1 and 6.2 show that the implementation with the Mahalanobis distance outperforms the KDE one in the AUC metrics. Instead, both of them achieve almost the same performances in the Defect Coverage. Regarding this metrics in particular, the choice of the parameter S is fundamental for both the implementations, since it describes the low-pass filter resolution. The results tell us that the number of orientations O do not affect so much the anomaly map resolution. In Figure 6.4 and 6.5, we report some anomaly maps coming from the two implementations. From these examples, we can realize that the implementation with the Mahalanobis distance is less coarse and reports fewer false indications with respect to the version with KDE. Apart from this, they seem to have the same recognition capability. In the end, the STSIM based solution reaches the best performances using the Mahalanobis distance, with the configuration $w_t = 18px$; $S = 2$; $O = 4$, achieving an AUC of 0.818, an IoU of 0.126 covering more than 50% of the defects with a minimal overlap of 0.240.

Algorithm configuration performance comparison with KDE			
Parameters	AUC	Defect Coverage	IoU
$w_t = 18px$ $S = 2$; $O = 4$	0.693	0	0.075
$w_t = 18px$ $S = 3$; $O = 4$	0.638	0	0.045
$w_t = 20px$ $S = 2$; $O = 4$	**0.779**	0.007	0.088
$w_t = 20px$ $S = 2$; $O = 7$	0.774	0.021	0.087
$w_t = 20px$ $S = 3$; $O = 4$	0.771	0.201	**0.126**
$w_t = 20px$ $S = 3$; $O = 7$	0.767	**0.208**	0.124
$w_t = 22px$ $S = 2$; $O = 4$	**0.779**	0	0.096
$w_t = 22px$ $S = 3$; $O = 4$	0.770	0	0.116

Table 6.1: Performance evaluation of the Anomaly Detection method based on Structural Texture Similarity using KDE as anomaly metrics at different algorithm configurations.

Furthermore, we report the performances at a lower threshold (3% of validation FPR) in the best configuration, achieving an IoU of 0.153, but dramatically decreasing the Defect Coverage to 0.127.

Algorithm configuration performance comparison with Mahalanobis			
Parameters	AUC	Defect Coverage	IoU
$w_t = 16px$ $S = 2 ; O = 4$	0.798	0.225	0.119
$w_t = 16px$ $S = 3 ; O = 4$	0.831	0.180	0.110
$w_t = 18px$ $S = 2 ; O = 4$	0.818	**0.240**	0.126
$w_t = 18px$ $S = 2 ; O = 7$	0.818	**0.240**	0.126
$w_t = 18px$ $S = 3 ; O = 4$	**0.848**	0.124	**0.135**
$w_t = 18px$ $S = 3 ; O = 7$	0.846	0.128	0.121
$w_t = 20px$ $S = 2 ; O = 4$	0.821	0.140	0.126
$w_t = 20px$ $S = 3 ; O = 4$	0.844	0.009	0.133

Table 6.2: Performance evaluation of the Anomaly Detection method based on Structural Texture Similarity using Mahalanobis distance as anomaly metrics at different algorithm configurations.

6.3.2 Final considerations

The major drawback of this method is that STSIM was originally designed to deal with large patches. In the original paper [41], the authors use patches of 128×128 pixels wide, but here we are constrained to work with as small as possible ones. The rationale behind larger patches is that they allow to deploy more scales in the subband decomposition and thus extract more information at different subband resolutions. The evaluation Tables 6.1 and 6.2 also tell us that increasing the number of orientations does not add any interesting information. As we can see in the Figures 6.4 and 6.5, this fact does not lead to recognize the overall defects but only the parts that are almost regular or very bright. Moreover, very bright zones that are not anomalous, such as the intersection of many fibers, are miss-classified producing false indications.

The exact opposite drawback is the use of a too large patches' anomaly score as per-pixel evaluation. This fact leads to produce square-like regions in the anomaly maps where the defects are present, which will reflect in the degradation of the Anomaly Detection resolution. In addition, we recall that only the 0.5% of the anomalies exceeds the 2% of the image size, so these tiny anomalies have a high probability not to be considered with respect to the rest of the patch. The trade-off that we have reached does not satisfy too much the metrics that we have chosen, in particular the Defect Coverage and the IoU report very low performances. We will see that the solution based on CW-SSIM will produce better results, particularly on these last two metrics, addressing the aforementioned problems.

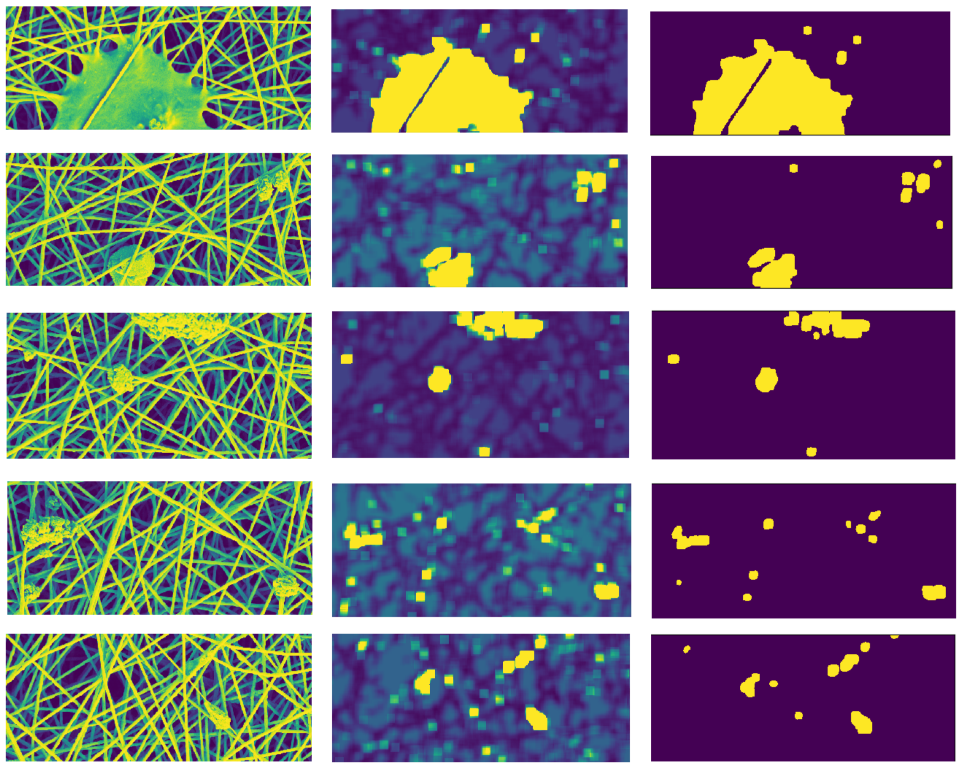

Figure 6.4: Portion of test images evaluated with the STSIM based method implemented with KDE as anomaly metrics. From left to right we can find the original test image I, the anomaly map ω_I and the binary mask of anomalies $\hat{\Omega}_I$.

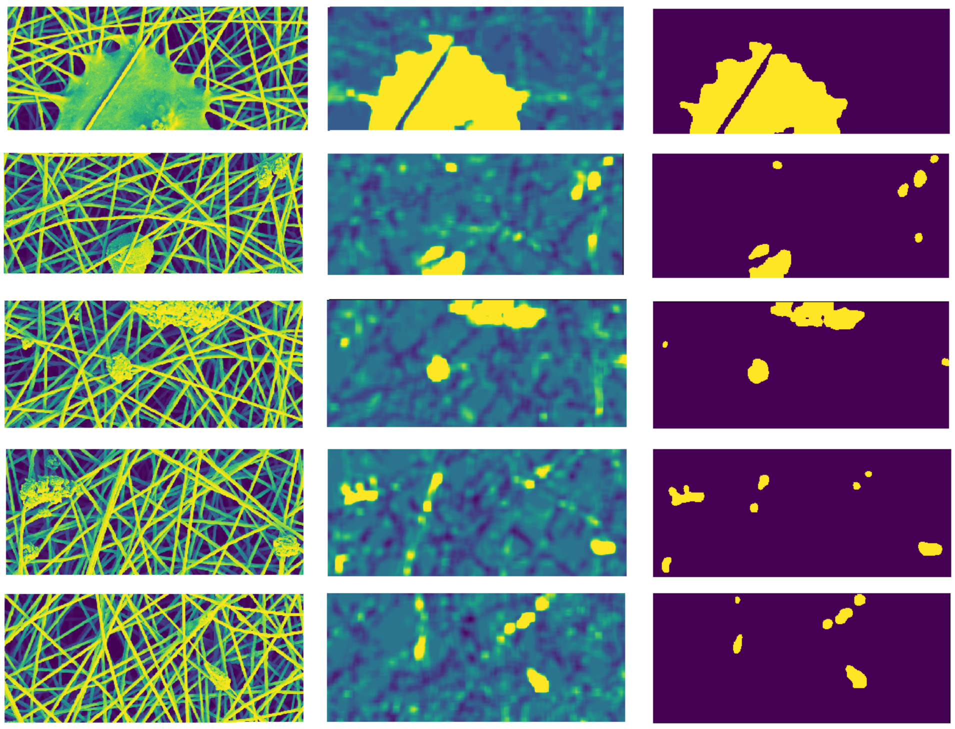

Figure 6.5: Portion of test images evaluated with the STSIM based method implemented with Mahalanobis distance as anomaly metrics. From left to right we can find the original test image I, the anomaly map ω_I and the binary mask of anomalies $\hat{\Omega}_I$.

6.4 Evaluation of the CW-SSIM based method

In this section, we report the performance evaluation of the Anomaly Detection method based on autoencoders. This method trains a fully connected autoencoder with normal patches, using CW-SSIM as loss function. As in the previous section, we initially investigate about the possible parameters configurations and their related performance evaluations. In the following, we compare CW-SSIM with the other distance metrics present in literature both as loss function of an autoencoder and as similarity metrics in the anomaly maps generation. We refer to CW-SSIM as anomaly metrics when we use it as similarity distance in the anomaly map generation. In the specific, we show that CW-SSIM increases the performances of the reconstruction error in autoencoder trained with a different loss function. But it achieves the best results when it is used both as loss and as anomaly metrics. In the end, we add some considerations about the performances of our two solutions.

6.4.1 Algorithm configuration performance comparison

As in the above Section 6.3, we report the performance of our solution tuning the most important parameters: the patch dimension w_t, the subband parameters S, O and finally the window size L upon which the loss is calculated. We recall that all these parameters are interesting for the calculation of the visual similarity between the input and the reconstructed patch. We can assume that, the more this index is precise in the identification of visually similar patches (without penalizing small geometry perturbations), the more the autoencoder is able to extract the features that enable this type of reconstruction. The complete performance evaluation with different configurations can be found in Table 6.3.

The research for the best configurations follows again a greedy approach similar to the one explained in the previous section: in the beginning, we focus mostly on the patch size, starting from 64 pixels and increasing it until we notice that the performance starts to decrease. This procedure is applied choosing S in such a way that the low-pass subband is 16 pixels wide and $O = 5$, which we empirically consider good configurations for the CW-SSIM calculation. The configurations that reach the best performance are investigated further varying the parameters O and L.

The Table 6.3 with the performances of different configurations reports

Algorithm configuration performance comparison			
Parameters	AUC	Defect Coverage	IoU
$w_t = 64px$; $L = 7$ $S = 4$; $O = 5$	0.974	0.947	0.254
$w_t = 128px$; $L = 7$ $S = 5$; $O = 5$	**0.977**	**0.962**	**0.261**
$w_t = 128px$; $L = 9$ $S = 5$; $O = 5$	0.972	0.933	0.253
$w_t = 128px$; $L = 5$ $S = 5$; $O = 5$	0.974	0.939	0.258
$w_t = 128px$; $L = 7$ $S = 5$; $O = 7$	0.973	0.946	0.249
$w_t = 256px$; $L = 7$ $S = 6$; $O = 5$	**0.977**	0.959	0.260

Table 6.3: Performance evaluation of the Anomaly Detection method based on CW-SSIM at different algorithm configurations.

that there are no huge differences in the parameters' choice. Regarding the patch size in this method, we have also to consider that smaller patches would decrease the overall training time. A patch size of $128px$, in addition to reach the best performances, seems a good trade-off between the number of possible subbands deployed and the training time. The window dimension L is an important parameter, since it regulates the sensibility of the metrics; we empirically found that a window size of $7px$ achieves the best performances. In the end, considering more orientations do not produce any interesting results.

6.4.2 Loss function performance comparison

In this section, we show that using CW-SSIM as loss function yields better Anomaly Detection performance than alternatives in the literature. For this purpose, we report the results of experiments in which we use the same autoencoder trained with Mean Squared Error (MSE) (2.10), Structural Similarity (SSIM) (2.15) and Multi-Scale Structural Similarity (MS-SSIM) (3.2). For sake of comparison, the training procedure, the parameters and the Anomaly Detection metrics are the ones that we proposed in the implementation Section 5.2, equal for all the experiments. It is not common practice to use an anomaly metrics different from the loss function; here this choice is made only for comparison purposes. The SSIM experiment is deployed as described in [3], where the window size of CW-SSIM is set to 11. The MS-SSIM metrics is implemented as in its original paper [37], with five scales and the window size set to 11. In Table 6.4, it is reported the performance evaluation over the nanofibrous dataset with different loss functions applied to the same autoencoder configuration.

Loss function performance comparison			
Loss function	AUC	Defect Coverage	IoU
MSE	0.962	0.865	0.245
SSIM	0.969	0.905	0.249
MS-SSIM	0.970	0.931	0.255
CW-SSIM	**0.977**	**0.962**	**0.261**

Table 6.4: Performance evaluation of the Anomaly Detection method based on CW-SSIM involving different loss functions, applied to the same autoencoder configuration.

In order to bring a visual proof of the performances reported in Table 6.4, in which our method outperforms all the others in each performance metrics, we report in Figure 6.6 the reconstruction of three different 128×128 patches in the four previous experiments. The first two patches present medium/small anomalous blobs, while the third is a normal one.

Looking at the anomalous patches in Figure 6.6, we can clearly see that all the four methods achieve different results in the reconstruction of the anomalies. Since it is trained only with normal patches, the first goal of the autoencoder is to reconstruct anomalous blobs in such a way that they look as similar as possible to normal textures. From the first two columns of Figure 6.6, we can notice that the experiment involving MSE (b) poorly achieves this result, generating undefined masses, because this metrics focuses mostly on the pixel wise-differences and not on general textures. From the tests with SSIM (c) and MS-SSIM (d) as loss function, we can perceive that the reconstructed anomalies are more similar to tubular patterns, and this clearly improves the performances. The last patch shows the result of the reconstruction using CW-SSIM (e) as loss function, which is the one that is the most able to replace anomalous blobs with normal textures. We think that it is fundamental to have a pattern in the reconstruction, which is as visually dissimilar as possible to the original one. These large differences allow to generate accurate anomaly maps when we use as anomaly metrics the one that we have proposed.

The second goal of the autoencoder is to perfectly reconstruct normal textures, in order to avoid the generation of false indications in the anomaly map. From the third column of Figure 6.6, we can realize that in this case all the four experiments achieve almost the same reconstruction ability. However, the reconstruction on the experiment with CW-SSIM results less blurred with respect to all the others. An explanation could be that the low-pass filters $L_1(w)$, in the Steerable filters decomposition, are effectively able to remove most aliased components increasing the definition of the filamentous material (see Section 3.1). Moreover, this fact is true also for the patches involving anomalous blobs.

6.4.3 Anomaly metrics performance comparison

Now we want to prove that also the metrics that we propose to compute the anomaly map, in which different configurations of CW-SSIM are applied, particularly increase the performances. To accomplish this, we repeat the experiments that we have presented before, but now the anomaly metrics used to compare the original and the reconstructed image is the one used as loss function. In this case, we have to make additional comments about the experiment involving MS-SSIM. As pointed out in [21], this metrics is undefined in the points where the covariance function, that is the numerator of the structural coefficient (2.13) in MS-SSIM equation (3.2), is negative. The numerical problem comes to light when MS-SSIM rises the structural term to a positive, non-integer number as it is proposed in the original paper [37]. One could argue that this problem is present also when MS-SSIM is used as loss function and not only to compute the anomaly map, that is correct. In that case, however, the structural term is averaged over all the window positions before applying (3.2). This fact allows to mitigate the presence of negative structural values, which are relevant when the two images in that specific window are very dissimilar. For the aforementioned reasons, we use SSIM as anomaly metrics in this experiment.

Comparing the two Tables 6.4 and 6.5, it is clear that CW-SSIM gives an important boost in the recognition of the anomalies over the nanofibrous dataset, both in terms of loss function and anomaly metrics. In particular, when using MSE, it is essential to use a metrics that penalizes differences in a wider region since, as we discussed in Section 4.4.4, the point-wise differences in anomalous areas are not so large. This fact is clearly visible

Anomaly metrics performance comparison			
Loss function & Anomaly metrics	AUC	Defect Coverage	IoU
MSE	0.665	0	0.062
SSIM	0.961	0.881	0.231
MS-SSIM	0.962	0.872	0.237
CW-SSIM	**0.977**	**0.962**	**0.261**

Table 6.5: Performance evaluation of the Anomaly Detection method based on CW-SSIM involving different loss functions and anomaly metrics, applied to the same autoencoder configuration.

in the second row (b) of Figure 6.7, where the pixel-wise l^2 metrics (which is MSE but not averaged) is applied. In the rest of the figures, the anomaly maps originated from the other experiments are reported. We can notice that the anomaly maps generated by SSIM (c) and MS-SSIM (d) are very similar, probably because they are produced with the same metrics. It is possible to realize that the CW-SSIM map (e) is much more defined with respect to the previous two, due to the averaging of CW-SSIM metrics at different window size L (see Section 4.4.4 for more details). In the end, we remark that these anomaly maps, related to the CW-SSIM configuration, show that the identification of the anomalies is more accurate. This is probably due to the analysis of the images at different scales and orientations using the Steerable filters decomposition.

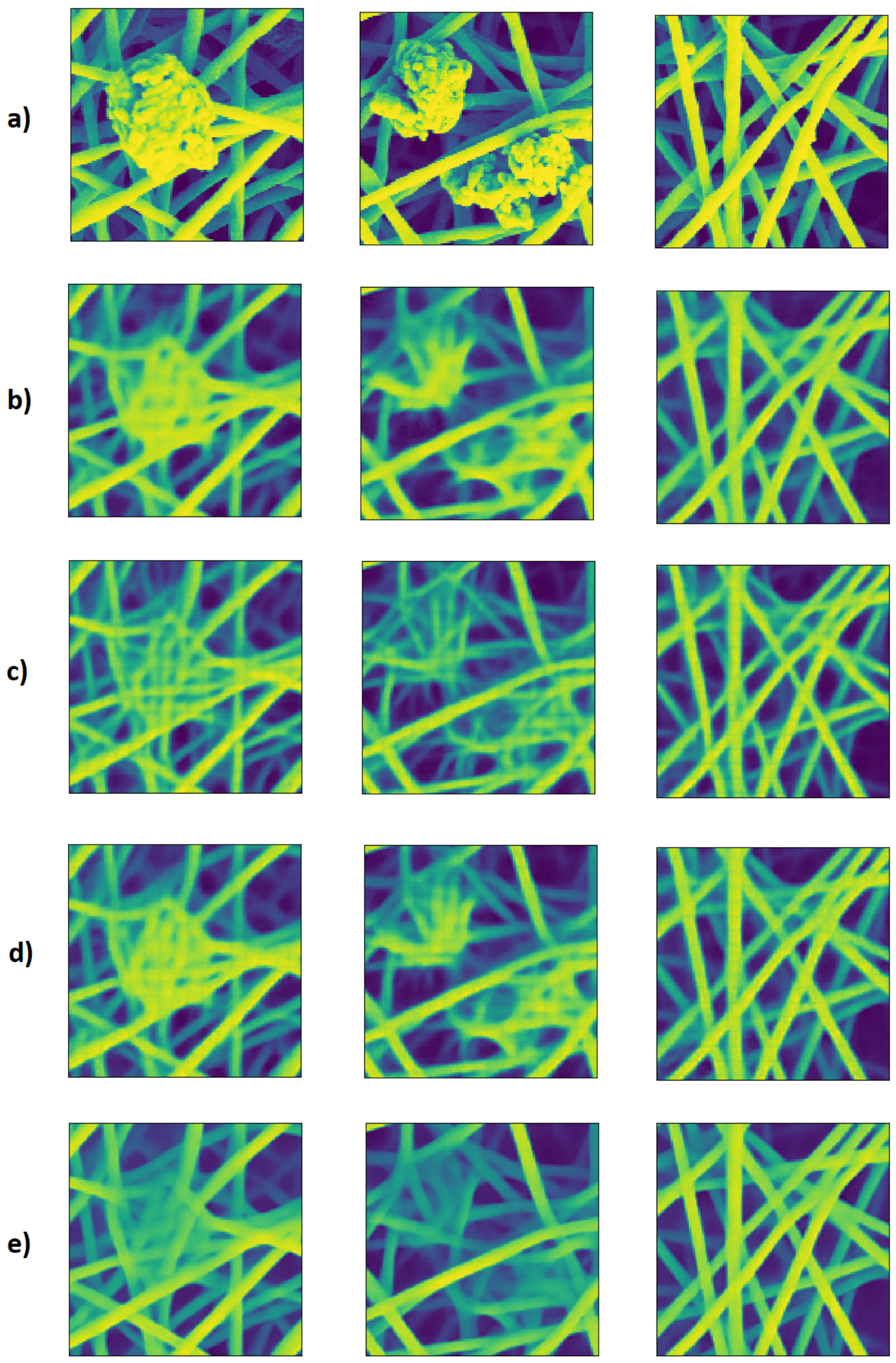

Figure 6.6: Comparison of 128×128 *patch reconstructed with different loss functions. a) Original patches. b) MSE Reconstructed patch. c) SSIM Reconstructed patch. d) MS-SSIM Reconstructed patch. e) CW-SSIM Reconstructed patch.*

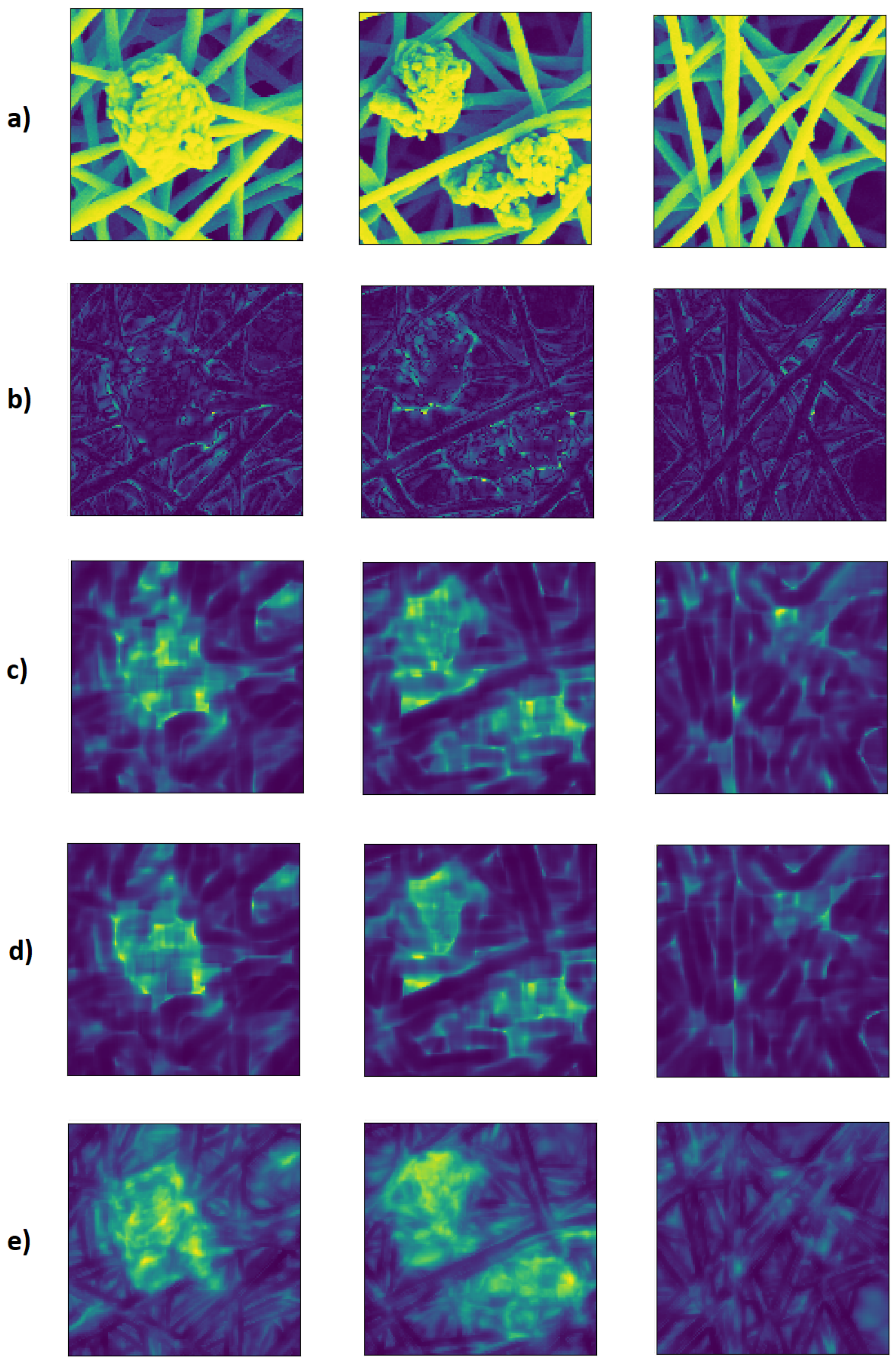

Figure 6.7: Comparison of 128×128 patch evaluation after applying the corresponding anomaly metrics between the original patches and the reconstructed ones shown in Figure 6.6. a) Original patches. b) Anomaly maps generated with l^2 metrics. c) Anomaly maps generated with SSIM metrics. d) Anomaly maps generated with MS-SSIM metrics. e) Anomaly maps generated with CW-SSIM metrics.

6.4.4 Final considerations

In this final section, we want to reconnect to the drawbacks of the STSIM based method and understand how they are solved. The first advantage is the use of the Steerable filters decomposition on large patches, which allows to extract more scale-related information. Then these features are not directly used as normal reference in the Anomaly Detection, but as anomaly metrics in an autoencoder. The benefit is that the features are learned forcing the autoencoder to reconstruct textures of nanofibrous materials, as perceptually identical to the input ones. Taking advantage of the way in which CW-SSIM works, it is possible to achieve fine-grained anomaly map resolutions, since the similarity metrics between images are computed in small windows. Moreover, we average the metrics that comes from the application of CW-SSIM at different windows size, increasing the robustness of the calculations. In a nutshell, we are able to train the model with larger patches, allowing more information extraction, and to evaluate the images in smaller windows increasing the anomaly map resolution.

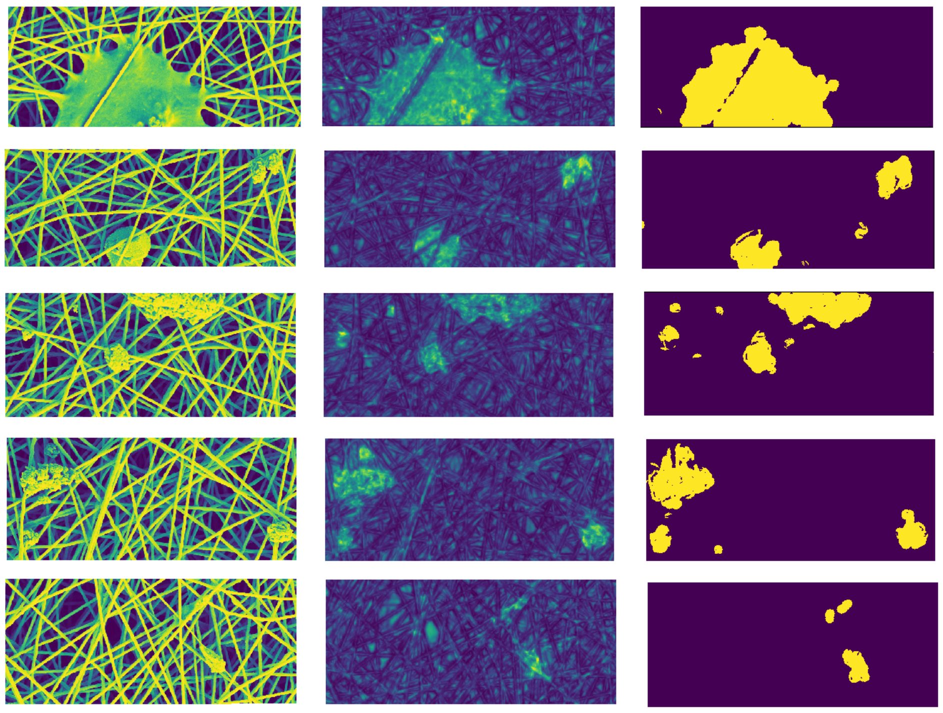

Figure 6.8: Portion of test images evaluated with the Anomaly Detection method implemented with CW-SSIM as loss function and anomaly metrics. From left to right we can find the original test image I, the anomaly map ω_I and the binary mask of anomalies $\hat{\Omega}_I$.

6.5 Comparison with the State of the Art

The Table 6.6 reports the three State of the Art performances and the ones achieved with our solutions using AUC metrics, Defect Coverage and IoU. We want to state that for the CNN-Based Self-Similarity performances in Table 6.5, we report the ones present in the paper since the code is not publicly available. For this reason, they are highlighted with an asterisk and only the AUC and Defect Coverage metrics are reported. Regarding the other two methods, instead, we repeat the experiments in our configuration and show the complete performances. From the evaluation table, we can clearly state that the STSIM based solution achieves worst performances with respect to the ones in literature, probably due to all its drawbacks. The CW-SSIM based method, instead, achieves comparable performances in the AUC scores, especially with respect to [20], but it outperforms by far all the other methods in the second metrics, covering more than 50% of the anomalies for more than 96% of their area. We want to finally report that this solution using the best configuration reaches an IoU of 0.326 using the threshold coming from FPR set to 3% in the validation set. In this settings, it reaches also the 0.92 in the Defect Coverage, which is still better than the State of the Art.

State of the Art performance comparison			
Method	AUC	Defect Coverage	IoU
Sparse Dictionary Learning [7]	0.926	0.65	0.223
CNN-Based Self-Similarity [20]	0.974*	0.85*	n.e.*
SSIM Applied to Autoencoders [3]	0.961	0.88	0.231
STSIM based method	0.818	0.24	0.126
CW-SSIM based method	**0.977**	**0.96**	**0.261**

Table 6.6: Performance comparison between the State of the Art and our solutions. We refer to Sparse Dictionary Learning to [7], CNN-Based Self-Similarity to [20] and SSIM Applied to Autoencoders to [3].

Both our two methods are trained and evaluated on a Windows 10 PC with an Intel i7-8700 3.20GHz $\times$ 12 CPU, 16 GB GDDR4-RAM and an NVIDIA GTX 1060 GPU with 4GB of dedicated RAM. The great advantage of the STSIM based procedure is that the training is very fast, since it takes on average 2.3 seconds for the KDE implementation and 1.3 seconds for the Mahalanobis one. This fact is really useful in an environment in which the normal textures continuously change, for example due to the degradation of the image source. The CW-SSIM based method, instead, takes about 11 hours to complete the training procedure described in the implementation Section 5.2. Table 6.7 reports the evaluation time of our solutions, compared with the ones in literature. We can notice that ours and [3], which are based on autoencoders, are much faster with respect to the previous two [20] [7]. This is for sure due to the avoidance of any learning during evaluation and, in particular for the second method involving the autoencoder, the execution of the most computational demanding operations in the GPU. The evaluation time of the STSIM method is very low, and very similar for both the implementations; the reason could be that we based our evaluation on few training data. Regarding the second method, we can realize that there are some differences between our evaluation time and the one in [3]; this could be related to the different autoencoder architecture, patch striding and evaluation procedure, but also for the GPU used.

Evaluation time comparison	
Method	Evaluation time
Sparse Dictionary Learning [7]	$50s \sim 55s$
CNN-Based Self-Similarity [20]	$15s \sim 55s$
SSIM Applied to Autoencoders [3]	$3.61s$
STSIM based method	$0.02s$
CW-SSIM based method	$13.73s$

Table 6.7: Evaluation time comparison between the State of the Art and our solutions. We use the same reference of Table 6.6.

6.6 Anomaly Detection Experiments based on Autoencoder Latent Distribution

The use of the reconstruction error, as anomaly score in autoencoder based methods applied to Anomaly Detection, is not mandatory. Many researches focus their attention upon the latent space, namely the distribution of the features in the bottleneck. Upon them we can cite some works based on Variational Autoencoders [1] [32], introduced in Section 2.2.4, which regularize their latent space in order to follow a prior distribution during training. In this case, the anomaly score is computed comparing the posterior distribution produced by test patches with respect to the prior one. Another similar approach [40] models the latent space of an autoencoder using the *Gaussian Mixture Model* (GMM) technique. Basically, GMM is a probabilistic model which assumes that all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering [2] to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians [26]. In [40], the authors jointly learn both the parameters of the deep autoencoder and the mixture model simultaneously in an end-to-end fashion. In this case, the anomaly metrics of a test patch is evaluated through the GMM, which predicts its posterior probability.

6.6.1 Experimental procedure

In these experiments, we take inspiration from [40], easing its algorithm in order to generate a pixelwise Anomaly Detection, based on the latent representation of an autoencoder. We have decided not to add this method among the core ones, since we have studied it less than the others, achieving very poor results. Furthermore, for the largest part of the experiments, we report only qualitative results, since we believe that they explain better than any other performance metrics the outcome that we have achieved.

This procedure, somehow, can be seen as a compromise between the STSIM and CW-SSIM based techniques. The training phase is very similar to the one in CW-SSIM based method: the autoencoder's parameters are learnt only with patches coming from anomalous-free images with the CW-SSIM loss function. In this phase, however, we do not use the entire training set. To ease the procedure, we separate the autoencoder training with the KDE one, used here as Anomaly Detection algorithm (as GMM in [40]). Once the autoencoder is trained, we extract the latent features providing the second normal set as input to the autoencoder, fit them to KDE and learn its parameters. The evaluation procedure follows exactly the STSIM based one: each test image is divided in patches, the anomaly score of each of them is computed through KDE and finally the anomaly map is built merging all the evaluations.

6.6.2 Latent Space Regularization

In the first experiment, we try to use the best autoencoder configuration of CW-SSIM based method. We recall that this autoencoder accepts input patches of 128×128, deploying a latent dimension of 512 features. The CW-SSIM has 5 orientations, 5 scales and a window size of $7px$. Then, we fit KDE with 2500 features vectors coming from the same amount of normal patches. Regarding the evaluation, we extract the patches from test images with a stride equal to 1, in order to have the highest possible resolution in the anomaly map. At this point, the latent features of test patches are computed and KDE calculates their posterior.

In order to qualitative evaluate the performances, we test this method on the abnormal image number 8 which presents both large and small anomalies. From Figure 6.9, which reports the anomaly map generated following the above procedure, we can clearly see that the latent dimension does not seem regular with respect to normal features.

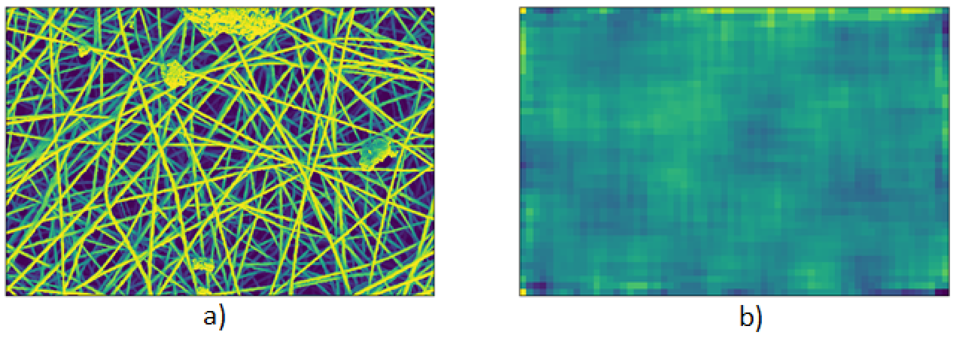

Figure 6.9: Latent-Based Anomaly metrics evaluation. Experiment number one: use the same autoencoder configuration of the best CW-SSIM based method. a) Test image. b) Anomaly map.

The idea is to regularize the latent distribution, as suggested in [25], adding a penalty term to the loss function. We add an l^2 regularization to the latent dimension, rewriting the loss function as:

$$\mathcal{L}_{reg}(x, y) = L_{cw}(x, y) + \sum_{j=0}^{a} z_j^2, \tag{6.5}$$

where z is the latent vector ($E(x) = z$) and a is the latent dimension (equal to 512 in this case).

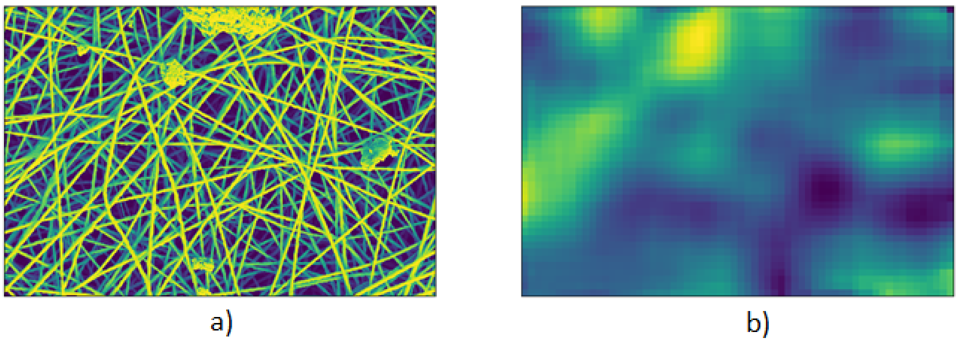

Figure 6.10: Latent-Based Anomaly metrics evaluation. Experiment number two: use the same autoencoder configuration of the best CW-SSIM based method but with (6.5) *as loss function. a) Test image. b) Anomaly map.*

Figure 6.10 shows that we have effectively regularized the anomaly map assigning almost the same posterior to every normal patch, but still we have a low resolution in the anomaly map.

6.6.3 Increase Anomaly Map Resolution

The problem of producing a coarse anomaly map has been already analyzed in Section 6.4, when we have discussed the performance evaluation of the STSIM based method. We recall that this problem rises because we assign, for each patch of 128×128 in input, a single anomaly score. This score has to be finally upscaled to the patch size, producing a low resolution anomaly map. In that case, the research for the best trade-off on patch dimension is crucial, in order to produce high resolution map without loosing information. The substantial difference here is that we have replaced the STSIM feature extraction with an autoencoder that provides feature information as well.

One of the possible solutions is to increase widely the latent spatial dimension, moving from a latent vector to a three dimensional one. This modification allows to generate a number of features vectors, for each input patch, that is proportional to the spatial dimension of the latent space. Each features vector in the latent cuboid is then evaluated through KDE, producing a bi-dimensional matrix of anomaly scores. At this point, upscaling a score matrix would generate a more accurate anomaly map with respect to a single score. The right latent dimension, however, is not easy to find: increasing only the spatial dimension would generate too much latent features (the number of features is given by the volume of the cuboid), leading the autoencoder not to accurately compress the information. One should consider to reduce the channel dimension, but then the accuracy of KDE could be compromised. In Figure 6.11, it is shown the anomaly map produced using an autoencoder with a latent dimension of $8 \times 8 \times 64$ (for a total of 4096 features). We can clearly notice that even if the resolution is increased, the autoencoder was not able to extract the features of the normal textures due to a large latent space. Some other experiments were deployed with different configurations but without particular gaining. One of the biggest limitations is the large training time (almost half a day on our machine) that slows down particularly these researches.

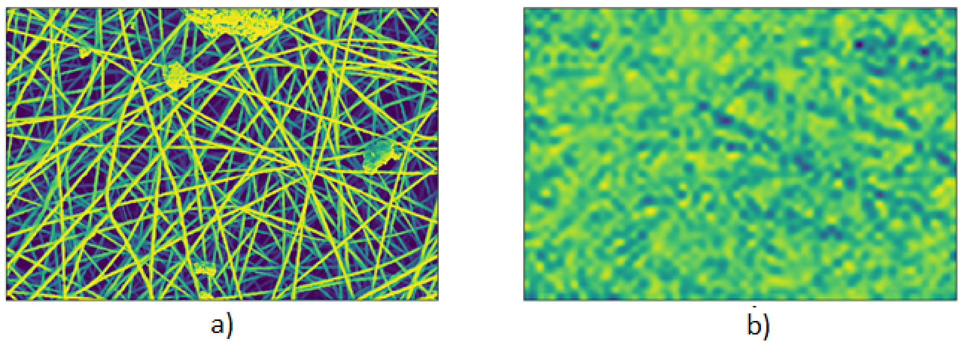

Figure 6.11: Latent-Based Anomaly metrics evaluation. Experiment number three: use an autoencoder with (6.5) *as loss function and a latent dimension of* $8 \times 8 \times 64$. *a) Test image. b) Anomaly map.*

Another possibility is, following the same reasoning of the STSIM based solution, to reduce the input patch size. In this case, we also need to reduce the latent dimension in order to keep a certain compression factor among the two. In Figure 6.12, it is reported the anomaly map using an autoencoder that accepts patches of 32 pixels wide and with a latent space of 64 features ($1 \times 1 \times 64$). This experiment allows to reach the best qualitative performance in this approach. We also report that it achieves an AUC (on image number

8) of 0.689 that is much smaller than the ones achieved with the STSIM based method (0.849) and CW-SSIM based solution (0.987), which uses the reconstruction error as anomaly metrics.

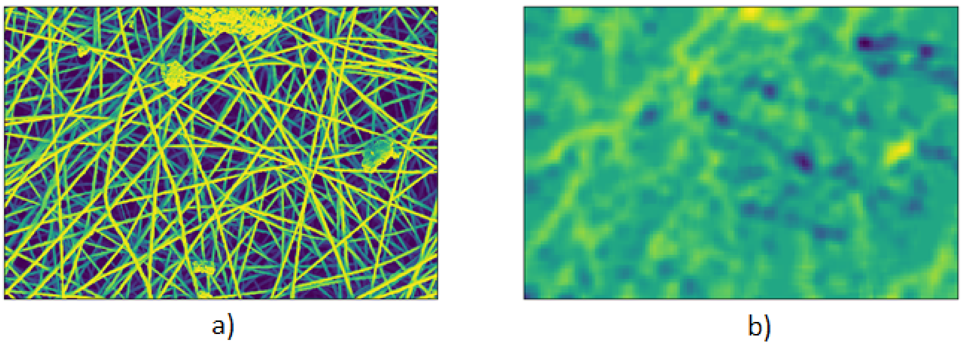

Figure 6.12: Latent-Based Anomaly metrics evaluation. Experiment number four: use an autoencoder with (6.5) *as loss function, input size of* 32×32 *and a latent dimension of 64. a) Test image. b) Anomaly map.*

The reasons for this poor performance could be many and regard both the autoencoder architecture and KDE parameters. In this section, we want to focus the attention only on the autoencoder choice. Figure 6.13 reports the reconstructed image produced merging all the decoder outputs. We can realize that, even if the reconstruction seems to be very accurate, the anomalies are reported without modifications, with respect to the previous method (see for example Figure 6.6) in which the anomalies are replaced with filamentous textures. This issue could be related either to a large latent dimension deployed, or to the possibility that the autoencoder is not able to extract features of normal textures from too small patches. As a final observation, we could also argue that having a reconstructed image with filamentous textures in place of anomalies is not helpful: it might mean that the only features extracted are the ones of normal textures, thus KDE will be unable to distinguish normal from abnormal patches.

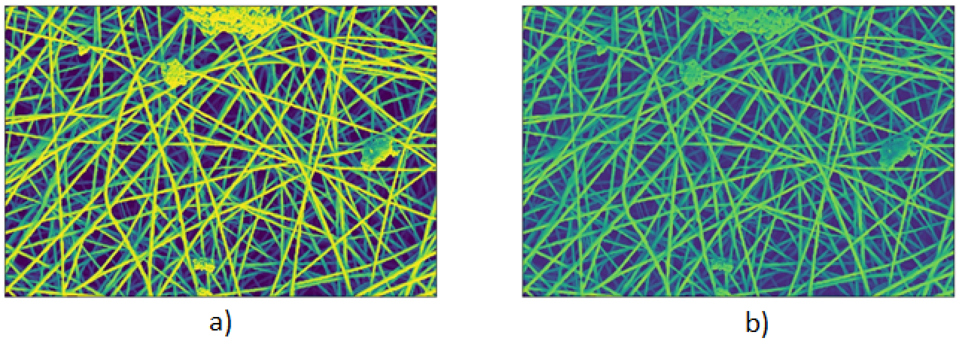

Figure 6.13: Latent-Based Anomaly metrics evaluation. Reconstructed image of experiment number four: using an autoencoder with (6.5) *as loss function, input size of* 32×32 *and a latent dimension of 64. a) Test image. b) Reconstructed image.*

Chapter 7

Conclusion and Future Work

In this thesis, we proposed two Semi-Supervised methods based on the Steerable filters decomposition to detect anomalies in textured images. The goal is to generate the full-resolution anomaly map of the input image, which classifies each pixel as normal or anomalous. Both of our two solutions are based on a patch-wise training procedure, in which the features that characterize normal textures are extracted. The Anomaly Detection method based on STSIM exploits the Steerable filters decomposition to extend the expressiveness of the images, looking for the response of different oriented convolutions and subsamplings. These responses, also referred to as sub-bands, are used to extract the features using STSIM which combines single- and multi-bands statistics. The rationale behind this procedure is that features extracted from normal patches should follow a certain distribution, while the ones from the anomalous patches might not follow the same distribution. The anomaly score of a test patch is achieved by calculating the marginal probability that its features belong to the normal distribution. In our method, we use the Mahalanobis distance and KDE to build the density model that provides this anomaly score. Our second method consists in an Anomaly Detection method based on autoencoder that uses CW-SSIM as loss function. This solution relies upon a different concept: instead of forcing the algorithm to extract a particular type of features, we train a Deep Neural Network to extract the features in a Semi-Supervised fashion. In this case, the neural network is trained, given in input a training patch, to reconstruct itself minimizing CW-SSIM. This similarity metrics exploits the Steerable filters decomposition, focusing on the perceptual similarity between patches with respect to their pixelwise differences. The general idea is that the anomalous blobs in the test patch are not perfectly reconstructed by the autoencoder, since it tries to generate normal textures in their place. The more the original test patch is different with respect to the reconstructed one, the higher the probability that it contains an anomaly is. CW-SSIM conducts two key roles in this method: the first one is to force the autoencoder to generate textures as similar as possible to the normal ones, especially in place of anomalies. The second role is to calculate the pixelwise similarity in the anomaly evaluation that, as we have proven in our experiments, outperforms the methods in literature. Both of our two solutions merge the anomaly scores of each test patch to create the anomaly map.

The first method, as we have reported in performance evaluation Section 6.3, suffers mostly the patch dimension chosen, and a trade-off partially mitigates its drawbacks. Furthermore, the training patches indicated are really few to incorporate all the normal texture possibilities, but empirical results proved us that increasing them would decrease dramatically the sensitivity of the anomaly metrics. This is something that in future works should be investigated further. With the second Anomaly Detection method, instead, we outperform the results in the literature but still achieve very low performances in the IoU metrics. The Figure 6.8 clearly shows that there are several false indications in the anomaly maps, that force us to use a big morphological operator (10 pixel) to get rid of the smallest ones. Deeper investigations report that most of them are produced by the low similarity score between the input and the reconstructed patch's background. The reason is that the autoencoder seems to have great difficulty in the generation of dark areas, and thus forces us to eliminate anomalies in the binary mask concerning too dark pixels. In the end, we have to report that sometimes the training procedure ends in such a way that the learned autoencoder produces inverted reconstructions. This implies that one should check this possibility, and if it is the case either retrain the whole model or simply multiply the output of the autoencoder by minus one.

In future works, we propose to focus on the Anomaly Detection method applied to autoencoders, since it can open many more paths. In the development of this thesis, we tried to use STSIM also as loss function of the second proposed method, but it achieved poor results. This is something that could be investigated further, because STSIM comes as an extension of CW-SSIM, achieving in literature better results in textured image classification [41]. Another possible path could be to investigate further the experiments in Section 6.6, in which we tried to exploit the latent representation of the autoencoder. The first step could be to investigate the issues left opened by our considerations in order to clarify the aspects related to the patch and the latent dimension. As an improvement, we propose to follow the idea in [40], in which the autoencoder and the anomaly metrics are trained together in an end-to-end fashion.

Bibliography

[1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2:1–18, 2015.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[3] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Carsten Steger. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. *arXiv:1807.02011*, 2018.

[4] Giacomo Boracchi, Diego Carrera, and Brendt Wohlberg. Novelty detection in images by sparse representations. In *2014 IEEE Symposium on Intelligent Embedded Systems (IES)*, pages 47–54. IEEE, 2014.

[5] Alan C. Bovik, Marianna Clark, and Wilson S. Geisler. Multichannel texture analysis using localized spatial filters. *IEEE transactions on pattern analysis and machine intelligence*, 12:55–73, 1990.

[6] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Now Publishers Inc, 2011.

[7] Diego Carrera, Fabio Manganini, Giacomo Boracchi, and Ettore Lanzarone. Defect detection in sem images of nanofibrous materials. *IEEE Transactions on Industrial Informatics*, 13:551–561, 2016.

[8] Florence Denis, Franck Davignon, and Atilla Baskurt. Steerable pyramid for contrast enhancement and directional structures detection. In *2002 11th European Signal Processing Conference*, pages 1–4. IEEE, 2002.

[9] Google Developers. Classification: ROC curve and AUC. Accessed: 2021-02-01.

[10] William T Freeman, Edward H Adelson, et al. The design and use of steerable filters. *IEEE Transactions on Pattern analysis and machine intelligence*, 13:891–906, 1991.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[12] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv:1406.2661*, 2014.

[13] IBM. Ibm cloud learn hub neural networks. Accessed: 2021-01-07.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[16] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22:79–86, 1951.

[17] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*. Citeseer, 2013.

[18] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[19] NanoTWICE. Nanotwice: Nanocomposite nanofibres for treatment of air and water by an industrial conception of electrospinning. Accessed: 2021-01-21.

[20] Paolo Napoletano, Flavio Piccoli, and Raimondo Schettini. Anomaly detection in nanofibrous materials by cnn-based self-similarity. *Sensors*, 18:209, 2018.

[21] Jim Nilsson and Tomas Akenine-Möller. Understanding ssim. *arXiv:2006.13846*, 2020.

[22] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[23] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33:1065–1076, 1962.

[24] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40:49–70, 2000.

[25] Sheng Qian, Guanyue Li, Wen-Ming Cao, Cheng Liu, Si Wu, and Hau-San Wong. Improving representation learning in autoencoders via multidimensional interpolation and dual regularizations. In *IJCAI*, pages 3268–3274, 2019.

[26] Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741:659–663, 2009.

[27] Simone Calderara Rita Cucchiara and Davide Abati. AImage Lab. University of Modena and Reggio Emilia. Accessed: 2021-02-27.

[28] 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. PMLR, 2018.

[29] Mehul P Sampat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik, and Mia K Markey. Complex wavelet structural similarity: A new image similarity index. *IEEE transactions on image processing*, 18:2385–2401, 2009.

[30] 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.

[31] Eero P Simoncelli, William T Freeman, Edward H Adelson, and David J Heeger. Shiftable multiscale transforms. *IEEE transactions on Information Theory*, 38:587–607, 1992.

[32] Daniel Soukup and Thomas Pinetz. Reliably decoding autoencoders' latent spaces for one-class learning image inspection scenarios. In *OAGM Workshop*, 2018.

[33] Stanford.edu. Cs231n: Convolutional neural networks for visual recognition. Accessed: 2021-01-19.

[34] Gilbert Strang. Wavelets and dilation equations: A brief introduction. *SIAM review*, 31:614–627, 1989.

[35] D-M Tsa and S-K Wu. Automated surface inspection using gabor filters. *The International Journal of Advanced Manufacturing Technology*, 16:474–482, 2000.

[36] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184, 11 2015.

[37] Z. Wang, Eero Simoncelli, and Alan Bovik. Multiscale structural similarity for image quality assessment. *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, 2:1398 – 1402 Vol.2, 12 2003.

[38] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13:600–612, 2004.

[39] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2:37–52, 1987.

[40] 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.

[41] Jana Zujovic, Thrasyvoulos N Pappas, and David L Neuhoff. Structural texture similarity metrics for image analysis and retrieval. *IEEE Transactions on Image Processing*, 22:2545–2558, 2013.

Appendix A

User Manual

The code of two methods presented in this thesis is publicly available on the GitHub website under the Anomaly Detection based on STSIM[1] and the Anomaly Detection based on CWSSIM[2] repositories. Both of them are developed in Python 3.7, where the Steerable filters decomposition's functionalities are implemented with TensorFlow 2.1.0. A list of all the other minor dependencies can be found in the main page of each repository. Both of them present a really simple Command Line Interface (CLI), which asks to provide the configuration file with the training and/or evaluation parameters.

In the STSIM based implementation, the CLI wants two parameters in input: using the prefix *-t*, it is possible to define the anomaly metrics to use, specifying one of Mahalanobis or KDE. After the prefix *-f*, instead, the path to the configuration file is required. Both the training and the evaluation parameters are required to be in a single file. Once both of them are specified, the software starts to train over normal patches and then evaluates one or more test images, based on the configuration file.

```
$ python Main.py [-h] [HELP] -t TYPE -f FILE
```

In the CW-SSIM based implementation, we have decided to divide the training from the evaluation procedure, since the first is particularly time demanding. For this reason, using the prefix *-a*, the CLI accepts the action to perform specifying one between training or evaluation. As before, after the prefix *-f*, the path to the configuration file is required. In this case, the

[1]https://github.com/AndreaBiondaPolimi/Anomaly-Detection-method-based-on-STSIM.git

[2]https://github.com/AndreaBiondaPolimi/Anomaly-Detection-method-based-on-CW-SSIM-applied-to-Autoencoders.git

training and evaluation parameters should be in different files; we have made this choice only for better usability. After the CLI input, the software starts performing the specified action. The documentation of each configuration file can be found inside the files themselves, under *Configuration* repositories' folder. All the default configuration files report the parameters of the best experiment.

```
$ python Main.py [-h] [HELP] -a ACTION -f FILE
```