University of Padua

Department of Information Engineering

Master Thesis in Computer Engineering

GAN-based defect transfer model for steel surface defects generation

Supervisors
Prof. Emanuele Menegatti
University of Padua

Stefano Totaro
Mermec Spa

Co-supervisor
Alberto Bacchin
University of Padua

Master Candidate
Francesco Gazzola

Academic Year : 2022-2023
Graduation Date : 27-11-2023

Abstract

Over the last years we have witnessed the use of many deep learning approaches in the industry and in particular in the field of defect detection. However, these techniques require large amount of data of defective samples which are not always available for all the defect classes and the materials. To address the problem of insufficient defect samples in the task of steel defect detection, we have proposed a GAN model which is based on a swapping autoencoder. Our solution disentangles the defect-specific content from the background and transfer the content from a steel surface with a specific texture to a new steel surface with a different texture by swapping the texture code. These synthetic defective samples enlarge the dataset and increase the diversity of defects on the target steel item. Through the experiments we could demonstrate that our solution can be used as a data augmentation tool improving thus the accuracy of a defect classifier.

Sommario

Negli ultimi anni abbiamo potuto assistere all'impiego di molte strategie basate sul deep learning nel settore industriale e in particolare nel campo del rilevamento dei difetti. Tuttavia, queste tecniche richiedono una grande quantità di dati di campioni difettosi che non sono sempre disponibili per tutte le classi di difetti e materiali. Per affrontare quindi il problema dell'insufficienza della quantità delle immagini di difetti nell'attività di rilevamento dei difetti dell'acciaio, abbiamo proposto un modello GAN basato sulla scomposizione dell'input in due codifiche. La nostra soluzione si impegna quindi a separare l'informazione che specifica la forma del difetto dall'informazione che identifica invece la texture della superficie. Grazie a tale sbrogliamento della rappresentazione dell'immagine, scambiando la codifica di una texture con la codifica di un'altra texture possiamo trasferire il difetto da una superficie in acciaio con una trama specifica a una nuova superficie in acciaio con una trama diversa. Questi campioni sintetici difettosi vanno quindi ad ampliare il set di dati e aumentano la diversità dei difetti su un oggetto in acciaio. Dagli esperimenti effettuati abbiamo potuto dimostrare che la nostra soluzione può essere utilizzata come strumento per la data augmentation, migliorando quindi l'accuratezza dei sistemi di classificazione dei difetti.

Contents

Abstract **iii**

Sommario **v**

List of figures **xi**

1 Introduction **1**

1.1 GAN overview 4

1.1.1 Metrics 6

1.2 Image to Image translation overview 8

1.3 Disentangled representation in the image domain 11

1.3.1 Content-Style disentanglement 12

2 Related works **15**

3 Methodology **19**

3.1 Swapping autoencoder 19

3.2 Network architecture 21

3.2.1 Encoder 21

3.2.2 Generator 21

3.2.3 Multi-task Discriminator 22

3.2.4 Patch discriminator 23

3.2.5 Classifier 24

3.3 Training objectives 25

3.3.1 Anchor domain hypothesis 25
3.3.2 Losses 26

4 Experiments **31**
4.1 Training 31
4.2 Dataset 32
4.3 Results 36
4.3.1 Quantitative evaluation 36
4.3.2 Qualitative evaluation 37
4.3.3 Anchor domain hypothesis verification 45
4.3.4 Data augmentation 51
Dataset 51
Training 52
Results 55

5 Conclusion **57**

References **59**

Listing of figures

1.1 GAN architecture 5

1.2 FID is evaluated for upper left: Gaussian noise, upper middle: Gaussian blur, upper right: implanted black rectangles, lower left: swirled images, lower middle: salt and pepper noise, and lower right: CelebA dataset contaminated by ImageNet images. The FID captures the disturbance level very well by monotonically increasing. Image is taken from [1]. 8

1.3 Example of single-modal multi-domain I2I on faces [2] 10

1.4 Example of multi-modal two-domain I2I [2] 11

1.5 Cycle consistency illustration between two domains A and B [2] . . 11

1.6 Illustration of some entangling techniques [3] 13

3.1 Original swapping autoencoder architecture 20

3.2 Encoder and generator 22

3.3 Discriminator architecture 23

3.4 Co-occurence patch discriminator 24

3.5 Overall architecture 25

4.1 Example of the cropping process of an original image representing a tear defect class 33

4.2 Training set samples about defective images. Each row contains defective images of the same class. From the first row to the last: mark, shell, scoring, scratch, notching, tear 34

4.3 Training set samples about the normal samples 35

4.4 Highlighted in red is the suspected case of mode collapse. Given the texture reference image in the the orange box, the model reconstructs it as shown in the yellow box. However its reconstruction is identical to another texture image in the training dataset (shown in the blue box). 38

4.5 Mode collapse study. The first row reports the original image, while the second row represents the corresponding generated image. We can see that although all the images share the same style and texture, only the image at the top in the red box is reconstructed wrongly. . 38

4.6 Generated samples with mark defects. First row represents the defect reference images, while the first column represents the texture reference images. 39

4.7 Generated samples with notching defects. First row represents the defect reference images, while the first column represents the texture reference images. 40

4.8 Generated samples with shell defects. First row represents the defect reference images, while the first column represents the texture reference images. 41

4.9 Generated samples with tear defects. First row represents the defect reference images, while the first column represents the texture reference images. 42

4.10 Generated samples with scoring defects. First row represents the defect reference images, while the first column represents the texture reference images. 43

4.11 Generated samples with scratch defects. First row represents the defect reference images, while the first column represents the texture reference images. 44

4.12 Texture reconstruction for the mark defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image. 45

4.13 Texture reconstruction for the notching defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image. 46

4.14 Texture reconstruction for the shell defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image. 47

4.15 Texture reconstruction for the tear defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image. 48

4.16 Texture reconstruction for the scoring defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image. 49
4.17 Texture reconstruction for the scratch defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image. 50
4.18 No augmented vs augmented dataset 52
4.19 Loss ResNet50 classifier on no augmented data 53
4.20 Loss ResNet50 classifier on augmented data 53
4.21 Accuracy curves of the ResNet50 classifier on no augmented data . 54
4.22 Accuracy curves of the ResNet50 classifier on augmented data . . . 54

1

Introduction

In today's fast-paced manufacturing and quality control processes, identifying and mitigating defects in steel is of paramount importance. Defects can lead to safety hazards, reduced product performance and quality, and significant financial losses. Indeed, steel is one of the most important raw materials in the industries and its quality directly affects the final performance of the industrial products. During its production process, different defects such as scratches and cracks usually appear due to the different processing techniques and the rolling equipment [4]. This has drawn importance on the employment of automatic processes for detecting and classifying surface steel defects based on machine vision.

Traditional defect detection methods often require labor and time. Indeed, the earlier methods were based on using threshold or machine-learning which required the collection of handcrafted features and a classifier [5]. The major drawback of this strategy is that they lack generalisation ability and robustness i.e. it is often

difficult to adapt it to complex and changing industrial scenario [6]. Indeed for different types of defects it requires the development of different algorithms [5].

The advent of deep learning has paved the way for more efficient and accurate defect detection techniques. Indeed, a deep learning-based model detection technique makes use of a neural network to automatically learn defect features from a large amount of data [6]. However, such techniques require a large number of images and label information. In particular, while it is simple to get sufficient defect-free samples, the number of defective samples in an industrial production line is usually small making it difficult to build a defect dataset with high quantity and diversity. As consequence, this limits the performance of a recognition model and poses a challenge to the detection task [7][6].

To overcome this issue, data augmentation method are often employed to enrich the training dataset by introducing different kind of invariance for the model to capture [6]. Different works have demonstrated that the generated defect data can significantly improve the accuracy of the defect detection models [6].

Traditional data augmentation techniques that are commonly adopted in deep learning are usually based on geometric transformations such as rotation, flips and so on. However, such methods can only enhance the existing dataset without supplementing other faulty patterns with additional information [6]. To resolve this, new defect-image synthesising approaches have been proposed which are able to generate images with various defect types, shapes, size and locations [6].

Before the deep learning era, the defect-image synthesising methods were mainly based on manually destroying defect-free workpieces, use computer aided design (CAD) or using digital image processing techniques [7]. However, such methods work well only for simple defects but real defect images tend to have complex texture [6].

In recent years, GAN model has gained significantly attention in the field of im-

age generation since its ability to generate realistic images [6]. Moreover, different GANs model has been designed over the last year which have achieved remarkable performance in natural image generation [5].

Generative Adversarial Networks are a class of artificial neural networks that have gained remarkable prominence in the field of artificial intelligence. They were introduced by Ian Goodfellow and his colleagues in 2014 and have since been applied in various domains, including image generation, natural language processing, and, notably, defect generation. The ability to create synthetic defects with GANs has far-reaching implications for a variety of industries, including manufacturing, medical imaging, and autonomous systems.

Defects generation using GANs involves the development of a generative model and a discriminative model that work in tandem to create images containing defects, whether in the form of anomalies, imperfections, or irregularities. The generative model strives to produce counterfeit defects, while the discriminative model attempts to distinguish between real and synthetic defects. This adversarial process leads to the refinement of the generative model over time, resulting in progressively more convincing and authentic defect images.

The advantages of using GANs for defect generation are manifold. It offers a scalable, cost-effective, and efficient way to generate diverse and controlled defect datasets. This, in turn, aids in the development and improvement of defect detection algorithms and systems. Indeed, it facilitates rigorous testing and evaluation of such systems under various defect scenarios, enhancing their reliability and robustness. Moreover, compared to the aforementioned methods, GANs can generate images at high speed.

However, as stated in [7] using GANs for defect generation is still challenging since they require a sufficient amount of data, and they might not be able to cover the entire variety of defects in terms of shape, location and color.

We tried to overcome these challenges proposing an image-to-image defect-generation method based on an existing algorithm that leverages on the disentangled learning in the field of the steelwork. Our work is thus carried out for addressing the lack of industrial defect images. We implemented a disentangled learning approach to strive to get a better texture representation of the surface since metal surfaces are known to have complex and noisy background [8].

Moreover, we try to propose a multi-domain defect transfer framework addressing one of the challenges affecting image-to-image translation that is the lack of simultaneous training for multi-domain translation with a single network [9].

The next sections are organized as follows. In section two we review the state of the art in defect generation. Then in the third chapter we give an overview of our framework defining the architecture and the training process. In the fourth chapter we present and discuss the results we got through the experiments to validate the quality of our images. Moreover we demonstrate that our model can be used efficiently as data augmentation tool for improving the accuracy of a classifier.

1.1 GAN overview

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow and his colleagues in 2014 [10], have emerged as a powerful tool for defect generation. GANs consist of two neural networks, the generator and the discriminator, which are engaged in a continuous adversarial learning process. The generator network creates synthetic data, while the discriminator network evaluates the authenticity of the generated samples. As the generator strives to produce more convincing defects, the discriminator becomes better at distinguishing between genuine and synthetic defects.

A GAN learns to achieve the aforementioned tasks by the employment of some

losses. The choice of the losses is crucial as it determines the overall performance of the GANs in terms of stability of the training, diversity and high quality of the generated images. The core loss is called adversarial loss and was introduced by Goodfellow [10]. It is formulated as follows :

$$min_G max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))]$$

As the above loss indicates, D and G play a two-player minimax game. In other words, we train D to maximize the probability of assigning the correct label to both the training images and the generated samples. Simultaneously G is trained to minimize $log(1-D(G(z)))$ i.e. the likelihood that a generated samples is predicted as fake by the discriminator.

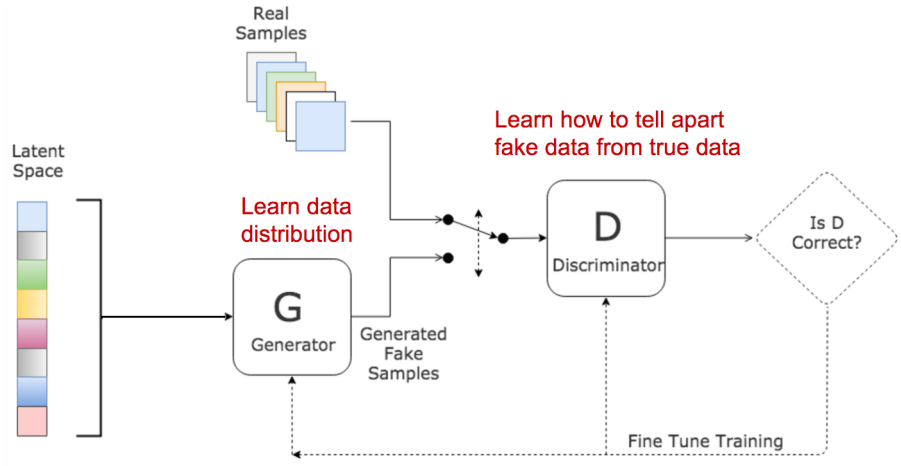


Figure 1.1: GAN architecture

However, the original adversarial loss might make the generator network suffer of the vanishing gradient problem which in turn makes the training difficult to converge. Indeed, in this scenario the discriminator will win the game before the generator. In other words, the discriminator will predict as fake all the samples coming from the generator as they are clearly different from the training data. For overcoming

this issue, Ian Goodfellow proposed a different formulation of the adversarial loss only for the generator that in the literature is referred as 'non-saturating loss' [10]. Such loss is based on training G to maximize $logD(G(z))$ instead of training G to minimize $log(1-D(G(z)))$ as showed previously.

The vanishing problem is not the only limitation preventing GANs to be applied to a practical scenario. Generative models suffer of the mode collapse problem, which makes the generator generate always the same sample class. This represents an issue since we usually would like GAN to produce diverse images. However this issue can be overcome setting up an appropriate loss such as the Wasserstein cost function adopted in WGAN [11]. Moreover, a scarce dataset might represent an issue as well since GANs tend to overfit when they are trained on little data [12]. To prevent overfitting and stabilizing the training it is possible to use some regularization techniques.

GANs are distinguished into conditional and unconditional GANs. The former uses additional information such as image, text, label and so on for guiding the generator to produce desired results [10]. The latter instead lets the generator generate random samples.

So far GANs have been employed in different tasks such as image super resolution, image restoration, multi-modal image synthesis, image translation, image inpainting, image editing and so on [6].

1.1.1 Metrics

Another challenge about generative models is the lack of an efficient metric for evaluating the quality of the generated images [13]. In the domain of GAN, for quality we mean both the fidelity, i.e. the realness of the generated images and the diversity, i.e. how well the generated images cover the whole variety of the real distribution.

The most common evaluation protocols is to use the Frescet Inception Distance (FID) and the Kernel Inception Distance (KID). Both metrics are based on measuring the distance between the real distribution and the generated distribution. In particular the comparison is made on the features of the real and the synthetic images which are extracted by the Inception-v3 network pre-trained on ImageNet. As regards FID, for both the real and the generated images, a Gaussian distribution is fitted to the extracted features. Then, the FID score is computed as:

$$FID = ||\mu_r - \mu_g||^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

where $\mathcal{N}(\mu_g, \Sigma_g)$ and $\mathcal{N}(\mu_r, \Sigma_r)$ are the Gaussians fitted to the generated and real data respectively.

FID does depend on the features extracted. Usually it is used the pool3 layer of the Inception-v3, but the choice of the layer can vary and it usually depends on the dataset size. For instance, pool3 layer can be used when the number of images is greater than 2048 in order to be able to compute the covariance matrix [14].

Regarding KID, it aims at improving FID by relaxing the Gaussian assumption. In particular KID is more suitable for small datasets [12] and it is measuring the mean discrepancy between the extracted features of the real and the generated samples:

$$KID = ||\mu_r - \mu_g||^2$$

A lower FID and a lower KID indicates a better match between the generated images and the real images in terms of their visual quality and diversity. We show in the Figure 1.2 how the FID score varies as the image appearance changes. The same illustration holds for KID.

Another evaluation tool is the human judgement. This is achieved by carrying out a survey where we ask to annotate if the synthetic images appear real or fake, or

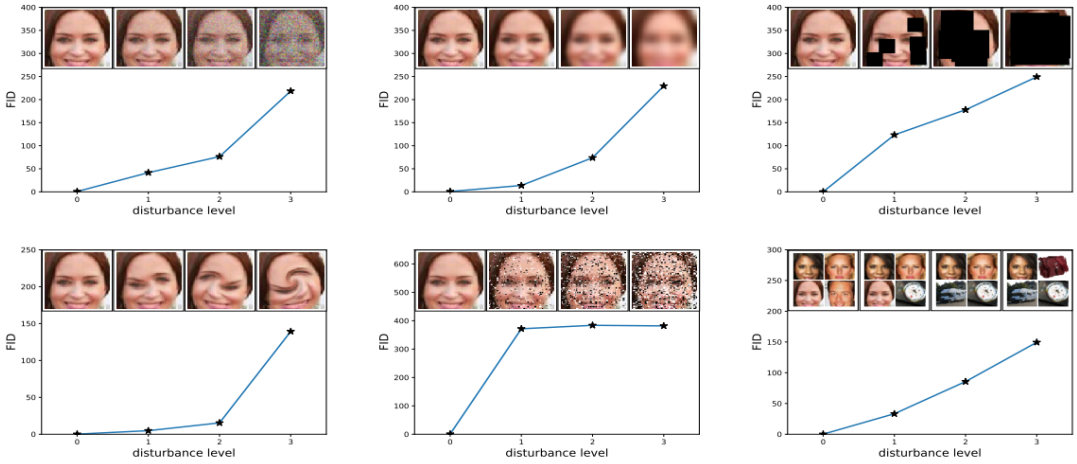


Figure 1.2: FID is evaluated for upper left: Gaussian noise, upper middle: Gaussian blur, upper right: implanted black rectangles, lower left: swirled images, lower middle: salt and pepper noise, and lower right: CelebA dataset contaminated by ImageNet images. The FID captures the disturbance level very well by monotonically increasing. Image is taken from [1].

to state other information. However, this practise is time-consuming and it does not measure diversity [15][16]. Moreover, it is subjective and might include biases of the reviewer. Therefore, human vision is not sufficient to evaluate the quality of the synthetic images.

Usually both a qualitative evaluation involving human visual reviewing and a quantitative assessment based on measure evaluations are performed to asses the performance of a generative network [13].

1.2 Image to Image translation overview

Image to image translation (I2I) aims at learning a mapping that can transfer images from a source domain, representing the content, to a target domain, representing the style, while maintaining the representative content of the input image [9]. For instance: one can take selfie images as the source domain and translate them to desired artistic style images given some cartoon images as reference target domain.

Thus I2I is usually achieved using GANs that are conditioned on an input content image and an input style reference image [2].

Image to image translation methods can be classified in different categories in terms of how diverse they can generate the outputs and how many input domains they can handle[2]. Based on the number of the input domains, we can distinguish: two-domains I2I and multi-domains I2I. The two-domains approaches let the model learn a mapping only between two domains at a time. Instead the multi-domains methods refer to the fact that the model can learn a map among different domains using a single model. The multi-domains scenario strives to simulate the learning process of human beings on the ability of human intelligence to integrate knowledge across different fields. However, multi-domain image translation represents a challenge in the field of I2I [9]. Indeed, most existing models learn individual two-domains mappings. Different techniques have been developed to address such issue. One strategy is the one used in the works [17][18][19] which consists in using both images and label representing the domain information for training a conditional generative model so that the desired domain translation can be then achieved by inputting the right label domain [9]. Then the multi-domain translation is enforced using a domain classification loss by means of an auxiliary domain classifier on top of the discriminator. The domain classifier indeed determines whether the translated image retains the characteristics of the target domain [19].
Another approach refers to the adoption of the perceptual feature loss which makes use of some domain features to represent the domain information [9]. The domain features are extracted by means of a classifier trained to classify the images into the domains. DoSGAN [20] and DCMIT [21] are exploiting such mechanism where they extract the style feature vector of a set of images using the classifier, then they extract the content using an encoder, and input the content code and the style feature vector into a generator to translate the input image to the target domain.

Thus they exploit a domain specific reconstruction loss for computing how close the generated image domain features are to the target domain features.
Furthermore, a single multi-domains model can be also achieved by embedding a generator per each domain as done in [22] [23].
Another challenge concerning the I2I is the lack of large paired dataset. In the recent years, such drawback has been tackled by proposing unsupervised methods that are based on the intuition that unpaired images from two domains should be consistent with their cycle reconstruction [9]. This is implemented through a cycle-consistency loss, which consists in generating an intermediate output by swapping the styles of a pair of images, then swaps the style between the intermediate output again to reconstruct the original image. This enforces the latent vector to preserve the right encoded style information when translating among different domains[24]. This constraint can be also applied in supervised approach in those application where it is desired to swap both style and attribute between images [24].
The pioneer architecture implementing such idea is CycleGAN [25].

Figure 1.3: Example of single-modal multi-domain I2I on faces [2]

GANs are classified also based on how diverse they can generate the samples. We classify them into single-modal and multi-modal. Single-modal model generates a single deterministic output, while multi-modal methods learn a one-to-many mapping between two domains making the model able to generate the same content with different styles. However, the multi-modality mapping is a tough task to achieve in the I2I field that is usually tackled by injecting some noise in the generator and/or using disentangle learning by swapping the latent codes [24][10][9].

Figure 1.4: Example of multi-modal two-domain I2I [2]

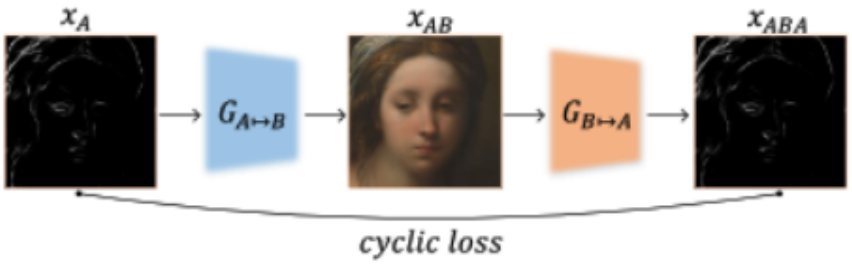


Figure 1.5: Cycle consistency illustration between two domains A and B [2]

1.3 Disentangled representation in the image domain

Disentangled representation learning (DRL) is a tool that lets one introduce expert knowledge into a deep learning model [3]. DRL consists in encoding the input data into separate factors which capture useful information of a variable of interest of the input data. However, it is difficult to identify every desired attributes of an image and generate every possible combination [3]. Moreover, such technique is working only for those factors that are truly independent of each other.

As defined in [3], a factor is defined as 'disentangled' when any modification on this

factor results in a specific change in the generated data.

DRL confers to the deep learning models robustness to represent unseen domains [3]. However, in unsupervised learning DRL can confuse content with style in some cases [9].

A disentangled representation of an image can be achieved either equipping the architecture of the GAN with an autoencoder such as in MUNIT [26], StyleGAN [27], DRIT [28] or adding some regularization terms to the objective function such as in INFOGAN [29]. Another generative model that can be used for achieving disentangled learning is the adoption of $\beta - VAE$ which uses an adjustable hyper-parameter to capture different factors of the input [30].

However, a recent trend in the disentangled learning focuses on the decomposition of the input image into a domain invariant 'content' and a domain specific 'style' representation using the encoders [3]. Indeed, such strategy allows to sufficiently extract the core information of the image, which is very suitable for the image generation field [4]. This approach is defined in the literature as 'Content-Style disentanglement'. Usually the content is encoded in a spatial representation to preserve the spatial correlation, while the style encoding is usually embedded as a vector and it stores the information about the image appearance such as texture, colour and intensity [3]. However, the definition of the content and the style depends on the application [21]. In our work, we refer to the content as defects and to the style as background.

In the next section we present some common techniques to achieve the content-style disentanglement.

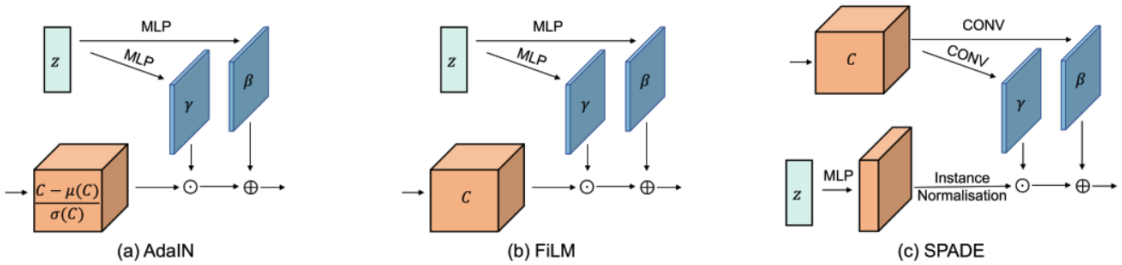
1.3.1 Content-Style disentanglement

At the encoder level the disentanglement is performed at the bottleneck (i.e the last feature vector prior the decoder begins) since it contains the most relevant features

captured by the network. For producing the content code, the content encoder is adopting instance normalization after each convolution layer which helps to remove the style information [3].

The style encoder is instead using average pooling and flattening the feature vector in order to suppress the content spatial information[3].

At the decoder level instead there exists different methods for entangling the content and the style. The most naive technique is to concatenate channel-wise the content and the style codes and input it into the decoder. Most advanced entangling strategies uses Adaptive Instance Normalisation (AdaIN) at different decoder levels to recombine the style and the content representations as done in StyleGAN [27]. Other methods employ weight demodulation such as in StyleGAN2 [31], Spatially-Adaptive denormalisation (SPADE) and feature-wise Linear Modulation (FILM) [3].



Disentanglement building blocks that combine content C with style z: a) AdaIN, b) FiLM, and c) SPADE. $\odot$ and $\oplus$ denote element-wise multiplication and addition, respectively. MLP and CONV denote multilayer perceptron and convolutional layers.

Figure 1.6: Illustration of some entangling techniques [3]

Disentanglement can be encouraged also by using losses such as segmentation loss [3] or using frequency decomposition. Indeed, an image can be decomposed also computing its Fourier transformation to extract the image amplitude and phase. The former reflects the image style, while the latter corresponds to the image content [3].

By recombining the content and the style latent space among different image do-

mains we can achieve multi-modality as well [9].

Moreover, the disentangled learning is proved to improve the performance of the networks when it needs to deal with dataset involving extreme shape variations [32].

2

Related works

The small number of images and the diverse nature of the defects on the industrial surfaces poses a significant challenge to the deep learning-based methods [6]. The advent of advanced machine learning techniques, particularly the Generative Adversarial Networks (GANs), can effectively solve this issue. Indeed, GANs have revolutionized the way defects are generated and manipulated for various applications.

In this chapter, we will explore the state of the art in the field of defect generation and disentangled learning.

The number of works in the literature tackling the industrial surface defect generation problem using the deep learning is very limited [6]. Indeed, as reported by the review work in [6], the deep learning methods applied to synthesise defects emerged only in 2019.

In the literature we can distinguish two types of GAN-based defect generation meth-

ods in terms of the input data provided to the network. One uses noise, while the other is conditioned on input images. The former encourages diversity introducing some randomness but limiting the quality of the generate images which are often low [4]. On the contrary, a GAN conditioned on an input image can retain more texture information from the original image and produce high-quality images [4]. Indeed, besides to allow a controllable generation, a conditional GAN is known to enhance the quality of the output, which can be further improved by means of additional discriminators or auxiliary classifiers to verify that the output image corresponds to the input conditional information [4].

Many conditional-GAN based works use the model CycleGAN to generate images of defects starting from no-defective image. However, such approach may excessively falsify the background [33] [34]. Nevertheless, good results from models built upon CycleGAN can be achieved by editing the network architecture and the loss fuctions [35] [36] [37]. However, CycleGAN operates a translation between two domains, and it cannot be extended to a multi-domain scenario. Indeed for performing a translation among n domains, we need to train $\frac{n(n-1)}{2}$ different models [23].

There are also many works exploiting a conditional GAN conditioned on segmentation masks for capturing the spatial information of the defect i.e. where it is localized on the image in order to reconstruct it at the exact position. This acts as an attention map encouraging the generator to focus on that particular region of the input image.

In [38] they used an input segmentation mask to control both size and location of the defects. The same strategy is exploited in the work of [39] by means of a spatial and categorical control map. In [40] they developed and designed the Dual Deep pix2pixGAN model which is a supervised approach using a two-stage network to generate defects on an unseen fabric texture where the input is artificially marked in white to indicate the position and shape of the desired defect. They first

trained the model on some paired images sharing the same texture and then used the trained model for generating defect on images with a different texture. Another mask-guided approach is MDGAN [33] which uses a binary mask to add the real background information and guide the generator to only focus on the generation of the defects in the specific region.

The spatial information can be captured also by exploiting the concept of disentangled learning by equipping the network with an autoencoder. The work reported in [12] proposes a model built upon StarGANv2, which is based on separating the background and the foreground of the input image to allow the generation of multiple defect types on different backgrounds. They achieved it by first generating separately the defect and the texture image, and later they fused them feature-wise and performed a convolutional operation on it to get a RGB image.

The disentangled representation learning has been employed successfully in various applications to be able to control the desired style and content of the generated image. It has been used in the following domains: face generation [41], medical imaging for artifact and disease removal and transfer [42] [43] [44], fashion [45], general style transfer [31] [46] [47] and content transfer [48].

3

Methodology

3.1 Swapping autoencoder

Our work is built upon the swapping autoencoder (SAE) model proposed by Taesung Park et all [49] which is in turn built on top of the StyleGAN2 architecture. As shown in the Figure 3.1 the original SAE consists of an encoder E, a generator G and two discriminators, one for judging the realness of the whole generated image and one for assessing the texture similarity between the generated image and the reference texture input image. SAE is a two-domain model that is based on taking in input two images both with a common content class and swapping the textures of the two contents for transferring the texture from one image to another. Our approach aims at extending it to a multi-domain scenario where images don't have any common content.

SAE thus aims at performing image manipulation through the disentangled learn-

ing. Specifically, reminding that the encoder E and the generator G form a mapping between image $x \sim X$ and the latent code $z \sim Z$, given two input images X_1 and X_2, SAE first encodes the two images into two components representing the structure and the texture. Thus, X_1 and X_2 are respectively mapped to (z_s^1, z_t^1) and (z_s^2, z_t^2). Then, the latent codes are swapped to get a hybrid image (z_s^1, z_t^2) which is expected to have the structure z_s^1 of X_1 and the texture z_t^2 of X_2. The texture code captures the texture distribution, while the structure code captures the content location-specific information of the input image.

Given an image of size H x H, the shape of the two codes are H/16 x H/16 x 8 for the structure, and 1 x 1 x 2048 for the texture code.

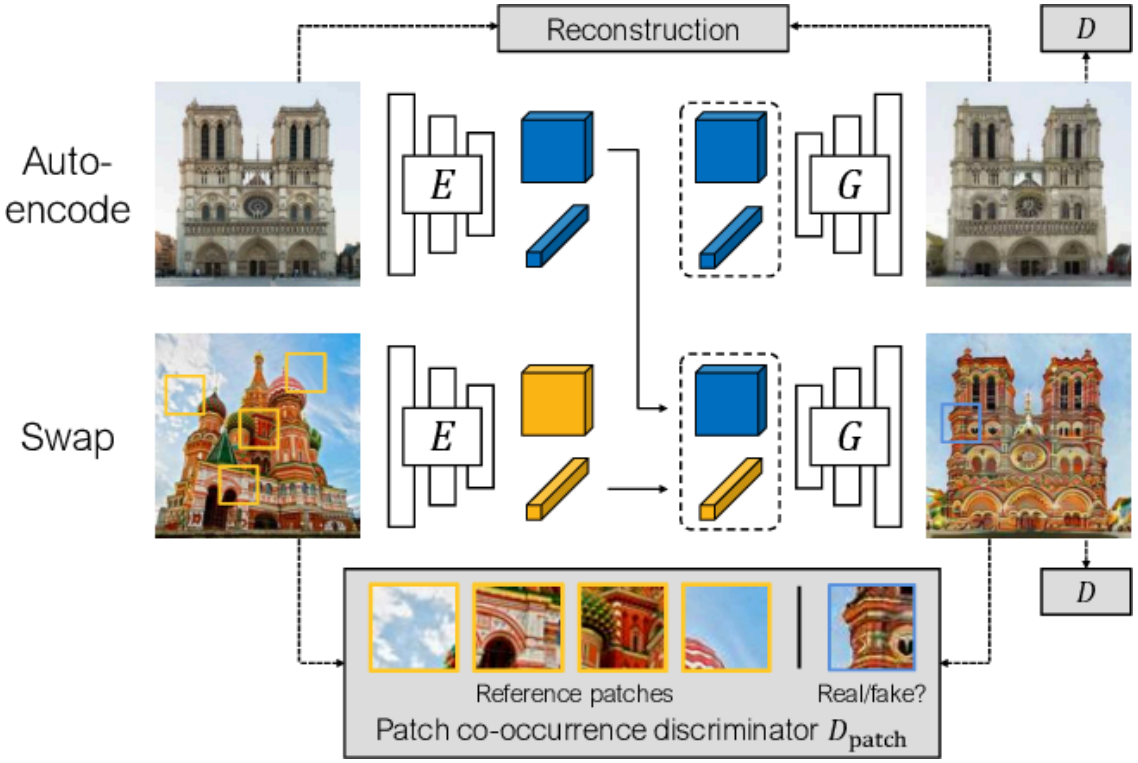


Figure 3.1: Original swapping autoencoder architecture

3.2 Network architecture

Here we present the architecture of our framework. The encoder and the generator are imported from the SAE framework, where the choice of the design of the layers in all the two networks follows StyleGAN2. Regarding the discriminator, we kept the co-occurance patch discriminator for supervising the texture generation and introduced a new structure classifier and a new multi-task discriminator for supervising the defect synthesis.
In the following sections we describe each network component.

3.2.1 Encoder

The encoder is dealing with mapping the input image to both the structure and the texture code. It consists of 4 downsampling ResNet blocks followed by two convolutional layers to produce the structure code. The texture code is instead produced by branching off the network after the 4 downsampling ResNet blocks and adding two convolutional layers followed by an average pooling and a dense layer for depriving it of the spatial information.
The architecture of the encoder is reported in the Figure 3.2.

3.2.2 Generator

The generator is based on StyleGAN2 [49][31]. It consists of 4 residual blocks and 4 upsampling residual blocks. The structure code is inputted directly into the network, while the texture code is injected using the weight modulation layer borrowed from StyleGAN2 [31]. Then, the output image is generated by applying a convolutional layer at the end of the residual blocks.
The architecture of the generator is reported in the Figure 3.2.

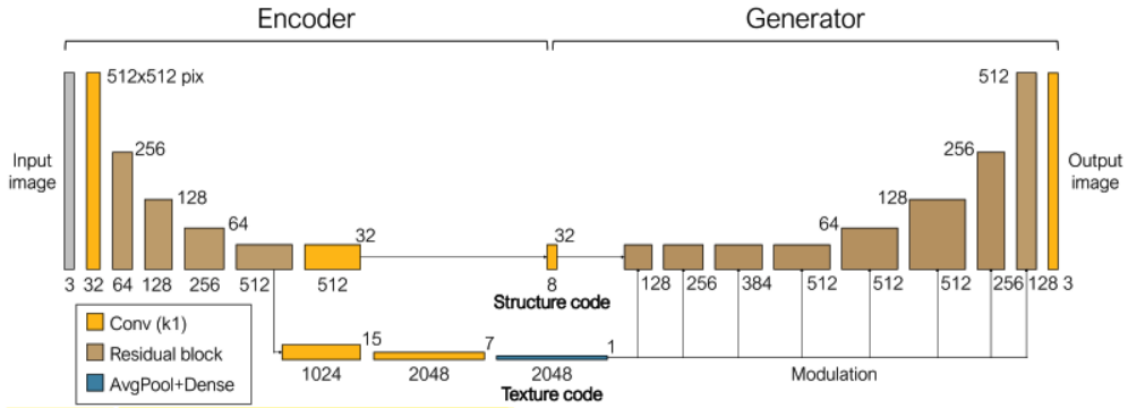


Figure 3.2: Encoder and generator

3.2.3 Multi-task Discriminator

We extended the original discriminator by branching off the network for each defect types.

The original discriminator network is borrowed from StyleGAN2 [31] except for the minibatch discrimination which has not been employed in the current work. The minibatch discrimination is a discriminative technique to avoid the mode collapse problem. It consists in discriminating between the whole minibatches of samples rather than between the individual samples [50] .

The StyleGAN2's discriminator first transforms the input image with the resolution $2^{LOG_RESOLUTION}$x$2^{LOG_RESOLUTION}$ to a feature map of the same resolution and then runs it through a series of residual blocks, where each block consists of two 3×3 convolutions with a residual connection. At each block the resolution is down-sampled by 2 while the number of features extracted are doubling.

The architecture of the StyleGAN2's discriminator is reported in the Figure 3.3.

We branched off this network by adding an head for each defect type. This allows us to ensure the realness of each generated defect shape Figure 3.5.

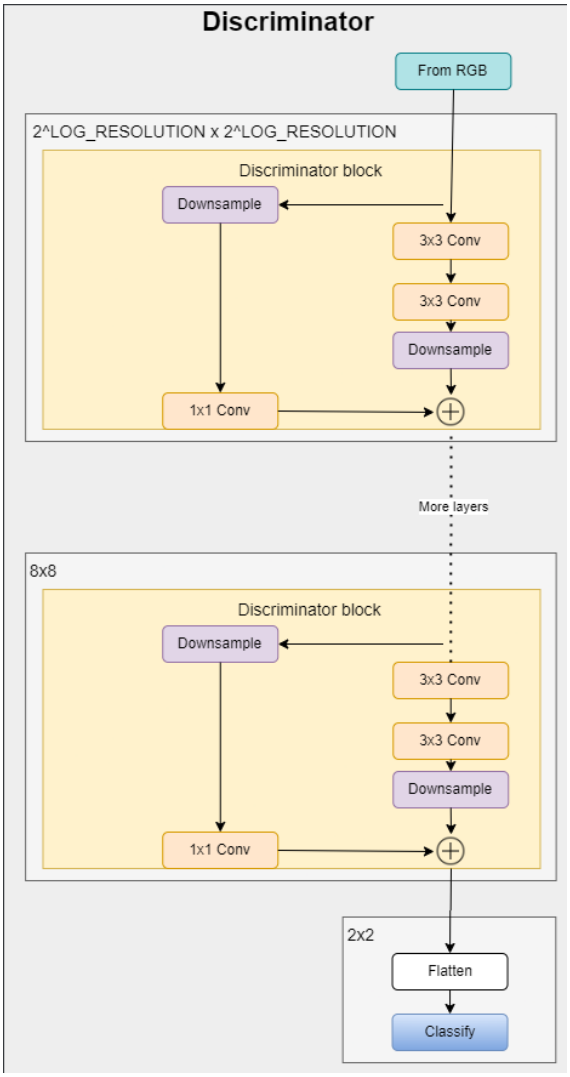


Figure 3.3: Discriminator architecture

3.2.4 Patch discriminator

The patch discriminator is a network that is trained to ensure that the texture in X_2 represents the same texture in X_{hybrid}. This is achieved by first getting a patch from the hybrid image, and a group of patches from the reference texture image. Then, for each patch, the patch discriminator extracts the features by using 5 downsampling residual blocks, 1 residual block and 1 convolutional layer.

Therefore, the features encodings of the reference patches are averaged together and concatenated channel-wise with the representation of the real/fake patch. Thus, it is inputted to a classifier made of 3 dense layers for outputting the final prediction. The score is indicating how similar the patches are. Indeed, any patch from the hybrid image should not be distinguished from a group of patches of the reference image.

The architecture of the patch discriminator is reported in the Figure 3.4.

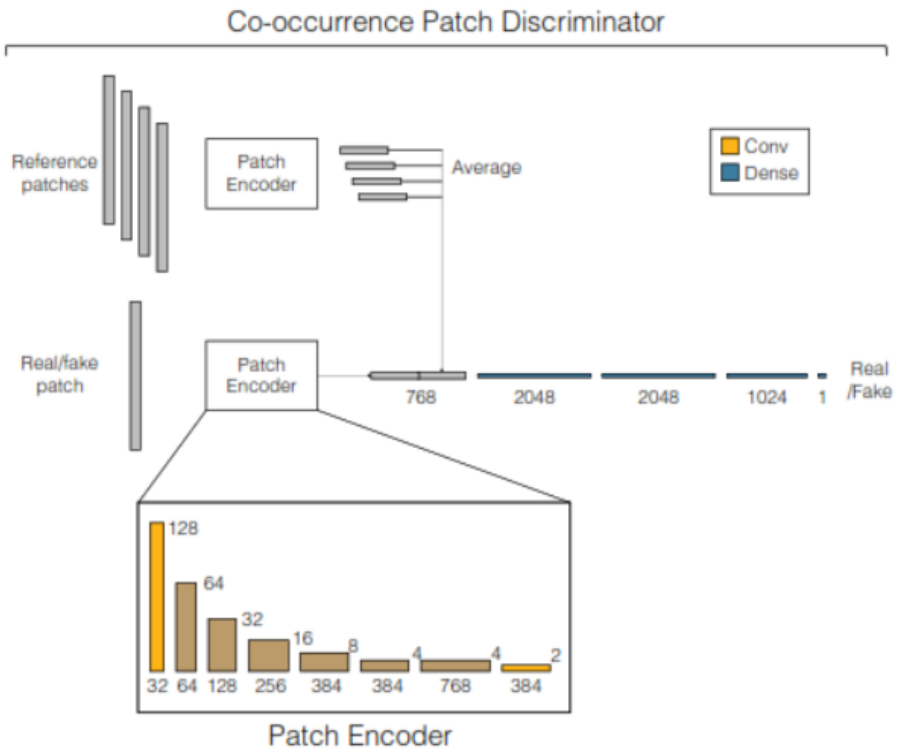


Figure 3.4: Co-occurence patch discriminator

3.2.5 Classifier

The classifier is a simple network made of an encoder whose architecture is the same as the one implemented at the generator side. However, here we extract only the structure code which is further inputted to a fully connected network for being

classified into the defect class (Figure 3.5).

The classifier aims at ensuring that the generated image embeds the structure code that was extracted by the encoder at the generator side. This is achieved by checking that the structure code represents the same defect class appearing on the defect reference image. Thus, it improves the disentangled representation of the reference defect image.

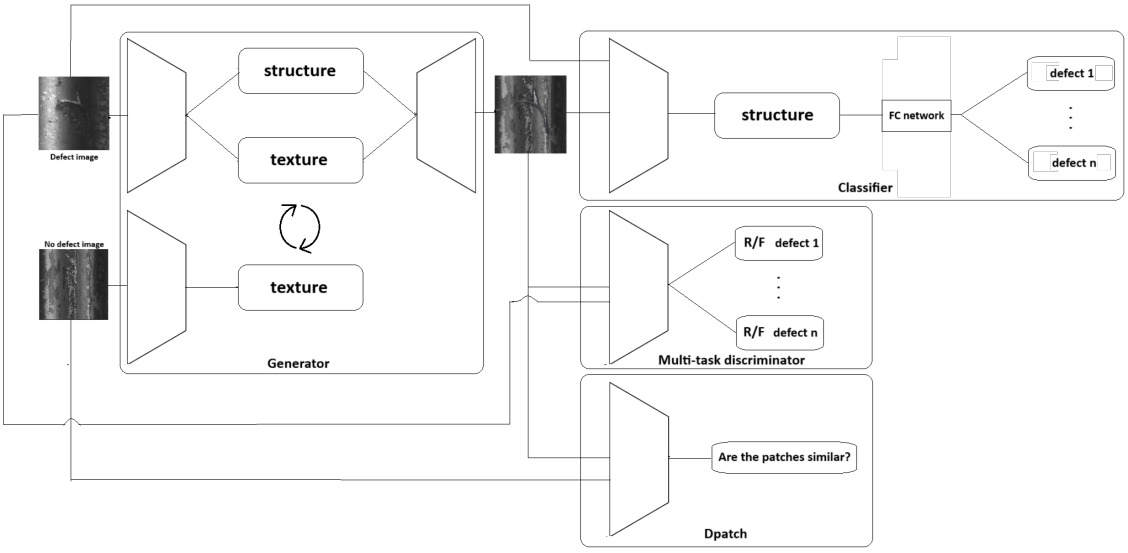


Figure 3.5: Overall architecture

3.3 Training objectives

In this section we present the losses functions adopted for training the network.

3.3.1 Anchor domain hypothesis

Inspired by the work presented in [12], we adopted the 'anchor domain' assumption in our work. The anchor domain hypothesis assumes that given an image where no defects appear, the structure code should be zero. By this hypothesis, mixing

any texture code with a zero structure code should return an image with only the texture. This hypothesis is supposed to encourage a better disentangled representation forcing the texture not to contain any defect-related information [51]. This strategy have been introduced successfully for the first time in the work reported in [52].

3.3.2 Losses

Given a defective image X_1 such that $(z_s^1, z_t^1) = E(X_1)$ and a texture reference image X_2 such that $(0, z_t^2) = E(X_2)$, we used the following losses for training our framework.

- NON-SATURATING ADVERSARIAL LOSS which enforces the generator and the encoder to produce realistic images that are indistinguishable from the real images.

$$L_{adv}^{G}(E,G) = E_{x_1,x_2\sim X}[-log(D(G(z_s,z_t)))]$$

$$L_{adv}^{D}(E,G,D) = E_{x_1,x_2\sim X}[log(D(x))] + E_{x_1,x_2\sim X}[log(1-D(G(z_s,z_t))]$$

  where $D(G(z_s,z_t))$ considers both the reconstruction of the original image $D(G(z_s^1,z_t^1)$ and the hybrid image $D(G(z_s^1,z_t^2)$

- TEXTURE ADVERSARIAL LOSS which guarantees that the texture code encodes the same texture of the input texture reference image. This is implemented using a patch co-occurrence discriminator which aims at discriminating between a group of patches of the texture reference image (indicated as $crops(x_2)$) and a patch of the hybrid image (indicated as $crop(G(z_s,z_t))$).

$$L_{CoccurGAN}^{G}(E,G) = E_{x_1,x_2\sim X}[-log(D_{patch}(crop(G(z_s,z_t)),crops(x_2))]$$

$$L^{D}_{CoccurGAN}(E,G,D) = E_{x_2 \sim X}[log(D_{patch}(crop(x_2), crops(x_2)))] + E_{x_1,x_2 \sim X}[log(1 - D_{patch}(crop(G(z_s,z_t)), crops(x_2)))]$$

where for $D(G(z_s,z_t))$ we mean both the reconstruction of the original texture reference image $D(G(0,z_t^2)$ and the hybrid image $D(G(z_s^1,z_t^2)$

- FEATURE RECONSTRUCTION LOSS. We designed it to be sure that the generator generates both the specific defect code and the specific texture code, and enforcing thus the encoder to decompose the input image correctly. Given the hybrid image $X_3 = G(z_s^1,z_t^2)$ such that $(z_s^3,z_t^3) = E(X_3)$ and the reconstructed texture reference image $X_4 = G(0,z_t^2)$ such that $(0,z_t^4) = E(X_4)$, the feature reconstruction loss is defined as follows:

$$L_{feature}(E,G) = MSE(z_s^3,z_s^1) + MSE(z_t^3,z_t^2) + MSE(z_t^4,z_t^2)$$

- CLASSIFICATION LOSS which ensures that the domain specific content is properly encoded and carries enough information from the target defect domain.
  Given a generated image X_{fake} with defect type y, we define the classification loss as follows:

$$L^{G}_{cls}(E,G) = \text{CrossEntropy}(P(y|X_{fake}),y)$$

  where with X_{fake} we mean both $G(z_s^1,z_t^2)$, i.e. the hybrid image, and $G(z_s^1,z_t^1)$ i.e. the reconstruction of the original defective image.

- STRUCTURE CONSISTENCY LOSS. We employed also a structure consistency loss inspired by the work of [53]. This encourages the generator to generate the right defect and the encoder to capture the code of the defect

better. The structure consistency loss is defined as:

$$L_{Structure}(E,G) = E_{x\sim X}[||(x_1 - G(0,z_t^1)) - (G(z_s^1,z_t^2) - x_2)||_1] \\ + E_{x\sim X}[||((G(z_s^1,z_t^1) - G(0,z_t^1)) - (G(z_s^1,z_t^2) - x_2)||_1]$$

where $G(0,z_t^1)$ represents the image X_1 without defects.

The structure consistency loss is based on the observation that the difference between X_1 and $G(0,z_t^1)$ and the difference between $G(z_s^1,z_t^2)$ and X_2 should be close since they are affected by the same defect.

- IMAGE RECONSTRUCTION LOSS, which lets the generator learn to reconstruct the input image and allows the implementation of an unsupervised approach.

$$L_{rec}(E,G) = E_{x\sim X}[||x_1 - G(z_s^1,z_t^1))||_1] + E_{x\sim X}[||x_2 - G(0,z_t^2))||_1]$$

Thus the final objective used for the encoder and the generator is:

$$L_G = L_{rec} + 0.5L_{adv}^G + L_{CoocurGAN}^G + L_{Structure} + L_{feature} + L_{cls}^G$$

Regarding the objective function for both the discriminators, it is defined as follows.

$$L_D = L_{adv}^D + L_{CoccurGAN}^D$$

As regards the classifier instead, given an image X with defect type y, we train the classifier using the following loss:

$$L_C = \text{CrossEntropy}(P(y|X), y)$$

where as images X we consider both the original defective reference image, its reconstruction and also the hybrid image.

4

Experiments

In the following chapter we present the set up for our experiments and discuss the results.

4.1 Training

We followed the training scheme as described in SAE with minor modifications. To fit the model on a single NVIDIA Tesla P100 and speed up the training we set the batch size to one and trained the model for 150.000 iterations. In particular we used Adam as optimizer with 0.001 learning rate, $\beta_1 = 0$, $\beta_2 = 0.99$.
Moreover, we normalized the training images in the range [-1,1].
Regarding the patches for the patch discriminator, their sizes varies from 32x32 pixels to 64x64 pixels. Both the discriminators and the classifier are regularized with R1 regularization every 16 interactions for stabilizing the training of the overall model.

The training time is about 30 hours.

4.2 Dataset

We used a real industrial dataset. The dataset contains six different kinds of defects: scratch, shell, mark, scoring, tear, and notching. All images are grayscale.
For building the training set, we took all the defective images and cropped them into patches of ratio 1:1 containing the defect and resized it to 256x256 resolution. The crop operation is illustrated in Figure 4.1. Moreover, since the original dataset appeared imbalanced, we performed some data augmentation operations that is vertical and horizontal flip, on the minority classes such as shell, scoring and notching. Furthermore we downsampled the original no-defective set and carrying out a 256x256-sized centered cropping operation.
In the tables below we report the statistics of the original and training dataset. In Figure 4.2 we report an example of the diversity of our training set per each type of defects, while in Figure 4.3 we illustrate some images of the normal samples used as training set. From these no-defective images we can see that the textures are really diverse and some patterns might be confused for a defect. This represents a challenge for the disentanglement mechanism, since the network might represent this information into the structure code.

scratch	shell	mark	scoring	tear	notching	normal
105	28	163	25	167	40	7262

Table 4.1: Original industrial set

scratch	shell	mark	scoring	tear	notching	normal
105	84	163	75	167	120	654

Table 4.2: Training set

The choice of the resolution 256x256 pixels has been done after we experienced that choosing a smaller resolution such as 128x128 pixels led to worse results. This might have occurred due to the fact that 128x128 patches requires to adopt larger patches for the patch discriminator than using 256x256 images, and hence larger parts of the defects are more likely to occur inside a patch. Thus, this might confuse the patch discriminator.

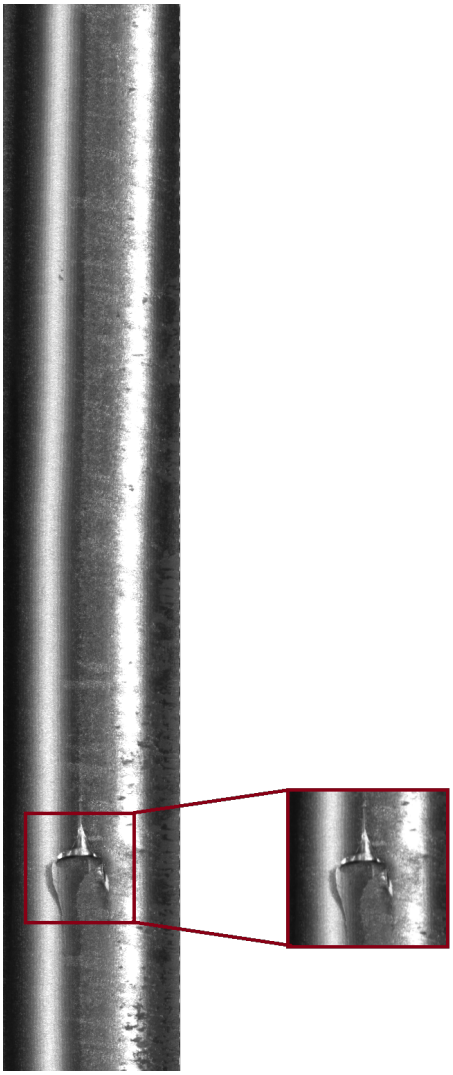


Figure 4.1: Example of the cropping process of an original image representing a tear defect class

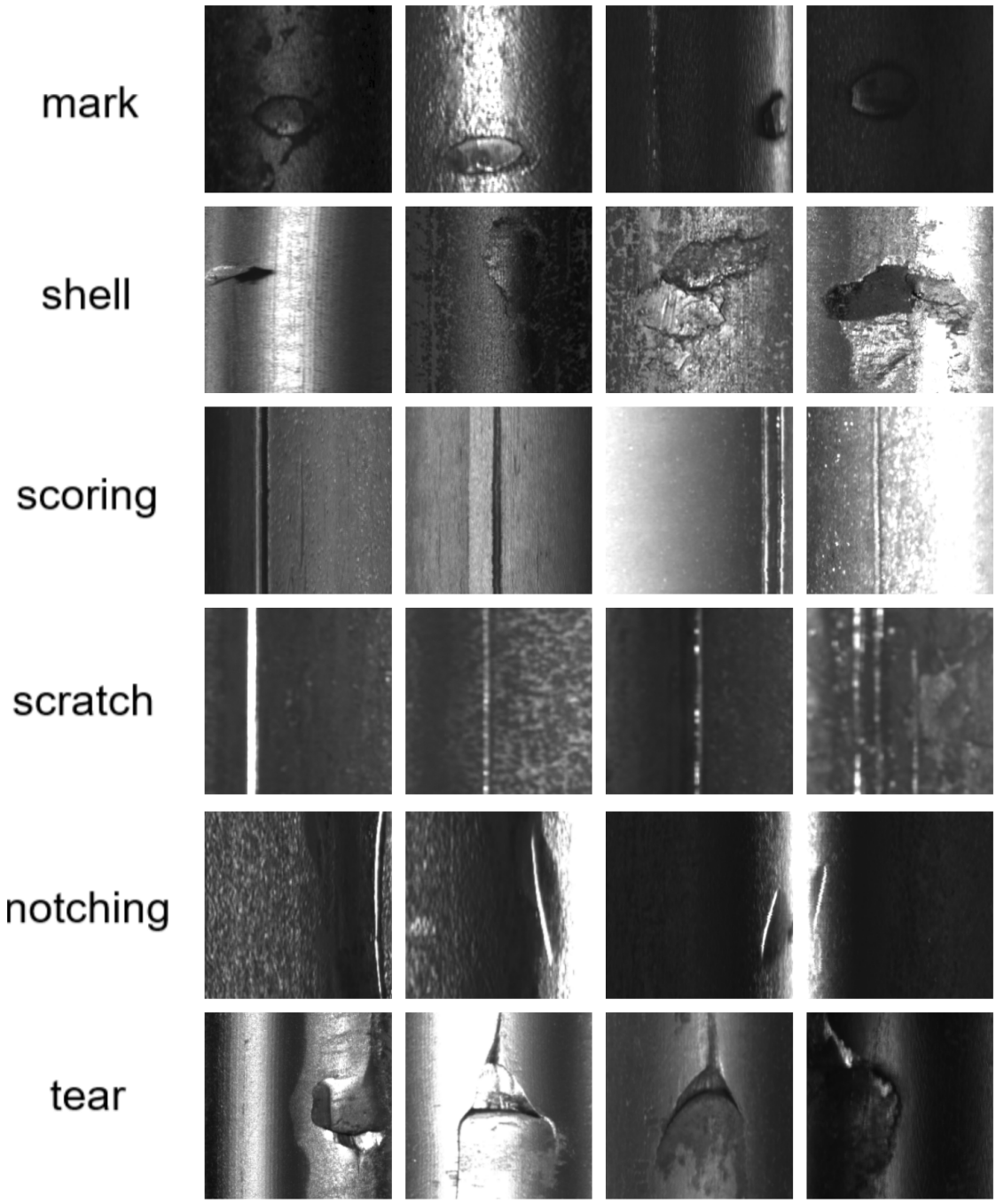


Figure 4.2: Training set samples about defective images. Each row contains defective images of the same class. From the first row to the last: mark, shell, scoring, scratch, notching, tear

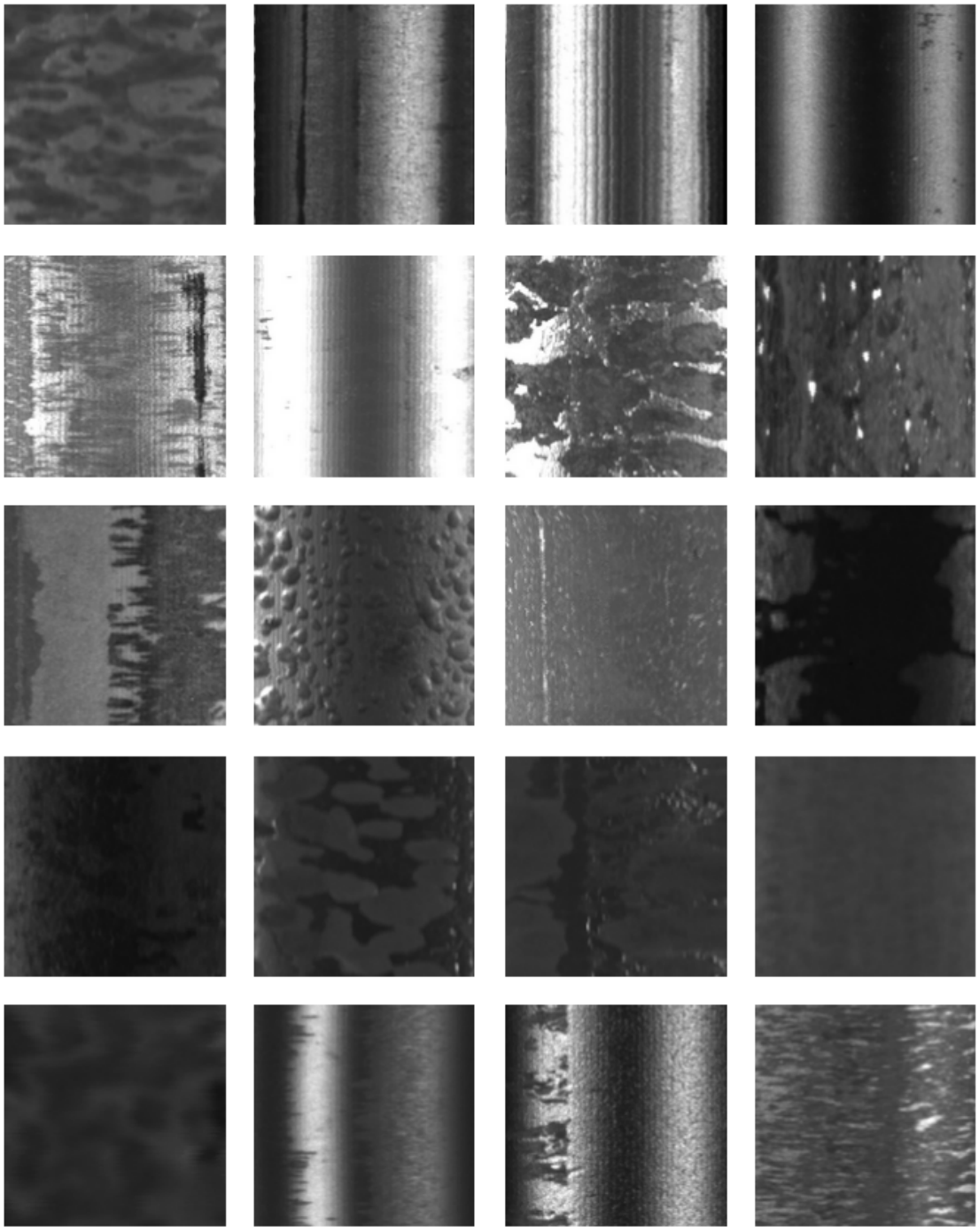


Figure 4.3: Training set samples about the normal samples

4.3 Results

In this section we present and discuss the results we obtained through the experiments.

For testing we followed the strategy used in [12] which consists in sampling defective reference images from the training set for each defect domain, while the normal samples are unseen images.

4.3.1 Quantitative evaluation

We employed the frechet inception distance (FID) to evaluate both the visual quality and the diversity of the generated images. Since the low volume of the training images, the FID score has been computed considering the activation of the first max pooling layer of the Inception-v3 network. This differs from the original implementation which is using the last average pooling features and requires at least 2048 images [1]. Thus, despite the lower layer features still have spatial extent, the resulting scores might no longer be correlated with visual quality. We report it for completeness of our work.

We thus report also the kernel inception distance (KID) score which is a more stable metric for small sets of images, so it is more suitable for our study.

The lower the FID and the KID scores, the closer are the features between the real and the defect datasets.

For computing both the scores, we first generated a synthetic dataset per each class of defect where the number of the generated images is close to the one in the original training dataset per each defect class. Then we calculated the scores between each defect class synthetic dataset and the corresponding defect class training dataset.

We report the scores in the table below. For the aforementioned consideration about the FID score, we should base our observations on the KID scores which are

low indicating so that the generated images resemble the realness of the original dataset.

	scratch	shell	mark	scoring	tear	notching
FID	2.45	4.93	0.72	7.12	0.74	1.15
KID	0.10	0.07	0.13	0.11	0.06	0.19

Table 4.3: Quantitative evaluation

4.3.2 Qualitative evaluation

For performing the qualitative evaluation, we picked up seven defective images of the same defect class but with different style and translated it into eight diverse texture domains.

From inspecting the generated images per each defect class which are reported in Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9, Figure 4.10 and Figure 4.11, we can see that not all the textures are generated correctly. In particular, in most of the cases the texture code seems embedding only the style of the reference texture image in terms of colours and shading. This results hence in a style translation instead of a texture/background translation. However, the network succeeds well in generating some complex textures such as the ones showed in the second and third row in Figure 4.6.

Regarding the defects generation instead, we can observe that some defect classes are generated with high fidelity compared to its corresponding defective reference image. However, we can notice that in some cases part of the texture of the defective reference image is painting onto the generated image as well. This let us state that some part of the texture is getting encoded also into the structure code.

Moreover, we could notice that there exists a specific texture reference image that is reconstructed wrongly but with high resolution. In particular we observed that given the input texture reference image reported in the orange box in Figure 4.4,

the network generates a texture (reported in the yellow box in Figure 4.4) whose style and pattern is similar to the corresponding input reference image but it is actually equal to another texture image existing in the training dataset (illustrated in the blue box in Figure 4.4). This made us suspecting that there was a case of mode collapse occurring when the texture input image resembled that particular style input image. We thus investigated it by regenerating those texture images whose style and texture were similar to that image (Figure 4.5). From this study, we could conclude that this case of mode collapse is restricted to only the texture input image reported in the orange box in Figure 4.4 .

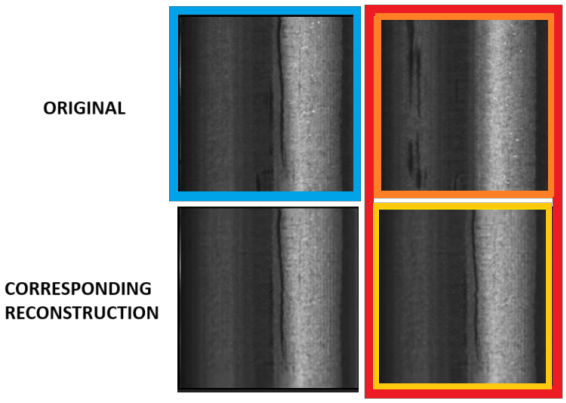


Figure 4.4: Highlighted in red is the suspected case of mode collapse. Given the texture reference image in the the orange box, the model reconstructs it as shown in the yellow box. However its reconstruction is identical to another texture image in the training dataset (shown in the blue box).

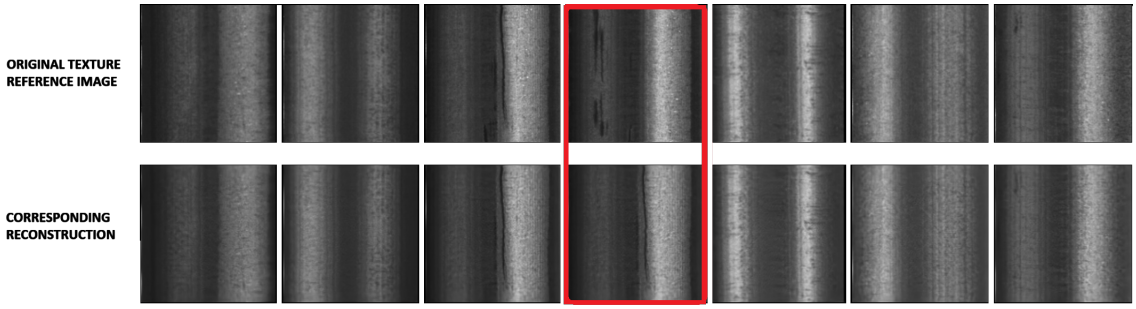


Figure 4.5: Mode collapse study. The first row reports the original image, while the second row represents the corresponding generated image. We can see that although all the images share the same style and texture, only the image at the top in the red box is reconstructed wrongly.

In the following images (Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9, Figure 4.10, Figure 4.11) we report the results we obtained from translating seven defective images of the same defect class but with different style, into eight diverse texture domains. Discussion about these results have been done previously at the beginning of the current 'Qualitative evaluation' section.

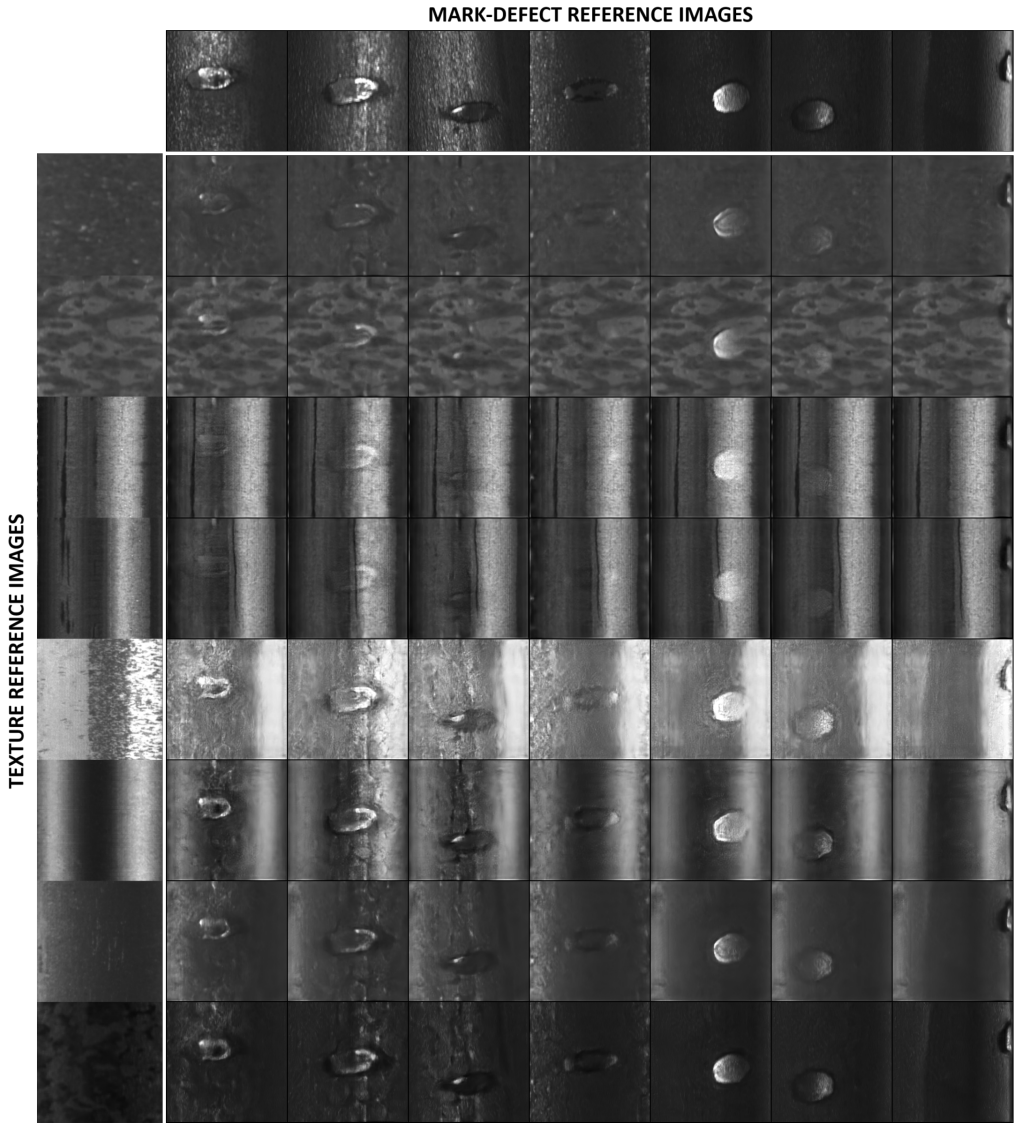


Figure 4.6: Generated samples with mark defects. First row represents the defect reference images, while the first column represents the texture reference images.

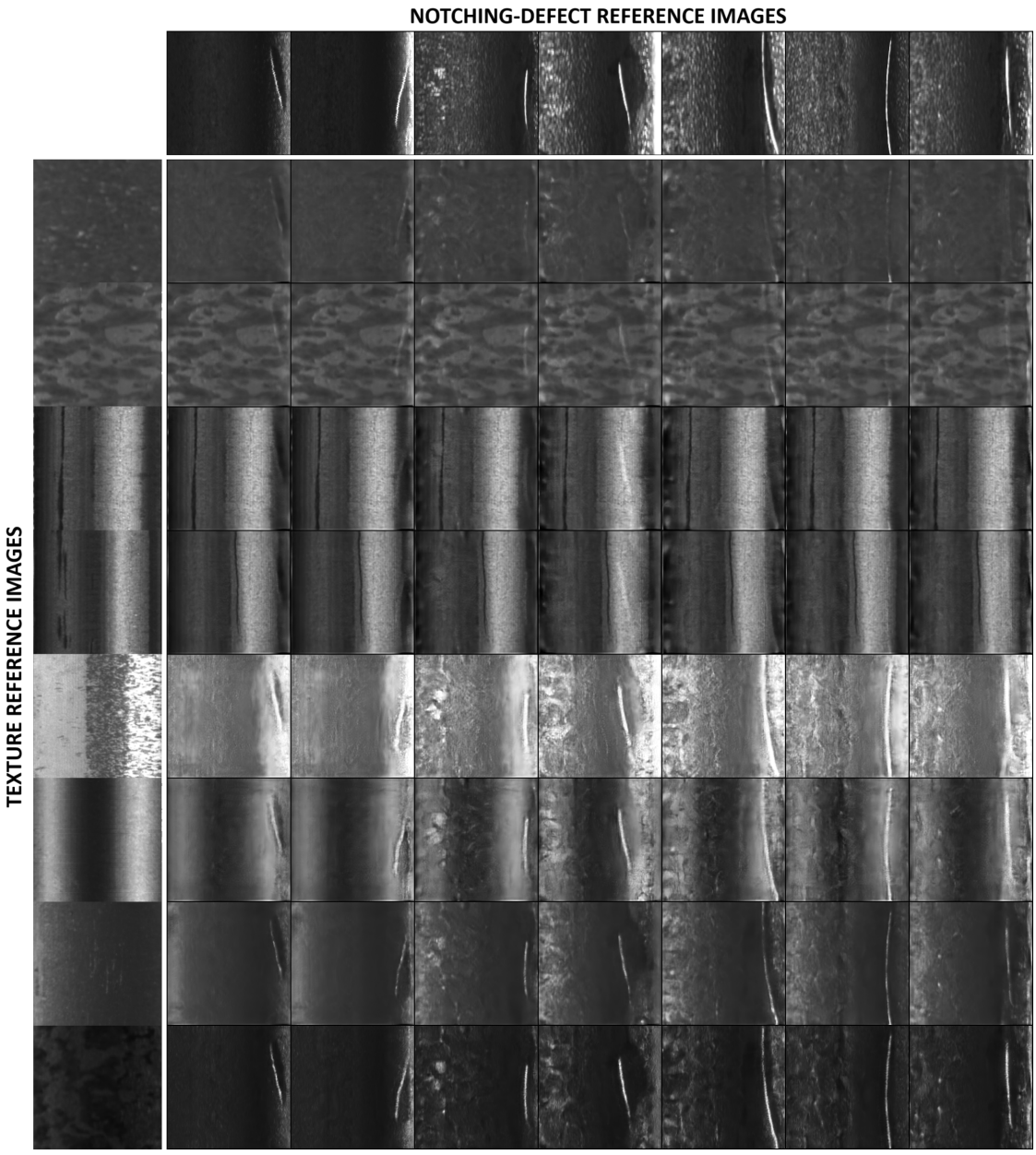


Figure 4.7: Generated samples with notching defects. First row represents the defect reference images, while the first column represents the texture reference images.

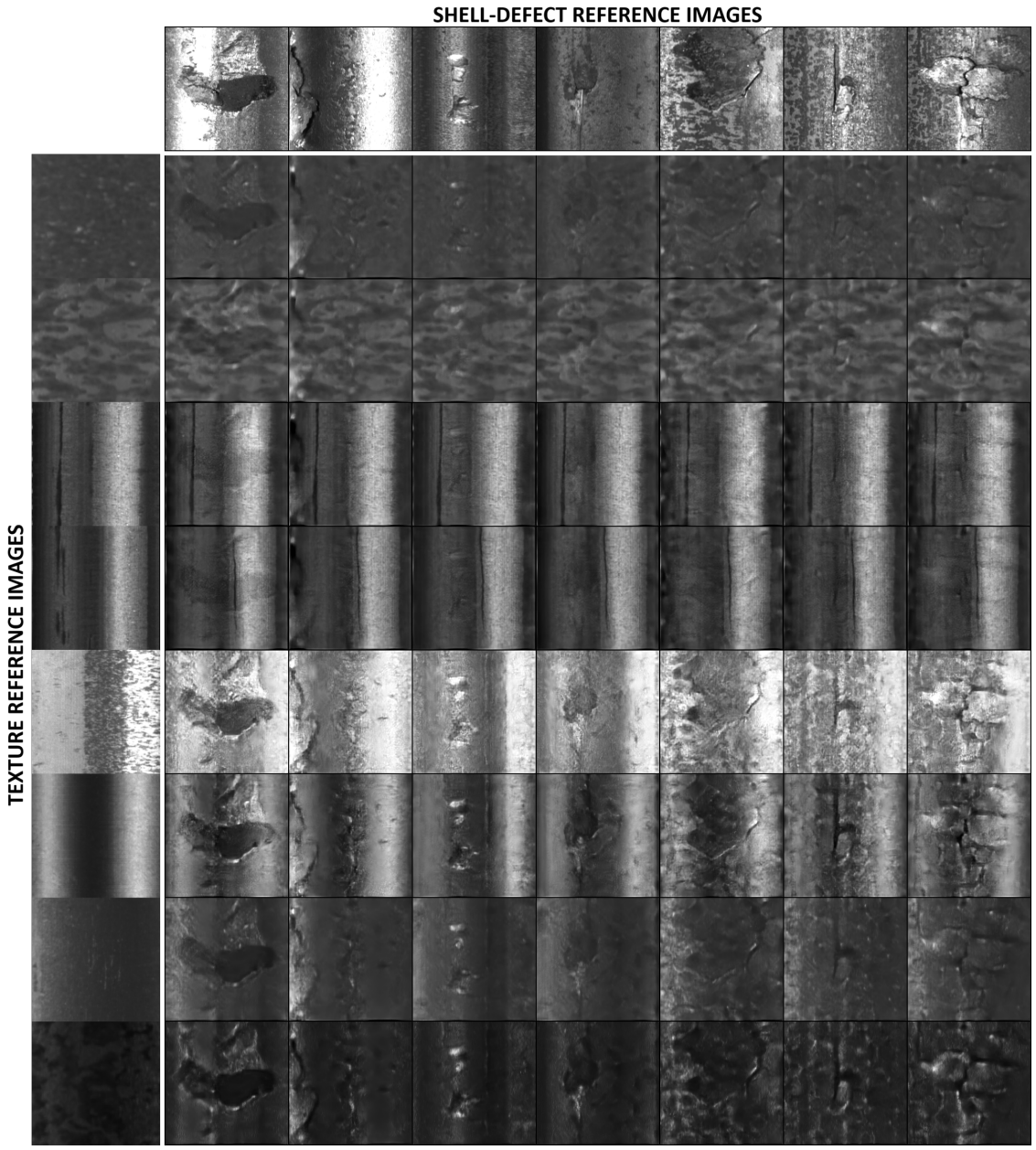


Figure 4.8: Generated samples with shell defects. First row represents the defect reference images, while the first column represents the texture reference images.

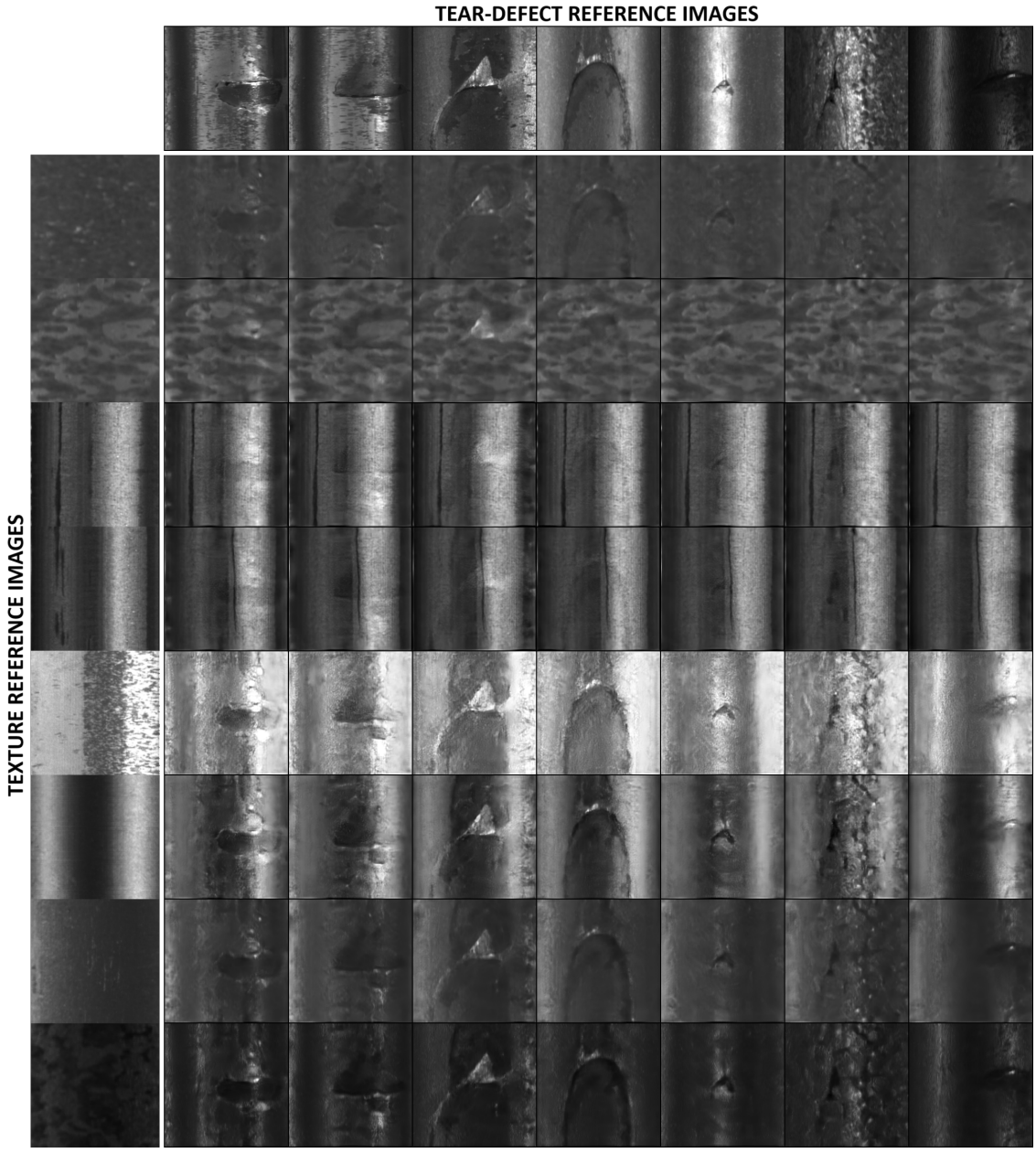


Figure 4.9: Generated samples with tear defects. First row represents the defect reference images, while the first column represents the texture reference images.

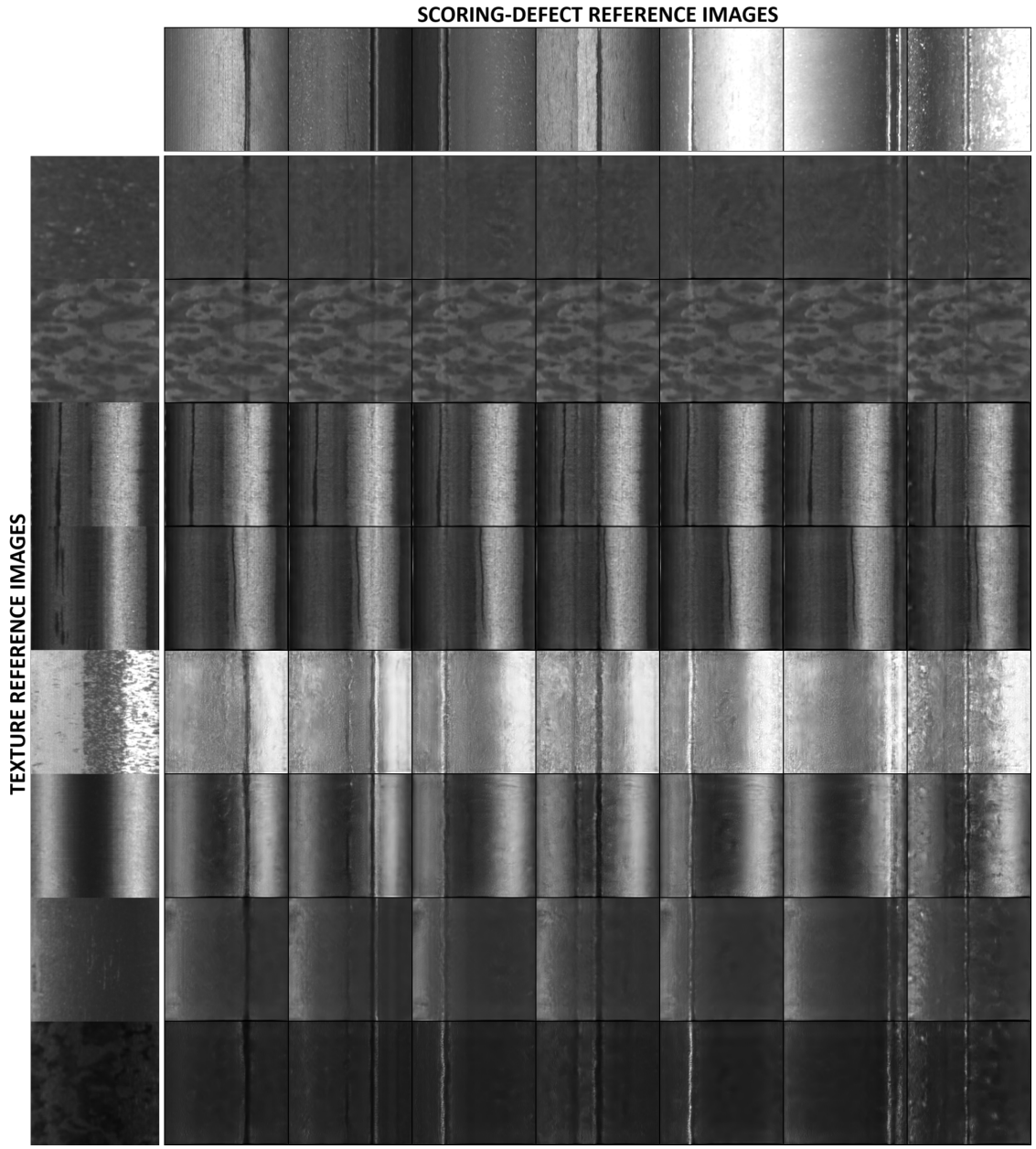


Figure 4.10: Generated samples with scoring defects. First row represents the defect reference images, while the first column represents the texture reference images.

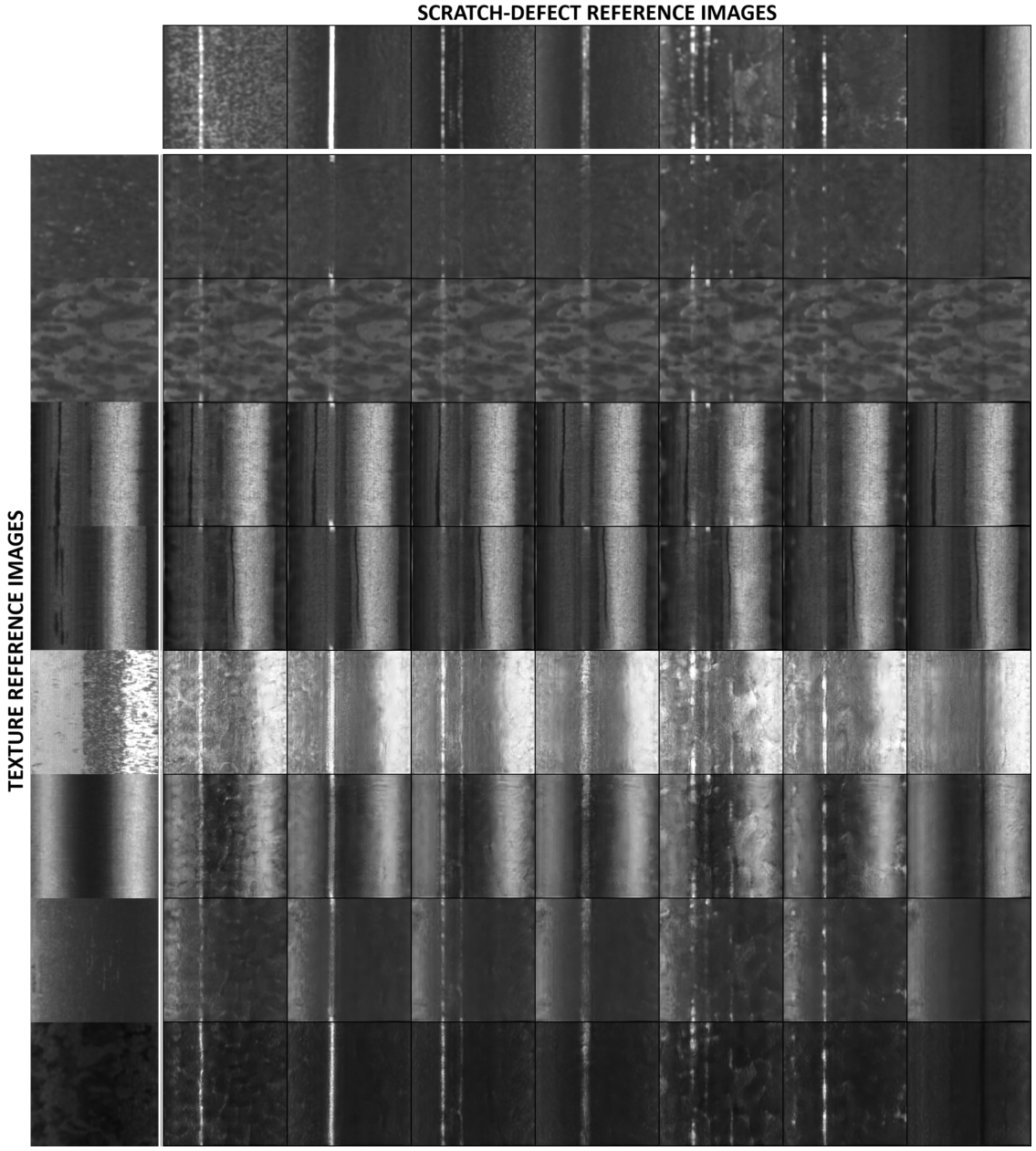


Figure 4.11: Generated samples with scratch defects. First row represents the defect reference images, while the first column represents the texture reference images.

4.3.3 Anchor domain hypothesis verification

We conducted also some experiments for verifying how well the anchor domain hypothesis holds. We thus generated some images using only the texture code of the defective images and setting the structure code to zero. We did the same for the normal images. We report the results in the images reported below (Figure 4.12, Figure 4.13, Figure 4.14, Figure 4.15, Figure 4.16 and Figure 4.17). We can see that as observed in the qualitative study, the network is generating mostly only the style of the input image. Moreover, looking at the texture images generated from the defective images, we can see that part of the defect got embedded in the texture code in some cases. This is particularly evident for the scoring defect class in Figure 4.16.

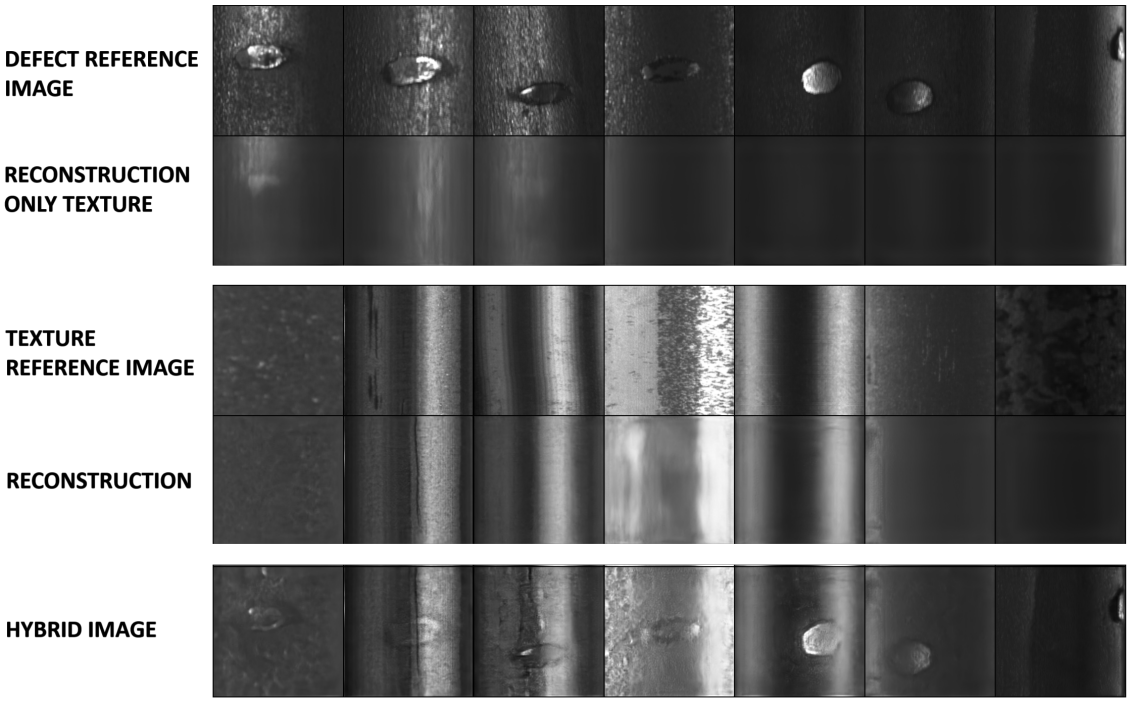


Figure 4.12: Texture reconstruction for the mark defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image.

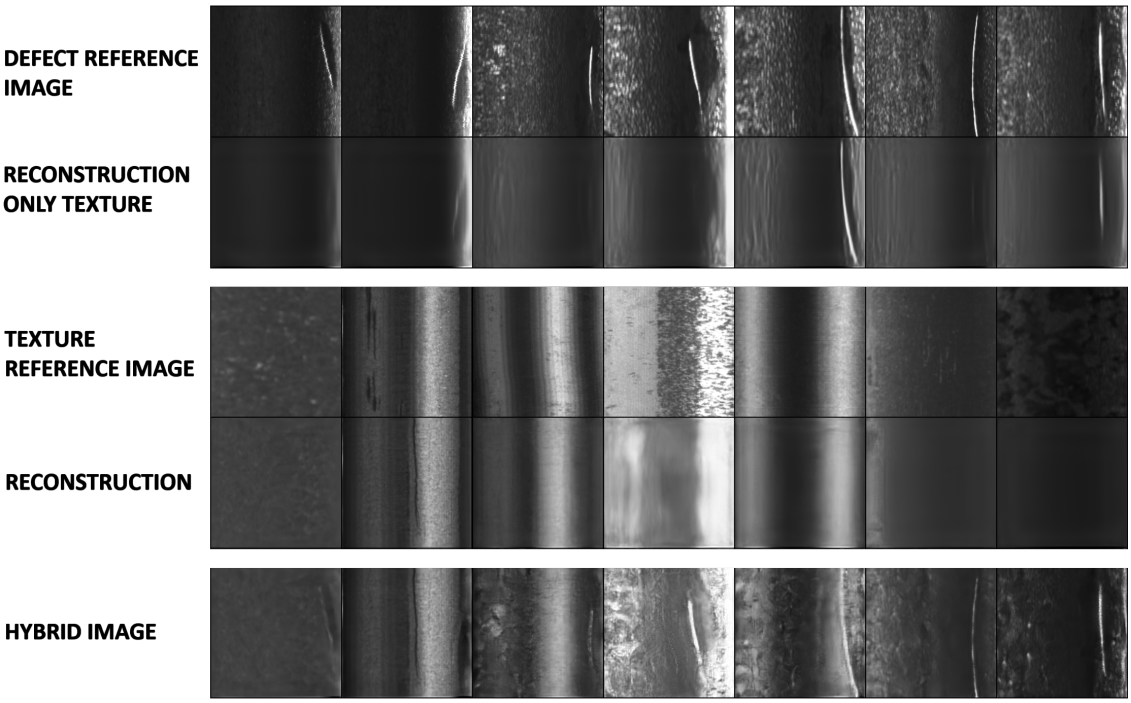


Figure 4.13: Texture reconstruction for the notching defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image.

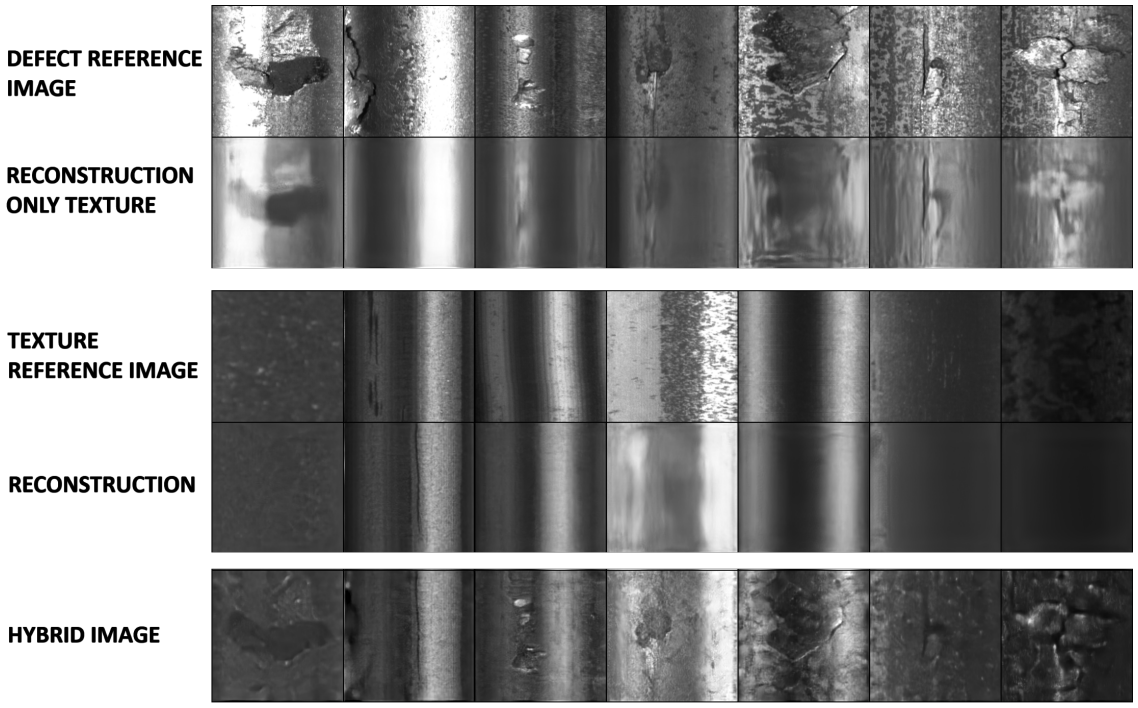


Figure 4.14: Texture reconstruction for the shell defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image.

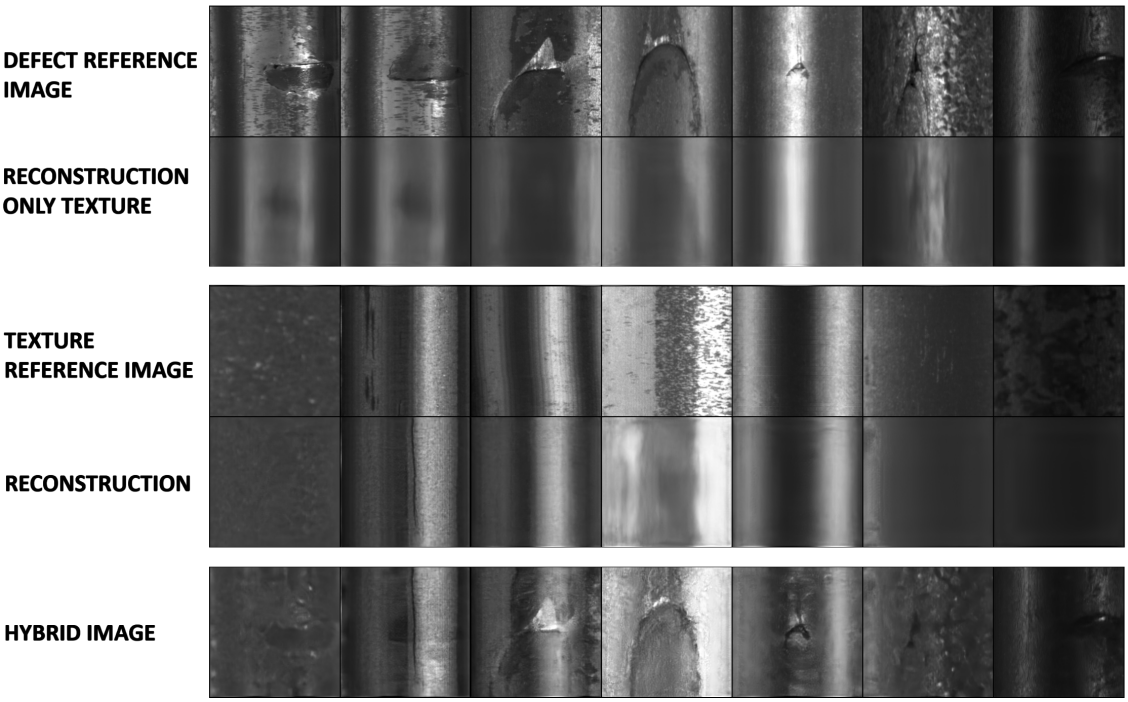


Figure 4.15: Texture reconstruction for the tear defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image.

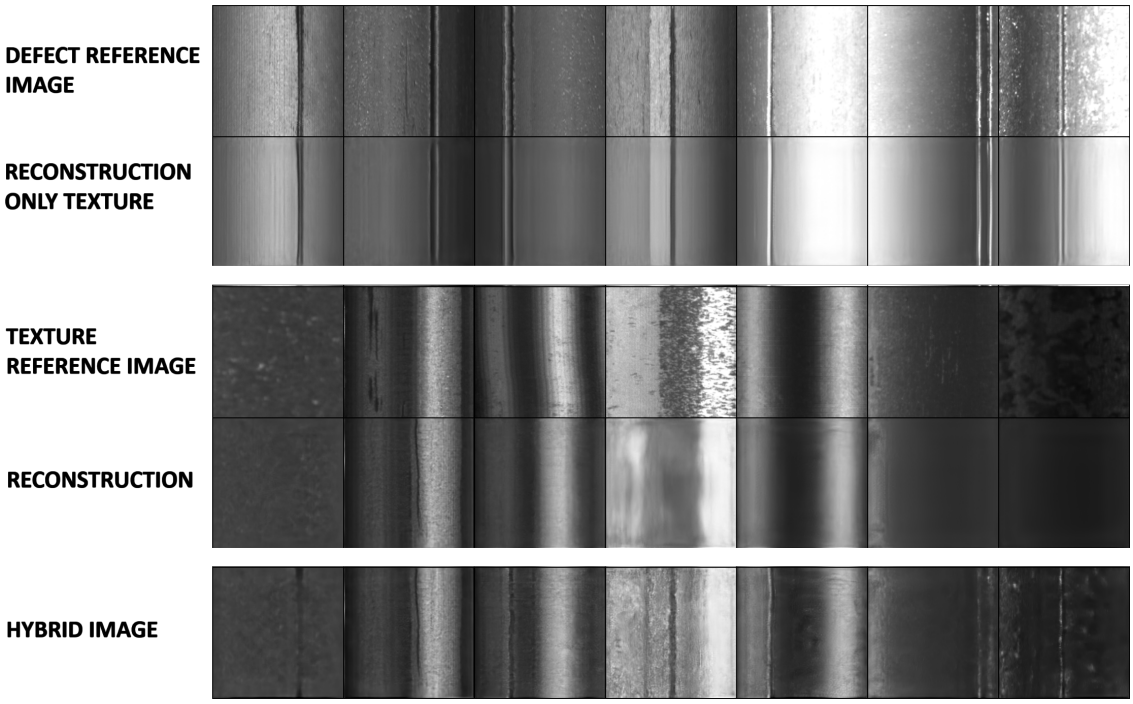


Figure 4.16: Texture reconstruction for the scoring defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image.

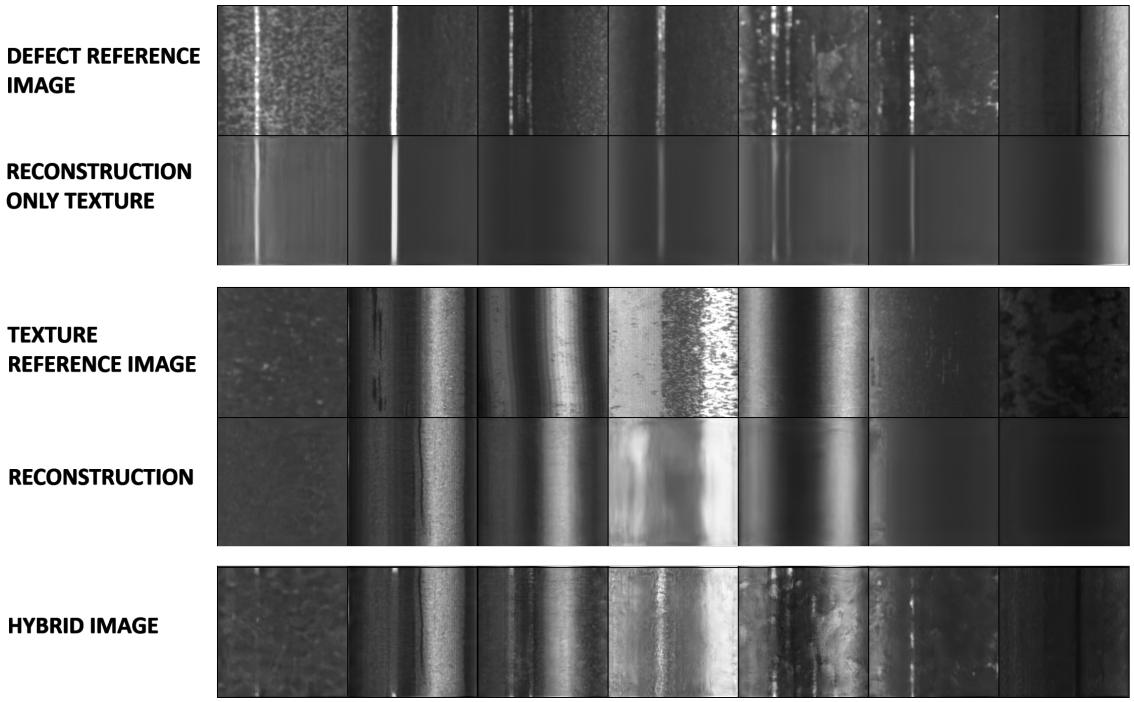


Figure 4.17: Texture reconstruction for the scratch defects. First and third row represents the defect and normal reference images respectively. The second and the fourth row reports the texture reconstruction for the defect and no-defect reference image respectively. The last row instead illustrates the images obtained by swapping the texture code of the defective image with the normal reference image.

4.3.4 Data augmentation

We also evaluated our method as a data augmentation tool for the defect classification task. We considered ResNet50 as backbone for our classifiers.

Dataset

We first trained a classifier on a dataset that had been built by splitting the industrial set (4.1) into train, validation and test set. In particular before splitting we downsampled the normal images as done for the training set of the GAN. Then, since the training set appeared imbalanced we followed the same strategy adopted for setting up the training set for the GAN. Thus we applied the augmented operations such as horizontal and vertical flip on the minority classes that is scoring, notching and shell. We report in the tables 4.4, 4.5, 4.6 the statistics for the three datasets, where the train set refers to the set built after performing the augmented operations.

We used the validation set for supervising the training and spot signs of overfitting or underfitting.

scratch	shell	mark	scoring	tear	notching	normal
59	42	120	33	124	69	604

Table 4.4: Training set for the classifier

scratch	shell	mark	scoring	tear	notching	normal
10	5	10	5	10	7	12

Table 4.5: Validation set for the classifier

scratch	shell	mark	scoring	tear	notching	normal
35	9	31	9	33	10	38

Table 4.6: Test set for the classifier

The second classifier has been trained on the previous training dataset augmented with some generated data. Such dataset was constituted as reported in the table 4.7.

scratch	shell	mark	scoring	tear	notching	normal
403	374	438	541	370	428	604

Table 4.7: Augmented training set for the classifier

We report in Figure 4.18 the pile chart showing the comparison of the statistics of the two datasets.

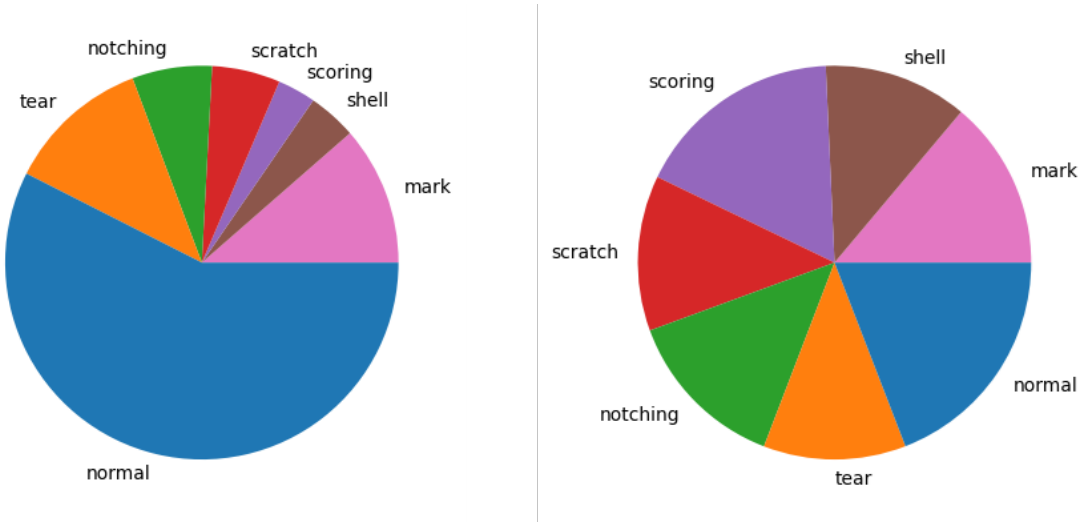


Figure 4.18: No augmented vs augmented dataset

Training

The classifiers have been trained for 1200 epochs using Adam with learning rate 0.0001 and weight decay set to 0.0001. We report in the figure below both the losses (Figure 4.19 and Figure 4.20) and the accuracy curves (Figure 4.21 and Figure 4.22) for both the classifiers. We can notice how the augmented data stabilized the training enhancing the generalization ability of the model.

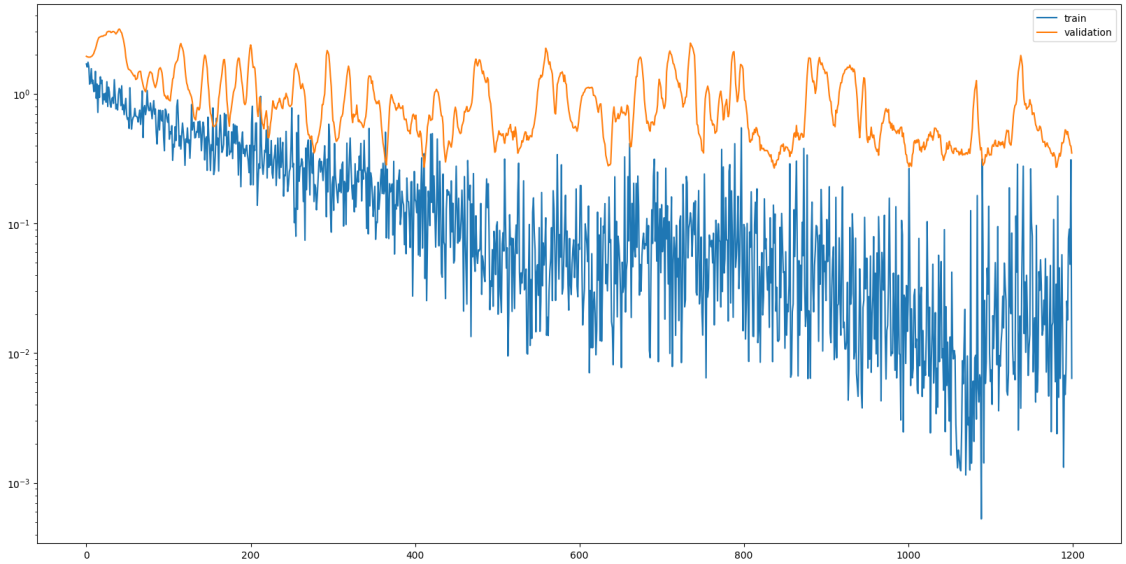


Figure 4.19: Loss ResNet50 classifier on no augmented data

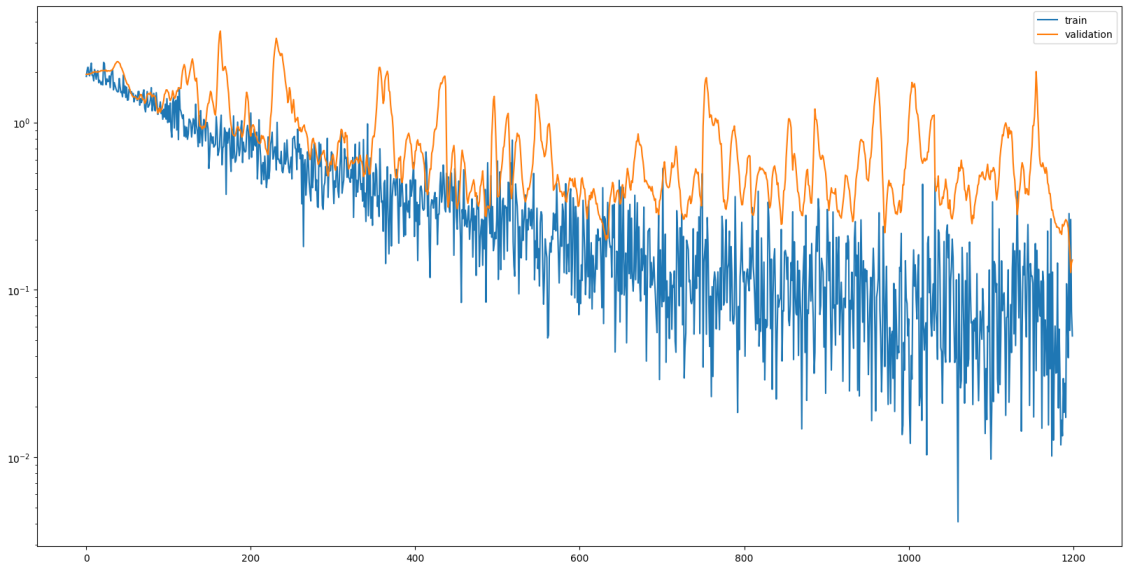


Figure 4.20: Loss ResNet50 classifier on augmented data

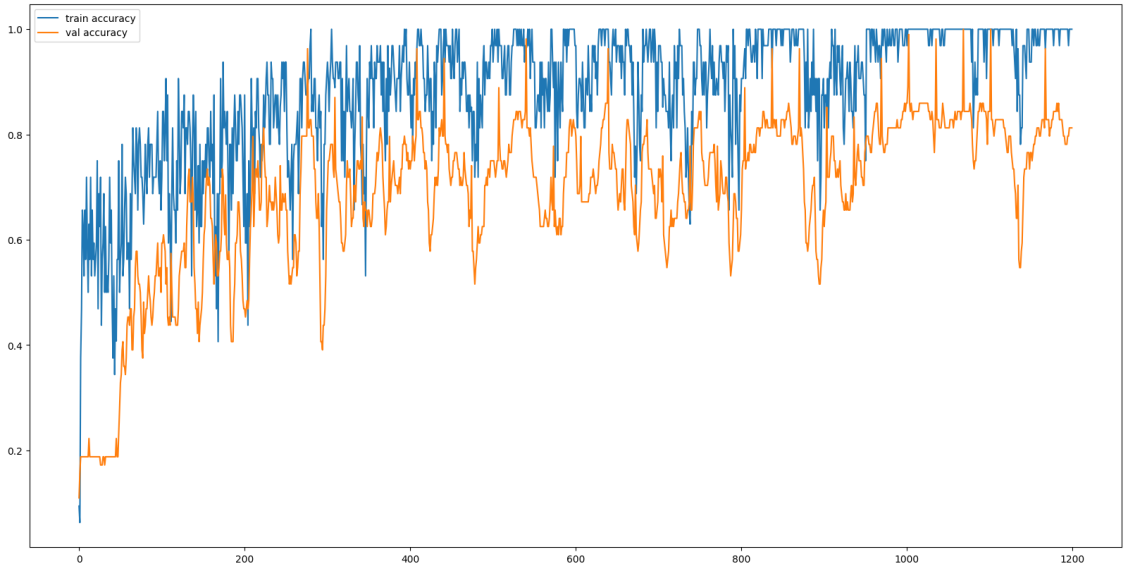


Figure 4.21: Accuracy curves of the ResNet50 classifier on no augmented data

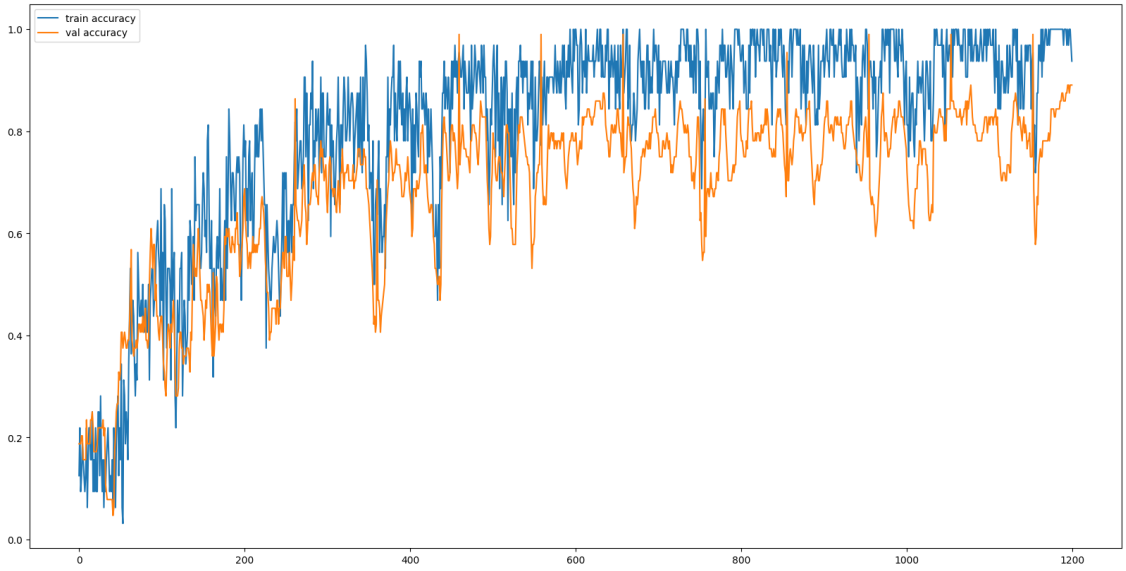


Figure 4.22: Accuracy curves of the ResNet50 classifier on augmented data

Results

As it is possible to see from the accuracy curves, our generative model improved the overall performance of the classifier by reducing the error rate from 9.03% to 5.42% on the test set.

Moreover, we conducted a study about the performance of the classifier on the single defect class. As shown in the table 4.8 the augmented set improved the classification about almost all the defect classes but the scratch.

	scratch	shell	mark	scoring	tear	notching	normal
No augmented	0%	77.8%	6.2%	11.1%	15.2%	0%	0%
Augmented	11.4%	11.1%	3.1%	0.0%	9.1%	0%	0%

Table 4.8: Error rate reported for each defect class obtained by training ResNet50 on the no augmented and augmented training set

From this last study we can state that transferring the defects across multiple background or changing its style can counteract the data-insufficiency problem in the real-world scenario.

5

Conclusion

This work aimed at proposing a GAN-based approach for overcoming the lack of defective images in the real-world scenario and in particular in the steel industry. We proposed a model that takes advantage of the disentangled representation learning for decomposing the images into two components representing so separately the defect and the background. Our model aims at achieving diversity by swapping the background code and thus transferring the defect from one surface to another. Through the experiments that we conducted we can state that our method is beneficial for improving the performance of a classifier. However, our generative network is confusing the structure with the texture in some cases which results in not transferring the defect or transferring it only partially. This might have occurred because the texture reference images contains patterns resembling defects. However, this behaviour is actually a drawback that can happen when employing the disentangled learning [9]. Thus, as future work we aim at improving the disentanglement mecha-

nism by tuning better the losses and/or adding more supervision such as classifying the texture. However, we might try to train the model for more epochs before doing any improvements. Indeed, because of the lack of computational power we didn't train the model for a number of epochs reasonable long considering that in the literature GAN-based model are trained also up to 500.000 interactions.
Defect generation is a field that was born recently which is proved by the fact that the related works in the literature are very scarce. Thus, more related works are to be expected for the upcoming years.

References

[1] Thomas Unterthiner Bernhard Nessler Martin Heusel, Hubert Ramsauer. Gans trained by a two time-scale update rule converge to a local nash equilibrium. 2018.

[2] Tao Qin Zhibo Chen Yingxue Pang, Jianxin Lin. Image-to-image translation: Methods and applications. 2021.

[3] SpyridonThermos Alison Q.O'Neil Sotirios A.Tsaftaris XiaoLiu, Pedro-Sanchez. Learning disentangled representations in the imaging domain. 2022.

[4] Biao Xu Tao Huang Cancan Yi, Qirui Chen. Steel strip defect sample generation method based on fusible feature gan model under few samples. 2023.

[5] Xinggang Wang andHuiLin Shuanlong Niu, Bin Li. Defect image sample generation with gan for improving defect recognition. 2020.

[6] Weixiang Liu Chongxin Hu Yuanlong Deng Zongze Wu Xiaopin Zhong, Junwei Zhu. An overview of image generation of industrial surface defects. 2023.

[7] Xinggang Wang HuiLin Shuanlong Niu, Bin Li. Defect image sample generation with gan for improving defect recognition. 2020.

[8] Xinggang Wang Songping He Yaru Peng Shuanlong Niu, Bin Li. Defect attention template generation cyclegan for weakly supervised surface defect segmentation. 2022.

[9] Yujiu Yang Jing-Hao Xue Weihao Xia a b. Unsupervised multi-domain multi-modal image-to-image translation with explicit domain-constrained disentanglement. 2020.

[10] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial nets. 2014.

[11] Léon Bottou Martin Arjovsky, Soumith Chintala. Wasserstein gan. 2017.

[12] Eduardo Monari Marco F. Huber Ruyu Wang, Sabrina Hoppe. Defect transfer gan: Diverse defect synthe- sis for data augmentation. 2023.

[13] Fabio Scotti Vincenzo Piuri Pasquale Coscia, Angelo Genovese. Applications and limits of image—to-image translation models. 2023.

[14] Frederik Hvilshøj Alexander Mathiasen. Backpropagating through fréchet inception distance. 2021.

[15] Eyal Betzalel et all. A study on the evaluation of generative models. 2022.

[16] Xinggang Wang andHuiLin Shuanlong Niu, Bin Li. Defect image sample generation with gan for improving defect recognition. 2020.

[17] Munyoung Kim Jung-Woo Ha Sunghun Kim Jaegul Choo Yunjey Choi, Minje Choi. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. 2017.

[18] Huan Liu Chao Wang Le Li Xun Shao Mingwen Shao, Youcai Zhang. Dmdit: Diverse multi-domain image-to-image translation. 2021.

[19] Wentao Jiang Chen Gao Tianyi Wu Qaingchang Wang Guodong Guo Defa Zhu, Si Liu. Ugan: Untraceable gan for multi-domain face translation. 2019.

[20] Yingce Xia Sen Liu Tao Qin Jiebo Luo Jianxin Lin, Zhibo Chen. Exploring explicit domain supervision for latent space disentanglement in unpaired image-to-image translation. 2019.

[21] Yujiu Yang Jing-Hao Xue Weihao Xia. Unsupervised multi-domain multimodal image-to-image translation with explicit domain-constrained disentanglement. 2020.

[22] Radu Timofte Luc Van Gool Asha Anoosheh, Eirikur Agustsson. Combogan: Unrestrained scalability for image domain translation. 2017.

[23] Gaurav Fotedar Nima Tajbakhsh Ziheng Zhou Lei He Weinan Song and Xiaowei Ding. Mdt-net: Multi-domain transfer by perceptual supervision for unpaired images in oct scan. 2022.

[24] Moustafa Meshry Yixuan Ren Larry S. Davis Abhinav Shrivastava. Step: Style-based encoder pre-training for multi-modal image synthesis. 2021.

[25] Phillip Isola Alexei A. Efros Jun-Yan Zhu, Taesung Park. Unpaired image-to-image translation using cycle-consistent adversarial networks. 2017.

[26] Serge Belongie Jan Kautz Xun Huang, Ming-Yu Liu. Multimodal unsupervised image-to-image translation. 2018.

[27] Timo Aila Tero Karras, Samuli Laine. A style-based generator architecture for generative adversarial networks. 2018.

[28] Jia-Bin Huang Maneesh Kumar Singh Ming-Hsuan Yang Hsin-Ying Lee, Hung-Yu Tseng. Diverse image-to-image translation via disentangled representations. 2018.

[29] Rein Houthooft John Schulman Ilya Sutskever Pieter Abbeel Xi Chen, Yan Duan. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. 2016.

[30] Arka Pal Loic Matthey Nick Watters Guillaume Desjardins-Alexander Lerchner Christopher P. Burgess, Irina Higgins. Understanding disentangling in β-vae. 2018.

[31] Miika Aittala Janne Hellsten Jaakko Lehtinen Timo Aila Tero Karras, Samuli Laine. Analyzing and improving the image quality of stylegan. 2019.

[32] Cheng He Shihua Huang and Ran Cheng. Sologan: Multi-domain multimodal unpaired image-to-image translation via a single generative adversarial network. 2022.

[33] Fei Shen Jing Wei, Zhengtao Zhang and Chengkan Lv. Mask-guided generation method for industrial defect images with non-uniform structures. 2022.

[34] Hyock Ju Kwon Yuwei Xia, Sang Wook Han. Image generation and recognition for railway surface defect detection. 2022.

[35] Yuan Qiu Xuefeng Ni Bo Shi Hongli Liu Jianwei Liu, Ziji Ma. Four discriminator cycle-consistent adversarial network for improving railway defective fastener inspection. 2022.

[36] Haiyong Chen Binyi Su, Zhong Zhou and Senior Member Xiaochun Cao. Sigan: A novel image generation method for solar cell defect segmentation and augmentation. 2015.

[37] Peigen Li Qiang Fang Haiting Xia Rongxin Guo Jiajun Song, Jiajun Song. Data augmentation by an additional self-supervised cyclegan-based for shadowed pavement detection. 2022.

[38] Xinggang Wang Yaru Peng Shuanlong Niu, Bin Li. Region- and strength-controllable gan for defect generation and segmentation in industrial images. 2021.

[39] Tzu-Yi Hung Shijian Lu Gongjie Zhang, Kaiwen Cui. Defect-gan: High-fidelity defect synthesis for automated defect inspection. 2021.

[40] Chao Zhi Mengqi Chen Youyong Zhou Shuai Wang Sida Fu Lingjie Yu Yuming Zhang, Zhongyuan Gao. A novel defect generation model based on two-stage gan. 2022.

[41] Fang-Lue Zhang Song-Hai Zhang Sen-Zhe Xu, Hao-Zhi Huang. Faceshapegene: A disentangled shape representation for flexible face image editing. 2021.

[42] Jianbo Yuan-S. Kevin Zhou-Jiebo Luo Haofu Liao, Wei-An Lin. Artifact disentanglement network for unsupervised metal artifact reduction. 2019.

[43] Yingying Zhu Jing Xiao Ronald M. Summers Youbao Tang, Yuxing Tang. A disentangled generative model for disease decomposition in chest x-rays via normal image synthesis. 2021.

[44] Huidong Liu Gagandeep Singh Jeremy Green Amit Gupta Dimitris Samaras Prateek Prasanna Lei Zhou, Joseph Bae. Lung swapping autoencoder: Learning a disentangled structure-texture representation of chest radiographs. 2022.

[45] Varun Agrawal Amit Raj Jingwan Lu Chen Fang Fisher Yu James Hays Wenqi Xian, Patsorn Sangkloy. Texturegan: Controlling deep image synthesis with texture patches. 2018.

[46] Wenbin Cai Jie Chang Yexun Zhang, Ya Zhang. Separating style and content for generalized style transfer. 2018.

[47] Yu Li Junbo Guo Yongdong Zhang Jintao Li Shuicheng Yan Rui Zhang, Sheng Tang. Style separation and synthesis via generative adversarial networks. 2018.

[48] Lior Wolf Amit Bermano Ron Mokady, Sagie Benaim. Mask based unsupervised content transfer. 2020.

[49] Taesung Park et all. Swapping autoencoder for deep image manipulation. 2020.

[50] Tim Salimans et all. Improved techniques for training gans. 2016.

[51] Charalambos Poullis Shima Shahfar. Unsupervised structure-consistent image-to-image translation. 2022.

[52] Tomer Galanti Sagie Benaim Ori Press. Emerging disentanglement in auto-encoder based unsupervised image content transfer. 2019.

[53] Haofu Liao et all. Adn: Artifact disentanglement network for unsupervised metal artifact reduction. 2019.