Multistage GAN for Fabric Defect Detection

Juhua Liu, Chaoyue Wang, Hai Su, Bo Du, *Senior Member, IEEE*, and Dacheng Tao, *Fellow, IEEE*

***Abstract*— Fabric defect detection is an intriguing but challenging topic. Many methods have been proposed for fabric defect detection, but these methods are still suboptimal due to the complex diversity of both fabric textures and defects. In this paper, we propose a generative adversarial network (GAN)-based framework for fabric defect detection. Considering existing challenges in real-world applications, the proposed fabric defect detection system is capable of learning existing fabric defect samples and automatically adapting to different fabric textures during different application periods. Specifically, we customize a deep semantic segmentation network for fabric defect detection that can detect different defect types. Furthermore, we attempted to train a multistage GAN to synthesize reasonable defects in new defect-free samples. First, a texture-conditioned GAN is trained to explore the conditional distribution of defects given different texture backgrounds. Given a novel fabric, we aim to generate reasonable defective patches. Then, a GAN-based fusion network fuses the generated defects to specific locations. Finally, the well-trained multistage GAN continuously updates the existing fabric defect datasets and contributes to the fine-tuning of the semantic segmentation network to better detect defects under different conditions. Comprehensive experiments on various representative fabric samples are conducted to verify the detection performance of our proposed method.**

***Index Terms*— Fabric defect detection, deep learning, semantic segmentation, generative adversarial network.**

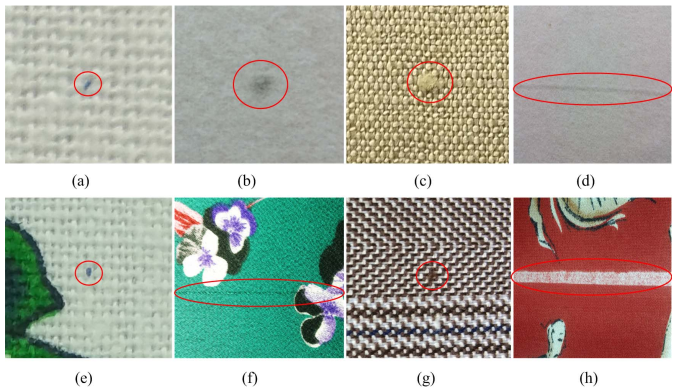

Fig. 1. Defect samples in simple- and complex-textured fabrics. From (a) to (d), the defects are classified as color spots, oil stains, knots, and broken ends in simple texture fabrics. From (e) to (h), the defects are classified as color spots, broken ends, broken yarn, and white strips in complex texture fabrics.

I. Introduction

FABRIC defect detection is an important quality control procedure that aims to identify and locate defects appearing in fabric [1]–[3]. Automated fabric defect detection is one of the most direct applications of artificial intelligence algorithms in an industrial setting, the effective implementation of which could effectively improve fabric quality and reduce labor costs.

Therefore, fabric defect detection is of considerable interest to both industrial and academic researchers. The aim is to detect various abnormal patterns in complex backgrounds. Many algorithms have been proposed for detecting defects under different assumptions [4]–[8]. Abouelela *et al.* [6] assumed that fabric texture is a composite of simple structures and judged a region with a different structure as a defect. Since there are significant differences between defective and defect-free textures in the frequency domain, Chan and Pang [7] used a Fourier transform to separate defective and defect-free regions. However, this spectral approach is not suitable for complex texture fabrics. Recently, deep learning has had a huge impact on many computer vision tasks. Some methods [9]–[11] have applied deep neural networks to fabric defect detection. Based on learned subdictionaries, Tong *et al.* [9] adopted a nonlocally centralized sparse representation (NCSR) model to reconstruct the input images; the resulting defect was segmented from the residual images of the reconstructed images and the input images. Li *et al.* [11] trained a Fisher criterion-based stacked denoising autoencoder (FCSDA) with equal-sized fabric image patches to classify test patches into defective or defect-free patches, the residual between the reconstructed images and the defective patches representing the defect location. This method achieved promising results on regularly patterned and more complex jacquard warp-knitted fabric. Despite these methods achieving promising performance in specific situations, most are limited to simple

Manuscript received May 1, 2019; revised September 6, 2019 and October 1, 2019; accepted November 29, 2019. Date of publication December 19, 2019; date of current version January 30, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61601335, Grant 61822113, and Grant 41871243, in part by the National Key R&D of China under Grant 2018YFA060550, in part by the Natural Science Foundation of Hubei Province under Grant 2018CFA050, in part by the Science and Technology Major Project of Hubei Province (Next-Generation AI Technologies) under Grant 2019AEA170, in part by the PhD Start-up Fund of Natural Science Foundation of Guangdong Province of China under Grant 2017A030310437, and in part by the Special Project on Industrial Technology Innovation of Suzhou under Grant SYG201714. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Khan M. Iftekharuddin. *(Juhua Liu and Chaoyue Wang contributed equally to this work.) (Corresponding author: Bo Du.)*

J. Liu is with the School of Printing and Packaging, Institute of Artificial Intelligence, Wuhan University, Wuhan 215123, China, with the UBTECH Sydney Artificial Intelligence Centre, School of Computer Science, Faculty of Engineering and Information Technologies, The University of Sydney, Darlington, NSW 2008, Australia, and also with the Suzhou Institute, Wuhan University, Suzhou 215123, China (e-mail: liujuhua@whu.edu.cn).

C. Wang and D. Tao are with the UBTECH Sydney Artificial Intelligence Centre, School of Computer Science, Faculty of Engineering and Information Technologies, The University of Sydney, Darlington, NSW 2008, Australia (e-mail: chaoyue.wang@sydney.edu.au; dacheng.tao@sydney.edu.au).

H. Su is with the School of Software, South China Normal University, Guangzhou 510631, China (e-mail: suhai@m.scnu.edu.cn).

B. Du is with the National Engineering Research Center for Multimedia Software, Institute of Artificial Intelligence, School of Computer Science, Wuhan University, Wuhan 215123, China (e-mail: remoteking@whu.edu.cn).

Digital Object Identifier 10.1109/TIP.2019.2959741

textures and fail to solve complex real-world fabric defect detection problems.

To improve real-world fabric defect detection performance, several challenges need to be solved. First, labeling the various defects in real-world products is time consuming and laborious. Because of the complex diversity in both fabric products and defects, it is hard to collect an annotated dataset to cover all possible fabric textures. Thus, when dealing with fabrics with unseen textures or materials, pretrained detection models usually fail to perform well. Moreover, since production processes and materials differ for each fabric type, the appearances and characteristics of different defects vary widely, which also makes fabric defect detection more difficult. Samples with different defects in simple and complex fabrics are shown in Fig. 1.

In this paper, we propose a generative adversarial network (GAN)-based framework for fabric defect detection. Considering the above challenges in real-world applications, the proposed fabric defect detection system is capable of learning existing fabric defect samples and automatically adapting to different textures during different application periods. Specifically, we train a generative model that aims to learn the pattern and distribution of fabric defects based on existing annotated defective samples. Since the defect appearances are dependent on background textures, we assume that fabric defects and their corresponding background textures obey a conditional distribution. Given a background texture style, the trained generative model aims to synthesize defects that can be added to the fabric background. Thus, it is unnecessary to collect and annotate training samples when processing a new fabric with a different background texture. We propose a multistage GAN to synthesize defects of unseen defect-free fabric samples. First, a texture style-conditional GAN is trained to synthesize rough defects conditioned on a given fabric's texture. Then, we fuse the defects generated in stage one with the corresponding fabric to obtain the final generated defective fabric samples.

Moreover, we train a semantic segmentation network for fabric defect detection based on both the existing and generated defective fabric samples. This semantic segmentation network is capable of detecting multiscale defects and can be fine-tuned to fit with the newly generated defective samples. Finally, similar to real-world applications, we collect a dataset for fabric defect detection and conduct experiments on this dataset. The results show that our proposed fabric defect detection system can detect different defect types in different fabrics.

In summary, this paper makes the following contributions. 1) We formulate the industrial fabric defect detection task into a GAN-based task and propose a novel fabric defect detection system. 2) We propose a multistage GAN to synthesize reasonable defects in unseen defect-free fabric textures, which can be applied to update our detection model to fit continuously appearing novel fabric textures. 3) We train a semantic segmentation network for fabric defect detection that can detect different types and scales of defects in both simple and complex background textures. 4) The experimental results show that our fabric defect detection system achieves outstanding performance in a series of fabric defect detection challenges.

The rest of the paper is organized as follows. In Section II, we briefly review semantic segmentation networks and GANs. In Section III, we introduce our general framework before analyzing each module. Section IV reports and discusses our experimental results. Finally, we conclude our study in Section V.

II. Related Works

A. Semantic Segmentation

Semantic segmentation is currently a key computer vision task [12]–[14]. The final output of semantic segmentation is an image with each of its pixels inferring a label for the corresponding pixel in the input image. With the revolution of deep learning, semantic segmentation problems are increasingly being addressed using convolutional neural networks (CNNs). CNNs employ successive pooling operations and striding in convolution to enlarge the receptive field to learn more abstract features while the output feature resolution is reduced. Therefore, semantic segmentation faces an inherent tension between semantics and location: global information resolves $what$ the defect is while local information resolves $where$ the defect is.

To solve the problem of reduced feature resolution, fully convolutional networks (FCNs) [12] modify existing classification networks by replacing all the fully connected layers with convolutional layers to obtain a spatial feature map instead of classification scores. Then, upsampling (also known as a transposed convolution) is utilized on the reduced-scale features to output a dense pixelwise-labeled map. SegNet [13] uses another variant to transform the classification network for segmentation. Specifically, SegNet maps the low-resolution image representation produced by the encoder to pixelwise predictions and uses a decoder composed of some upsampling blocks and one softmax layer at the end. Each upsampling block has an upsampling layer and several successive convolution layers, and their max pooling indices correspond to the pooling layers in the encoder stage.

Other methods exist to integrate global context knowledge into CNNs, such as conditional random fields (CRFs), atrous convolution, and feature fusion. CRFs [15] are a common approach for segmentation result refinement that are usually applied as a postprocessing step. CRFs can combine pixel-level information such as pixelwise interactions with the output label predictions, hence capturing the long-range correlations of pixels that are ignored in CNNs, $e.g.$, Chen *et al.* [16], [17] used the fully connected pairwise CRF in DeepLab models as a standalone postprocessing step of the framework, such that the system recognized detailed structures taking both short and long correlations into consideration. Atrous convolution, also known as dilated convolution, is an alternative to the consecutive combinations of pooling and deconvolution used in most CNNs. Atrous filters use the atrous rate to control the sampling stride, and by modifying the atrous rate, the receptive field of the filter is changed while the size of the output feature map is maintained because no striding is used. DeepLab models [16]–[19] also take advantage of atrous convolutions by setting an incremental atrous rate for larger receptive fields without the cost of learning extra parameters. Feature fusion [20]–[22] is also a fusing method for integrating global information and can be categorized into early fusion and late fusion. Early fusion first unpools the global feature to the same size as the local feature and then concatenates the global and local features to train a single classifier, while late fusion can be viewed as fusing all predictions made by classifiers at each layer to produce the final prediction results. Note that multiscale prediction and feature fusion are also beneficial for multiscale object segmentation.

B. Generative Adversarial Networks

Generative adversarial networks (GANs) [23] have also received considerable attention over the last few years. Due to their performance learning real distributions, GANs and their variants have been widely used in a series of image processing, computer vision, and computer graphic tasks such as image translation, video generation, and 3D image generation. Overall, GANs consist of a generator and a discriminator that play an adversarial game with each other to explore the training data distribution. Taking a noisy sample $z \sim p_z$ as the input, the generator G outputs new data $G(z)$, whose distribution p_g is supposed to be close to that of the data, p_{data}.. At the same time, the discriminative network D is employed to distinguish the true data sample $x \sim p_{data}$ and the generated sample $G(z) \sim p_g(G(z))$. In the original GAN, this adversarial training process is formulated as:

$$\underset{G}{min}\,\underset{D}{max}\,Æ_{x\sim p_{data}}[log D(x)] + Æ_{z\sim p_z}[log(1 - D(G(z)))] \tag{1}$$

According to the adversarial objective function, the discriminator is trained to maximize the log-likelihood of assigning the correct label to both the (real) training samples and the (fake) generated samples, while G is trained to minimize the objective function in an attempt to prevent the discriminator from assigning a fake label to generated samples. Accompanying this adversarial process, the generator G can be trained to synthesize even more realistic samples.

In addition to generating images from noisy samples, GANs have been introduced to solve many real-world applications [24]–[36]. . For example, pix2pix [24] was proposed for image-to-image translation. By introducing a GAN loss between paired image samples, pix2pix successfully solved a series of image translation tasks. Recently, Wang *et al.* [25] devised a novel framework for synthesizing high-resolution images that were capable of semantically manipulating generated images. Moreover, other GAN variants [26]–[28] have focused on cross-domain image translations by exploring the cyclic mapping (or primal-dual) relationship between different image domains. Specifically, a primal GAN aims to explore the mapping relationship from source images to target images, while a dual (or inverse) GAN performs the inverse task. These two GANs form a closed loop and allow images from either domain to be translated and then reconstructed. By combining the GAN loss and cycle consistency loss (or recovery loss), these models can be used for image translation tasks in the absence of paired examples. Last but not least, many GANs

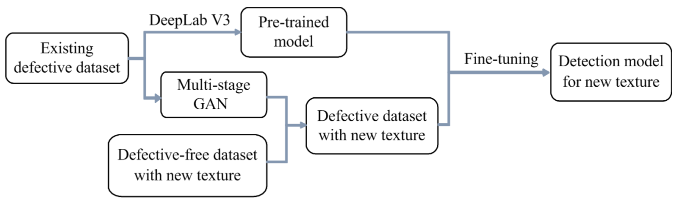
Existing defective dataset
DeepLab V3
Pre-trained model
Multi-stage GAN
Defective-free dataset with new texture
Defective dataset with new texture
Fine-tuning
Detection model for new texture

Fig. 2. The architecture of our proposed system for fabric defect detection.

have been proposed for other image manipulation tasks, such as the SRGAN [29] for super-resolution, ID-CGAN [30] for image de-raining, iGAN [31] for interactive applications, IAN [32] for photo modification, and Context Encoders [33] for image in-painting.

III. Methodology

In this section, we first introduce our fabric defect detection pipeline before discussing the details of the three main modules: 1) the segmentation network for fabric defect detection, 2) the multi-stage GAN for defective fabric data generation, and 3) the fine-tuning strategy for adapting the pretrained model to the new fabric texture.

A. Framework Description

As noted above, in real-world applications, insufficient annotated datasets and the complex diversity of both fabric textures and defects are significant challenges for fabric detection methods. Given such circumstances, we propose a GAN-based system for fabric defect detection, which is capable of training a detection model on existing fabric defect samples and automatically adapting to different textures during different application periods.

Fig. 2 depicts our proposed fabric defect detection system. The inputs of the system are one existing defective fabric dataset and one defect-free fabric dataset in which the textures are very different from the existing defective dataset. First, the existing defective fabric dataset is used to train the fabric segmentation network, which is customized for defect detection. Based on the state-of-the-art semantic segmentation network DeepLab V3 [18], we modify the atrous rate of some of the atrous convolutional layers and add a weight parameter to the loss function to enhance it for fabric defect segmentation. Moreover, based on both the defective and the defect-free datasets, we aim to synthesize more defective samples with different textures ($i.e.$, backgrounds). Thus, these synthesized defective samples can be regarded as additional training samples for the detection model to further improve its performance. Specifically, we devised a multistage GAN-based module to synthesize new defective fabric samples. First, a texture-style-conditional GAN network is trained to generate defective patches based on given (conditioned) textures. Then, we utilize the adversarial loss to train a defect-fusing network, which aims to fuse generated defects into defect-free samples. Considering that most defects are highly related to their background texture and colors, we assume that the appearance of defects obeys a conditional distribution

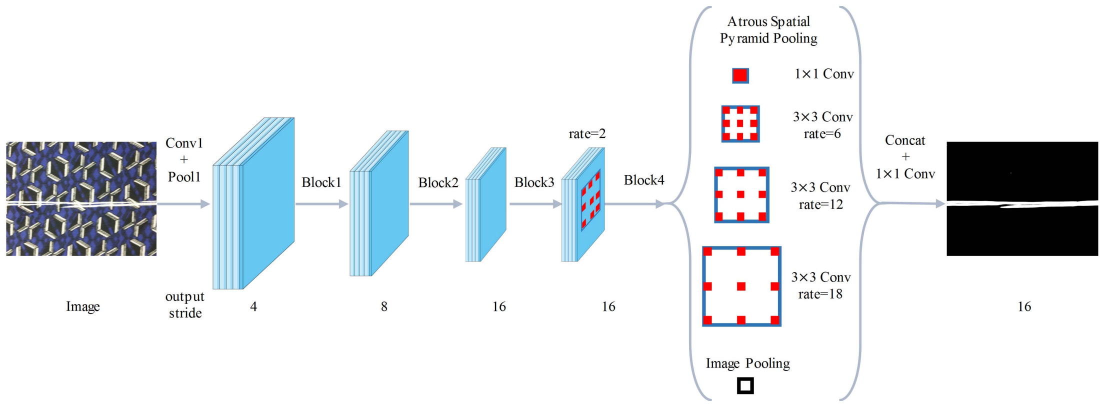

Fig. 3. The architecture of DeepLab V3. Image-level features augmented by parallel atrous convolution modules (ASPP) are included.

given a specific fabric background. Therefore, we first train a conditional GAN to explore such relationships between defects and their surroundings and aim to generate reasonable defects on novel fabric backgrounds. However, the generated defects are based only on the 'style' information of the given backgrounds and are hard to fuse smoothly into a specific location. Therefore, we further train a defect-fusing network to synthesize realistic defective samples. Then, we fine-tune the pretrained segmentation model using the generated defective dataset. Thus, the model is capable of continuously detecting defects in the new texture.

B. Segmentation Network

With respect to the choice of the base defect detection network, it is unwise to choose from popular object detection networks such as Faster-RCNN [37], YOLO [38], or SSD [39] because the defects usually vary widely in scale and aspect ratio, which may reduce the detection accuracy. Instead, we choose a semantic segmentation network, DeepLab V3, proposed by Chen *et al.* [18], as our base network. The DeepLab V3 architecture is shown in Fig. 3. Following this work, the *output stride* is defined as the ratio of the original image size to the size of the output feature map.

Deep CNNs usually use several cascaded convolutional blocks that include several convolutions and strides for feature extraction, hence enlarging the receptive fields of the filters and shrinking the extracted feature maps through the chain of blocks. To preserve the size of the feature maps in the deeper layers, DeepLab models use atrous convolution instead of convolution and striding. We use ResNet-50 [40] as the backbone network, which contains four blocks, each including three 3×3 convolutions and the last convolution containing a stride of 2 except for block4. Then, atrous convolution is used as the alternative to successive striding, and the atrous rates are set according to the desired *output stride* values.

Another challenge is the accurate detection of defects at various scales. Inspired by spatial pyramid pooling [41], DeepLab V3 deploys an atrous spatial pyramid pooling (ASPP) module on the top of the feature map. Specifically, the module contains four parallel atrous convolutions with different atrous rates, which sample features at different scales to effectively capture multiscale information. As a side note, batch normalization is included in each convolution. To extract global features, one method is to set a large atrous convolution rate. However, if the atrous rate is set to a value as large as the feature map size, then the outer filter weights will be applied on padded zeros, and thus the filter will degrade into a smaller one that might not capture global information. In this case, DeepLab V3 adds a global information branch to the ASPP module that consists of average pooling, a 1×1 convolution, and upsampling. Therefore, five branches are included in the final ASPP module: one image-level information branch, one 1×1 convolution, and three 3×3 convolutions with *atrous rates* = (6, 12, 18) and an *output stride* = 16. Then, the features from all the branches are concatenated, fed to one 1×1 convolution with batch normalization, and forwarded to a final 1×1 interpreting convolution.

When the semantic segmentation network is used for fabric defect detection, three problems should be noted:

1) The final *output stride* represents the size ratio of the input image to the output feature map, which means that if we set the *output stride* too large, then the final feature map will be so small that tiny defects are easily ignored. Under such a constraint, we set our desired *output stride* to 8 instead of 16 by also applying atrous convolution in block3. We set the block3 rate to 2 and the block4 rate to 4, and the rates in the ASPP module are doubled to (12, 24, 36) accordingly. As a result, the striding is altered by atrous convolution, and the final *output stride* equals the *output stride* of block2, which is 8. In this way, the size of the final output feature map is doubled, and tiny defects are preserved. Our experiments show that setting the final *output stride* to 8 preserves most tiny defects.

2) Semantic segmentation methods usually apply fully connected CRFs as a postprocessing stage for edge refinement. However, in fabric defect detection, we are only interested in whether the defects are detected, and the accuracy of the detected edge is not of primary concern. Moreover, considering speed and efficiency, we do not apply CRFs for edge refinement in our framework.

3) We also consider modifications to the loss function, as defects are usually far smaller than the background texture. Hence, we add different weights to the defect segmentation

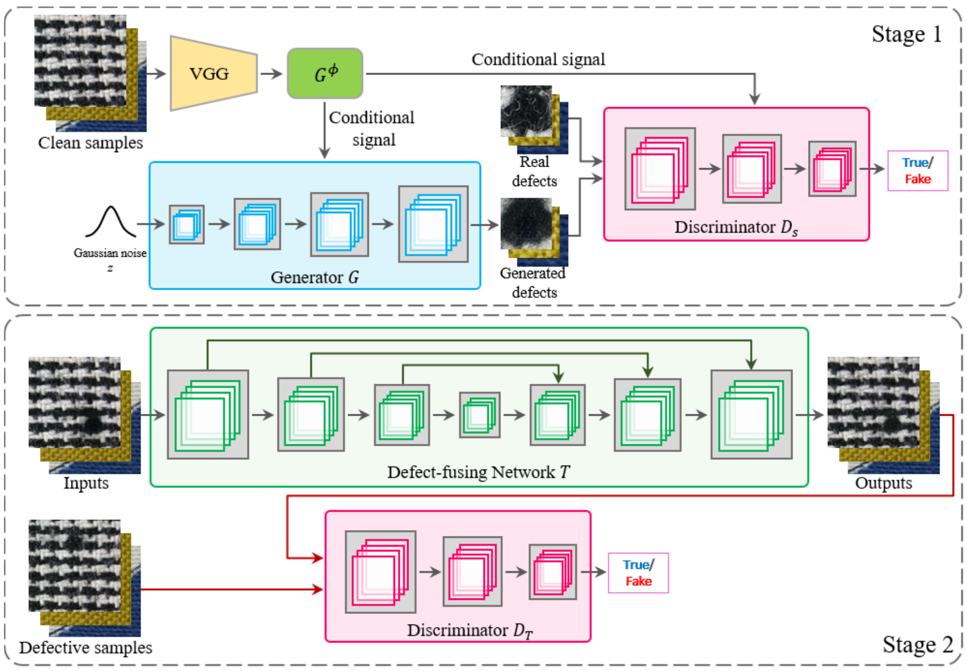

Fig. 4. Schematic of a multistage GAN. Stage 1: Generating defects based on the given texture; Stage 2: Fusing the generated defect into the defect-free sample.

loss and the background segmentation loss with the aim of improving the detection accuracy and speeding up the convergence of the loss function. The weighted loss function is shown in Eq. 2, where $L_{background}$ denotes the segmentation loss of the background texture and L_{defect} denotes the segmentation loss of the foreground defect. α_1 and α_2 are their weights, and in our case, we set α_1=1 and α_2=9.

$$L = \alpha_1 L_{background} + \alpha_2 L_{defect} \tag{2}$$

C. Synthesizing Novel Defective Samples Using a Multistage GAN

In real-world applications, since fabric textures are multifarious and continuously updated, we must consider the weak transferability issues of existing fabric defect detection methods. To maintain our method's adaptability, we propose a multistage GAN-based module for synthesizing defective fabric samples with novel textures. As one type of data augmentation solution, synthesized defective samples are utilized to further fine-tune our defect detection network. Thus, we no longer need to collect and label defective fabric samples with novel textures, instead simply synthesizing new defective fabric samples based on continuously updated defect-free samples. To synthesize reasonable and realistic defects, we first generate standalone defects according to the given texture and then attempt to fuse them into the corresponding fabric background. Specifically, in the first stage, a conditional GAN (CGAN) [42] [43] is trained for standalone defect generation, which is supposed to generate different types of defects based on the input condition, that is, the background texture in our case. Moreover, if we directly paste the generated defects on the fabrics, the defects still do not look real enough as they do not blend in with the surrounding area very well. Therefore, in the second stage, we propose a defect-fusing network that properly fuses the generated defects into defect-free fabric images.

Stage 1. First, we attempt to use the 'style label' to represent the background texture style of novel fabric samples. Specifically, we collect fabric patches as the input of stage 1 by randomly cropping from areas without defects. A pretrained VGGNet is then used to extract features from these patches. Then, the Gram matrix [44] can be computed from the features and be used as style labels. Define $\phi(x)$ as the feature map of shape $C \times H \times W$. By reshaping $\phi(x)$ into a $C \times HW$ matrix ψ, its Gram matrix $G^{\phi}(x)$ can be computed as:

$$G^{\phi}(x) = \frac{\psi\psi^T}{CHW} \tag{3}$$

Given the 'style label' $G^{\phi}(x)$, we train the generator G to generate corresponding defects. We define a real defect cropped from the original defective sample and its corresponding style label as a real pair and a generated defect and its corresponding style label as a fake pair. Suppose the distribution of real defects x is $p_{x|G^{\phi}}$ and the distribution of the noise z is p_z, then the conditional adversarial loss is computed by:

$$\underset{G}{min}\underset{D_s}{max} L(D_s, G) = Æ_{x\sim p_{x|G^{\phi}}}\left[log D_s(x, G^{\phi})\right] + Æ_{z\sim p_z}\left[log(1 - D_s(G(z), G^{\phi}))\right] \tag{4}$$

where G^{ϕ} is the corresponding style label of x. D_s is trained to better distinguish real and fake pairs, while G is trained to generate more realistic defects that aim to fool D_s. Along with the adversarial training process, the 'style labels' extracted from the defect-free fabric textures are regarded as the conditional signal for generating corresponding defects. Given different fabric textures (*i.e.*, style labels) as inputs, defects with different colors and textures should be synthesized. Finally, the conditional distribution of defects given textures will be learned, and reasonable defects given novel fabric samples can be generated.

Stage 2. After the defects are generated, we fuse the defects and textures using a defect-fusing network as depicted in stage 2 of Fig. 4. First, the defective zones from the training patches are cropped out, leaving only the defect-free fabric patches with blank windows. Then, the generated defects are resized and pasted onto the blank windows, thus obtaining the imperfect inputs x for the defect-fusing network. To be well defined, we name the generated patches $T(x)$ and the training patches as real images y. Note that the defect-fusing network is trained to fuse different generated defects into their corresponding backgrounds; *i.e.*, all training defective samples (with different textures) are utilized as real data during adversarial training. In addition to the adversarial loss, we devise three losses to fine-tune the network:

1) We design a reconstruction loss between the output fake image and the real defective one since we suppose that the generated defect-fused image should be similar to the real defective patch. However, since it is not reasonable to assume that the generated defective sample is the same as the original, we utilize the hinge reconstruction loss:

$$\$_{rec} = max(0, \|y - T(x)\| - m) \tag{5}$$

where m defines the hinge that relaxes the reconstruction constraint.

2) Inspired by [44], we impose a constraint on the feature map of fused patches extracted by a pretrained VGGNet and penalize it to be similar to the corresponding real patch. In this way, the generator is forced to fuse defects while preserving the principal features of the defects and the textures as much as possible. The feature reconstruction loss between the features of real and fake images is computed by:

$$l_{feat}^{\phi,j}(T(x), y) = \frac{1}{C_j H_j W_j} \left\|\phi_j(T(x)) - \phi_j(y)\right\|_2^2 \tag{6}$$

where $\phi_j(\cdot)$ represents the activation from the jth layer of the VGGNet of shape $C_j \times H_j \times W_j$. $T(x)$ and y denote the fake and real images, respectively.

3) The main difference between the fake image and the real image lies at the junction of the defect and the background. Therefore, a defective fusing discriminator D_T is employed for distinguishing real defective samples from synthesized novel samples. The discriminator D_T is mainly trained to inspect junction areas, while the defect-fusing network T is forced to synthesize samples with improved and well-fused defects. In this way, a minimax game is formed between them, and the adversarial loss function is defined as:

$$\underset{T}{min}\,\underset{D_T}{max} L(D_t, G) = Æ_{y\sim p_y}[log D_T(y)] + Æ_{x\sim p_x}[log(1 - D_T(T(x)))] \tag{7}$$

where $T(\cdot)$ represents the proposed defect-fusing network and x and y are the imperfect and training defective samples, respectively.

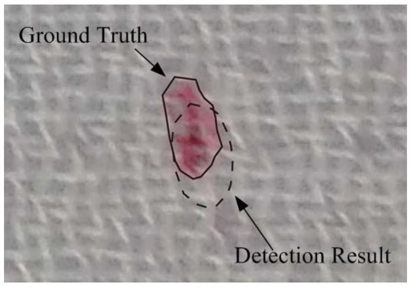

Fig. 5. Illustration of the IoU of the ground truth and the detection result.

After separately training the multiple stages of the defect-generating module, the network can generate reasonable defective images given a novel fabric patch. During testing, defect-free fabric samples with new textures are used as input into this module; thus, we obtain defective patches with new textures with well-fused defects. Several groups of generated defective samples and their corresponding detection results are shown in Section IV to demonstrate the reliability of this generative module.

D. Fine-Tuning Strategy

Due to the complex diversity of both fabric textures and defects, it is hard to collect an annotated defective fabric dataset that covers all possible fabric textures, and it is also difficult to train a general detection model that adapts to all fabrics. Fortunately, it is easy and feasible to collect defect-free samples of the fabric to be detected, and those samples can then be used as the input of the above multistage GAN to generate defective samples, which are used to continuously update the detection model. In this way, we use those generated defective samples to train the fabric defect detection model. To make the model converge faster, we pretrain a defect detection model using existing annotated defective samples, which are also used to train the multistage GAN model. Given a fabric with a new texture to be detected, we use the defect-generating module to synthesize defects in the defect-free fabric samples. Finally, the pretrained model is used for detection network initialization, the generated defective samples are fed into the network, and the resulting model can then be used to detect defects of the given fabric. Therefore, using our proposed GAN-based system, the detection model can automatically adapt to different textures during different application periods.

IV. Experiments

In this section, we evaluate our proposed fabric defect detection system by conducting experiments on representative defective fabric datasets. In Section A, we introduce how we evaluate the performance of our proposed method. The detailed experimental settings are described in Section B. The defect generation results and detection results on simple texture fabrics and complex texture fabrics are presented in Sections C and D, respectively. Further analysis is provided in Section E.

A. Evaluation Metrics

In fabric defect detection, we are more concerned with whether the defects are detected than whether every pixel is predicted correctly. Therefore, we use an evaluation metric that classifies a defect as either detected or undetected. We use the intersection over union (IoU), which is the ratio of the intersection area to the union area for the ground truth and our detected polygonal region, as shown in Fig. 5. The IoU represents how closely our detection result matches the ground truth. As we do not need the resulting polygon to perfectly match its ground truth, we set the threshold for whether the defect is detected to 0.5.

Next, successfully detected defects are categorized as true positives (TPs), mistaken detections are categorized as false positives (FPs), and undetected defects are categorized as false negatives (FNs). With these three indicators, we calculate the $precision$, $recall$, and $F{-}measure$ as the evaluation metrics, and their formulae are as follows:

$$precision = \frac{TP}{TP + FP} \times 100\% \tag{8}$$

$$recall = \frac{TP}{TP + FN} \times 100\% \tag{9}$$

$$F - measure = 2 \times \frac{precision \times recall}{precision + recall} \tag{10}$$

As the general rule, a higher $F - measure$ reflects a better detection performance.

B. Experimental Steps

We next describe our experimental settings in detail. For clarity, we split our experiment into three parts according to our proposed detection system in Section III: detection model pretraining, defective sample generation for new textures, and model fine-tuning.

In the first step, we collect defective fabric images as our original dataset and categorize them according to the cause of the defect. We collected four defect types in both simple and complex texture fabrics. Due to different production processes and materials, the fabrics with simple textures and complex textures also have different defect types: color spot, oil stain, knot and broken end in simple texture fabric, and color spot, broken end, broken yarn, and white strip in complex texture fabric, as shown in Fig. 1. We categorize these defect types with the aim of improving different aspects of our method. In short, color spots test the ability to detect small-scale defects, oil stains are used to enhance fuzzy edge detection, knots and broken yarn defects train our method for improving sharp edge detection, and broken ends and white strips require our model to have a wider field to capture large-scale defects. Note that our purpose is to detect defects not to classify the types of defects.

For data preprocessing, we randomly crop the defective samples to 512×512 according to the defect positions. Since our defective fabric dataset is not large enough to train a general model, to extend the variability of the dataset and avoid overfitting, we adopt data augmentation methods including hue, brightness, rotation, and mirroring to the defective samples. For clarity, we name this preprocessed dataset $dataset0$. After preparing the original defective fabric dataset, we use $dataset0$ to train our DeepLab V3 network. When the training procedure is complete, we obtain the pretrained detection model as the output of step 1. This pretrained model detects defects in textures present in $dataset0$ well. However, as discussed above, because the background texture represents most of the fabric image, there is a significant difference between two fabrics when they have very different textures, which may cause the pretrained model to fail.

In the second step, we focus on generating defective samples for the detected fabric, the texture of which may be very different from the textures in $dataset0$. For a given fabric to be detected, we can easily collect defect-free samples of this fabric and apply our multistage defect-generating module to generate defective samples for the detected fabric. In stage 1, a conditional GAN is trained with style labels and real defects to generate different types of defects. In stage 2, another GAN is trained with composite images and real defective images to generate images with properly fused defects. A hinge reconstruction loss and perceptual loss are also applied to further constrain the generating module. After training, the module is fed with defect-free samples to generate defective $dataset1$ with new textures. $Dataset1$ is mixed with a random collection of $dataset0$ at a ratio of 1:5 to form the mixed $dataset2$.

In the final step, we use the mixed-source $dataset2$ to fine-tune the pretrained model from step 1. Specifically, we use our pretrained model as the initialization of the DeepLab V3 defect detection network, and then $dataset2$ is used as the training dataset to further fine-tune the network. The network is supposed to be able to extract different features for the detected fabric, so the detection ability of the final fine-tuned model, which is the output of step 3, is improved with regard to the new texture samples.

Finally, we apply the fine-tuned model in the testing phase using real defective samples. The evaluation criteria are described above, and here we use these criteria to evaluate our detection results. The TPs, FPs, and FNs in different experimental settings are counted, and the corresponding assessment criteria, *i.e.*, the $precision$, $recall$, and $F - measure$ values, are calculated.

C. Results of Generating Defective Samples

We first train the proposed multistage defect-generating module to synthesize defective samples. Some generated fabric samples from both stages are reported, and we attempt to qualitatively analyze the effects and advantages of different stages. Since simple-textured fabrics are distinct from fabrics with complex textures, we divide the collected original defective fabric images into two categories: simple-textured fabric samples and complex-textured fabric samples. Each contains four different defect types. Our goal is to learn the patterns of different defect types and further add them into new defect-free images. Since these new synthesized 'defective' fabric images will be used to fine-tune the pretrained detection model, we hope that these synthesized fabric images are as realistic as possible.

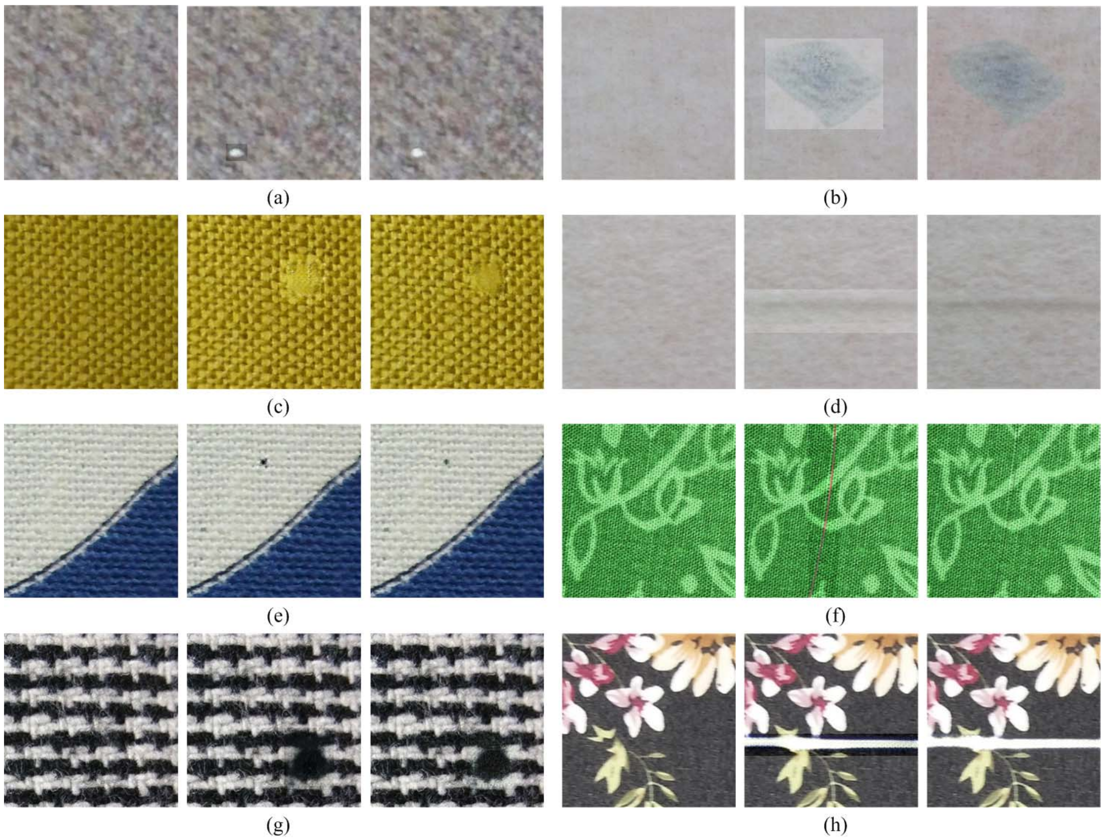
Fig. 6. Samples of synthesized defective fabric samples. Defect types are (a) color spot, (b) oil stain, (c) knot, and (d) broken end in simple-textured fabric and (e) color spot, (f) broken end, (g) broken yarn, and (h) white strip in complex-textured fabric.

In Fig. 6, we present some synthesized samples from each stage of our GAN-based module. Among them, the first four sets, $i.e.$, (a), (b), (c), and (d), correspond to four synthesized defects in simple texture fabric. The other four sets, $i.e.$, (e), (f), (g), and (h), show the defects added to the complex-textured fabric. Within each set, the first (leftmost) image represents the original defect-free fabric image. Note that these input defect-free fabric images do not belong to $dataset0$. According to the style labels extracted by the pretrained VGGNet, in the first stage, the defect patches are generated and then pasted into the input defect-free fabric image. For each set in Fig. 6, the middle image represents the synthesized image after stage 1. Then, we treat these images as input and feed them into the well-trained stage 2 network. Here, the stage 2 network aims to fuse the pasted defects into the fabric background and generate more realistic samples. In Fig. 6, the last image of each set is the final synthesized defective fabric sample.

As shown in Fig. 6, after stage 1, our generated module can already generate reasonable defects based on the given fabric's style. However, since only the defects are synthesized, the synthesized defects are not perfectly fused into the background. Moreover, for some spot or block defects, such as oil stain and knot defects, the synthesized defects appear unrealistic. Therefore, we introduce stage 2 to refine those images synthesized in stage 1. After stage 2, the generated defects can naturally fuse into the fabric background, which delivers realistic defective samples to further fine-tune the detection model.

D. Detection Results

In this section, we report the detailed testing results of the fine-tuned model under diverse settings. As mentioned above, we prepared one simple texture dataset and one complex texture dataset and then conducted experiments on both. According to our categorization, there are four defects in simple-textured fabric (color spot, oil stain, knot, and broken end) and four defects in complex-textured fabric (color spot, broken end, broken yarn, and white strip).

1) Detection Results on the Simple Texture Dataset: Table I lists the number of defective images and the number of defects of each type in our training and test datasets. As a single image can contain multiple defects, the number of defects is greater than the number of images. On average, knots are present more often in one image than the other defect types, while the oil stain and broken end defects mostly occur in only a single image. The test dataset shares a similar ratio between the number of images and number of defects to the training dataset for each defect type. The average size of the images is 5300$\times$3000 pixels both during the training and test phases.

For comparisons with other methods, we also implement a fast Fourier transform (FFT)-based method [7] and train two fabric defect detection models (NCSR [9] and FCSDA [11]) with the same dataset. The results of the compared methods and our detection framework on the simple texture dataset are shown in Table II. Our framework achieves very competitive performance and outperforms all the comparison methods by a relatively large margin for all four defect types.

TABLE I

NUMBER OF DEFECTIVE IMAGES AND DEFECTS OF EACH TYPE IN THE SIMPLE TEXTURE DATASET IN THE TRAINING AND TEST PHASES

Defect Type	Training		Test	
	Number of defective images	Number of defects	Number of defective images	Number of defects
Color spot	1043	2145	260	554
Oil stain	1016	1276	254	325
Knot	1165	4221	291	1028
Broken end	1008	1083	252	265
Total	4232	8725	1057	2172

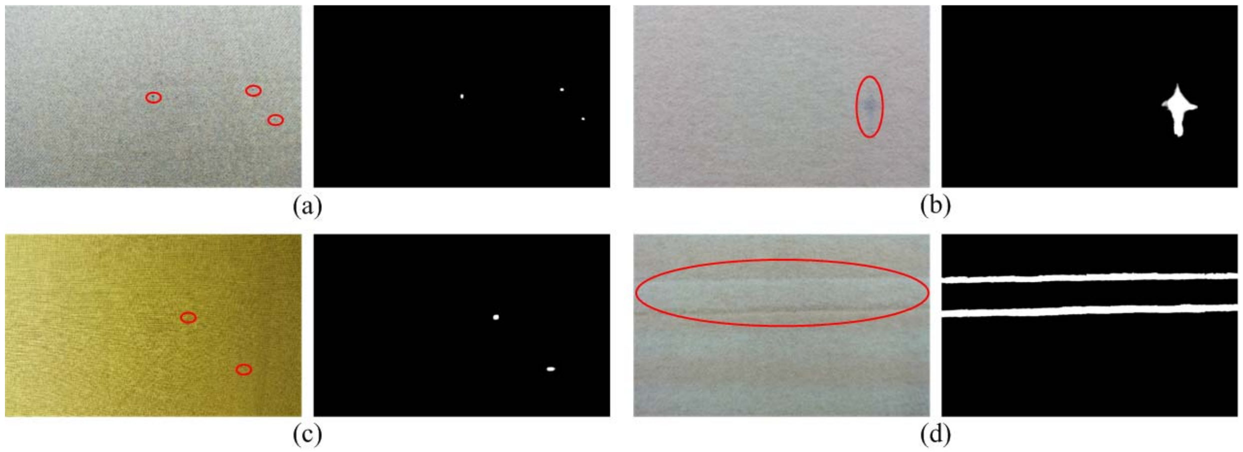

Fig. 7. Samples of detection results from the simple texture dataset. Defect types are (a) color spot, (b) oil stain, (c) knot, and (d) broken end.

We achieve a 96.2% F-measure on average, which is fairly high in the field of fabric defect detection. Over 96.8% of all the defects are recalled for each defect type, while the precision values are $\geq$ 98.5% of all the defects except for knots. The relatively inferior performance for detecting knots in simple textures is analyzed below. Some examples of our detection results on full-size test images are shown in Fig. 7.

2) *Detection Results on the Complex Texture Dataset:* Table III lists the number of defective images and defects of each defect type in complex textures in the training and test phases. The image size is the same as that for simple texture fabrics. We follow the same procedures as those used in the simple texture dataset.

The detection results of the compared methods and our framework are listed in Table IV. As shown in Table IV, all the compared methods failed to detect defects in complex-textured fabrics. The FFT-based method detects defects by monitoring changes in the frequency spectrum, but the frequency spectrum of normal textures in complex texture fabrics is also irregular as that of the defective textures. Moreover, both NCSR and FCSDA detect defects by calculating the residual between the reconstructed image and the defective image and cannot distinguish normal textures from the defective textures when reconstructing the defective image. The defects in the defective image are also reconstructed as the normal texture, leading to a failure in defect detection. Compared with Table II, complex texture defect detection had a 2.2% higher precision, while the recall fell by 5.7%. The reason for the decrease in recall is that a few more color spots were not detected by our method, as some colored spots are easily confused with the complex textures. The $F-measure$, which balances the $precision$ and $recall$, decreased slightly by 1.8%. Given that the detection was in complex textures, the performance can still be considered satisfactory for these four defect types. Some examples of our detection results on the full-sized testing images are shown in Fig. 8.

TABLE II

DETECTION RESULTS IN THE SIMPLE TEXTURE DATASET

Defect type	**Method**	**Precision**	**Recall**	**F-measure**
Color spot	FFT	84.2	73.3	78.4
	NCSR	90.3	76.9	83.0
	FCSDA	96.8	87.7	92.0
	Ours	**99.3**	**96.8**	**98.0**
Oil stain	FFT	88.2	78.5	83.1
	NCSR	92.2	87.7	89.9
	FCSDA	95.7	96.9	96.3
	Ours	**99.0**	**98.2**	**98.6**
Knot	FFT	59.1	86.7	70.3
	NCSR	70.6	90.3	79.2
	FCSDA	84	95.1	89.2
	Ours	**90.5**	**97.6**	**93.9**
Broken end	FFT	81.9	78.5	80.2
	NCSR	89.8	86.0	87.9
	FCSDA	97.7	97.4	97.5
	Ours	**98.5**	**98.9**	**98.7**
Total	FFT	69.8	80.9	75.0
	NCSR	79.5	86.0	82.6
	FCSDA	90.2	93.8	91.9
	Ours	**94.8**	**97.6**	**96.2**

TABLE III

NUMBER OF DEFECTIVE IMAGES AND DEFECTS FOR EACH DEFECT TYPE IN THE COMPLEX TEXTURE DATASET IN THE TRAINING AND TEST PHASES

Defect Type	Training		Test	
	Number of defective images	Number of defects	Number of defective images	Number of defects
Color spot	1325	3163	322	603
Broken end	1267	1738	316	343
Broken yarn	840	920	220	241
White strip	738	1157	215	376
Total	4170	6978	1073	1563

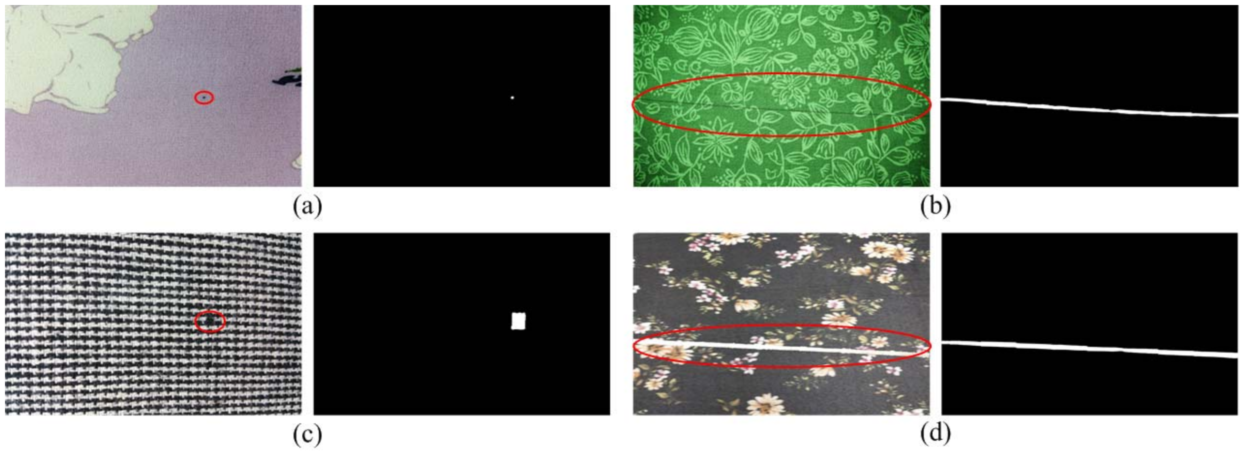

Fig. 8. Examples of the detection results from the complex texture dataset. The defect types are (a) color spot, (b) broken end, (c) broken yarn, and (d) white strip.

TABLE IV

DETECTION RESULTS FOR THE COMPLEX TEXTURE DATASET

Defect type	**Method**	**Precision**	**Recall**	**F-measure**
Color spot	FFT	-	-	Failed
	NCSR	-	-	Failed
	FCSDA	-	-	Failed
	Ours	98.5	88.7	93.4
Broken end	FFT	-	-	Failed
	NCSR	-	-	Failed
	FCSDA	-	-	Failed
	Ours	96.8	95.9	96.3
Broken yarn	FFT	-	-	Failed
	NCSR	-	-	Failed
	FCSDA	-	-	Failed
	Ours	93.2	90.5	91.8
White strip	FFT	-	-	Failed
	NCSR	-	-	Failed
	FCSDA	-	-	Failed
	Ours	97.3	95.2	96.2
Total	FFT	-	-	Failed
	NCSR	-	-	Failed
	FCSDA	-	-	Failed
	Ours	97.0	91.9	94.4

E. Analysis

1) Performance on Lower-Resolution Datasets: As mentioned above, the resolution of images used in the above experiments is relatively high, and the average size is 5300×3000 pixels in both the simple and complex texture datasets. We also evaluate the effectiveness of our framework on lower-resolution datasets. Since collecting and labeling defective fabric images are time-consuming and laborious tasks, we obtain lower-resolution datasets by directly scaling the above datasets and their corresponding annotations with scaling ratios of 0.25 and 0.5 and then conduct experiments following the same steps as those of the above experiments. Note that some small defects in the original image probably become so small in the lower-resolution image that they are not visually considered defects. Therefore, we ignore those defects less than 8 pixels in the lower-resolution images. The detection results on the simple and complex texture datasets with different resolutions are listed in Table V and Table VI, respectively. As shown in these tables, compared with the original datasets, the performance on lower-resolution datasets decreased slightly but still achieves very competitive results, which demonstrates that our proposed framework is also applicable to fabric defect detection in lower-resolution images.

2) Detecting Large-Scale Defects: From the visualization results, we note one particular drawback of our method, namely, that detecting large-scale defects is unsatisfactory. Fig. 9 visualizes the ground truth and our detection result for a large-scale white-strip defect in a complex texture. This white-strip defect takes up 38% of the entire image, and our method fails to detect such large-scale defects. The cause of this poor performance is that we crop 512×512 pixel patches from the sample images according to the defect position due to training memory limitations; however, if the defect is far larger than the crop size, then the detection is likely to fail.

Multiscale detection, which uses multiscale images as the input to be detected, can to some extent improve the detection

TABLE V

PERFORMANCE COMPARISON ON THE SIMPLE TEXTURE DATASETS WITH DIFFERENT RESOLUTIONS

Defect type	Scaling Ratio	Precision	Recall	F-measure
Color spot	0.25	98.1	95.3	96.7
	0.5	98.9	96.2	97.5
	1.0	**99.3**	**96.8**	**98.0**
Oil stain	0.25	97.8	96.3	97.1
	0.5	98.4	97.5	98.0
	1.0	**99.0**	**98.2**	**98.6**
Knot	0.25	90.0	96.7	93.2
	0.5	90.4	97.3	93.7
	1.0	**90.5**	**97.6**	**93.9**
Broken end	0.25	97.0	97.7	97.4
	0.5	98.1	98.5	98.3
	1.0	**98.5**	**98.9**	**98.7**
Total	0.25	93.9	96.4	95.2
	0.5	94.5	97.2	95.8
	1.0	**94.8**	**97.6**	**96.2**

TABLE VI

PERFORMANCE COMPARISON ON THE COMPLEX TEXTURE DATASETS WITH DIFFERENT RESOLUTIONS

Defect type	Scaling Ratio	Precision	Recall	F-measure
Color spot	0.25	98.1	87.2	92.4
	0.5	98.2	88.2	92.9
	1.0	**98.5**	**88.7**	**93.4**
Broken end	0.25	95.4	91.0	93.1
	0.5	96.4	94.2	95.3
	1.0	**96.8**	**95.9**	**96.3**
Broken yarn	0.25	90.8	89.6	90.2
	0.5	92.3	89.6	90.9
	1.0	**93.2**	**90.5**	**91.8**
White strip	0.25	96.2	94.1	95.2
	0.5	97.0	94.9	96.0
	1.0	**97.3**	**95.2**	**96.2**
Total	0.25	95.8	90.1	92.9
	0.5	96.5	91.3	93.9
	1.0	**97.0**	**91.9**	**94.4**

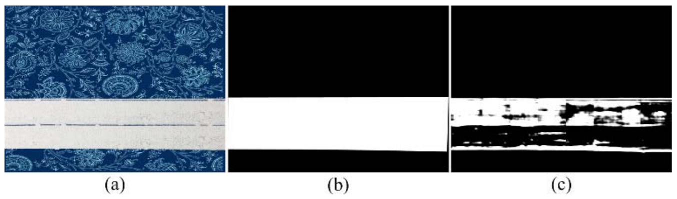

Fig. 9. Example detection result with a large-scale white-strip defect. (a) Testing image. (b) Ground truth. (c) Detection result.

performance for large-scale defects. Nevertheless, multiscale detection will introduce other problems. First, more computational capacity is required for detection at multiple scales, while in practice, fabric defect detection must be efficient. More seriously, for complex textures, multiscale detection is prone to classifying background texture as defects, which will increase the number of mistakes.

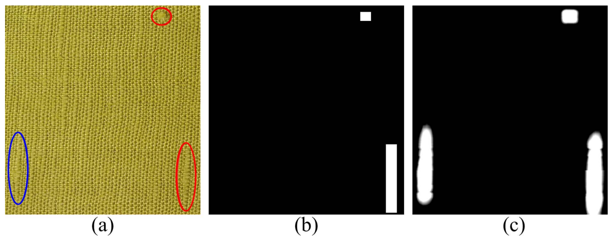

Fig. 10. Example detection result with a knot defect. (a) Testing image. (b) Ground truth. (c) Detection result.

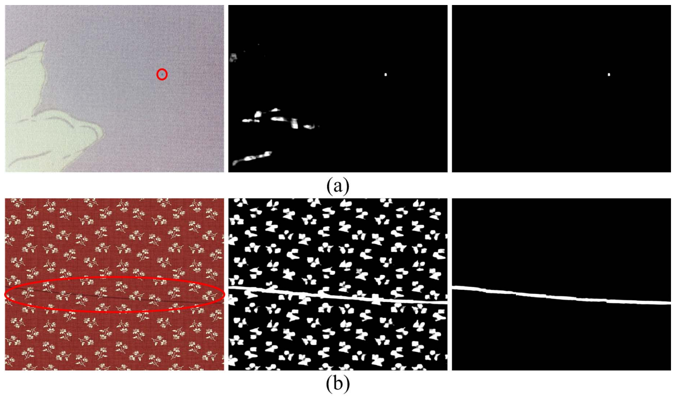

Fig. 11. Image in each column is a testing image; result before fine-tuning and result after fine-tuning, respectively. The defect types are (a) color spot and (b) broken end of complex-textured fabric.

3) High FN Result for Knots in Simple Textures: We also note that the FN rate is exceptionally high when detecting knots in simple texture samples. We manually checked all the visualized detection results for this defect and found that the model usually correctly recognized the knot, but the knot was not labeled as a defect in the corresponding ground truth image. Thus, we conclude that this failure is caused by imperfect annotation, even though our dataset was annotated by experienced staff from a textile mill. When annotating the ground truth for the knot defect, as these textures are usually coarse grained, even expert staff are not always accurate. Fig. 10 illustrates this situation; the bottom-left knot circled by a blue ellipse is not labeled as a defect in the ground truth image. However, this omitted knot is detected as a defect by our detection model, accounting for the high FN rate for knot defect detection. Therefore, we conclude that our method still detects knot defects in simple textures.

4) Improvement by Fine-Tuning: To demonstrate the improvement in detection due to the fine-tuning step, we also conduct experiments using the generated defective samples with new textures as the test dataset for the pretrained model. The model before fine-tuning is unfamiliar with some of the new texture features, so it is possible to mistakenly categorize complex background textures as defects. Fig. 11 shows two defective samples with new textures. Before fine-tuning, the model predicts many normal texture as defects, including some stripes in Fig. 11(a) and all the flowers in Fig. 11(b), which are recognized as background after fine-tuning. Given this clear evidence of improvement after fine-tuning, we deploy a fine-tuning module in the latter part of our framework, forcing the detection model to learn the features of the new texture fabric samples.

V. Conclusion

In this paper, we propose a GAN-based framework for fabric defect detection that can detect defects of various scales in both simple- and complex-textured fabrics. A pretrained semantic segmentation network is trained on an original dataset of defects to better address the fabric defect detection task. Using a multistage defect-generating GAN, we synthesize defective samples for the defect-free dataset with new textures and then used these synthesized samples for data augmentation and fine-tuning the pretrained model. Thus, this unified framework can detect defects in new texture fabrics. The experimental results show that our method functions well in detecting different defects in various background textures and also show that an efficient fine-tuning procedure allows our pretrained model to adapt to new texture fabrics.

Acknowledgment

The numerical calculations in this paper have been done on the supercomputing system in the Supercomputing Center of Wuhan University.

References

[1] H. Y. Ngan, G. K. Pang, and N. H. Yung, “Automated fabric defect detection—A review,” *Image Vis. Comput.*, vol. 29, no. 7, pp. 442–458, 2011.

[2] K. Hanbay, M. F. Talu, and Ö. F. Özgüven, “Fabric defect detection systems and methods—A systematic literature review,” *Optik*, vol. 127, no. 24, pp. 11960–11973, 2016.

[3] F. S. Cohen, Z. Fan, and S. Attali, “Automated inspection of textile fabrics using textural models,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 8, pp. 803–808, Aug. 1991.

[4] A. Latif-Amet, A. Ertüzün, and A. Erçil, “An efficient method for texture defect detection: Sub-band domain co-occurrence matrices,” *Image Vis. Comput.*, vol. 18, nos. 6–7, pp. 543–553, 2000.

[5] D. Chetverikov and A. Hanbury, “Finding defects in texture using regularity and local orientation,” *Pattern Recognit.*, vol. 35, no. 10, pp. 2165–2180, 2002.

[6] A. Abouelela, H. M. Abbas, H. Eldeeb, A. A. Wahdan, and S. M. Nassar, “Automated vision system for localizing structural defects in textile fabrics,” *Pattern Recognit. Lett.*, vol. 26, no. 10, pp. 1435–1443, 2005.

[7] C.-H. Chan and G. K. H. Pang, “Fabric defect detection by Fourier analysis,” *IEEE Trans. Ind. Appl.*, vol. 36, no. 5, pp. 1267–1276, Sep./Oct. 2000.

[8] L. Tong, W. K. Wong, and C. K. Kwong, “Differential evolution-based optimal Gabor filter model for fabric inspection,” *Neurocomputing*, vol. 173, pp. 1386–1401, Jan. 2016.

[9] L. Tong, W. K. Wong, and C. K. Kwong, “Fabric defect detection for apparel industry: A nonlocal sparse representation approach,” *IEEE Access*, vol. 5, pp. 5947–5964, 2017.

[10] S. Mei, Y. Wang, and G. Wen, “Automatic fabric defect detection with a multi-scale convolutional denoising autoencoder network model,” *Sensors*, vol. 18, no. 4, p. 1064 Apr. 2018.

[11] Y. Li, W. Zhao, and J. Pan, “Deformable patterned fabric defect detection with Fisher criterion-based deep learning,” *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1256–1264, Apr. 2017.

[12] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.

[13] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[14] W. Zhang, X. Wang, W. You, J. Chen, P. Dai, and P. Zhang, “RESLS: Region and edge synergetic level set framework for image segmentation,” *IEEE Trans. Image Process.*, vol. 29, pp. 57–71, 2019.

[15] J. Lafferty, A. Mccallum, and F. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *Proc. ICML*, 2002.

[16] C. Liang-Chieh, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, “Semantic image segmentation with deep convolutional nets and fully connected crfs,” in *Proc. Int. Conf. Learn. Represent.*, 2015.

[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2017.

[18] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” 2017, *arXiv:1706.05587*. [Online]. Available: https://arxiv.org/abs/1706.05587

[19] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 801–818.

[20] A. Raj, D. Maturana, and S. Scherer, “Multi-scale convolutional architecture for semantic segmentation,” Robot. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-RITR-15-21, 2015.

[21] G. Lin, A. Milan, C. Shen, and I. Reid, “RefineNet: Multi-path refinement networks for high-resolution semantic segmentation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1925–1934.

[22] Z. Zhang, X. Zhang, C. Peng, X. Xue, and J. Sun, “Exfuse: Enhancing feature fusion for semantic segmentation,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 269–284.

[23] I. Goodfellow *et al.*, “Generative adversarial nets,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[24] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1125–1134.

[25] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, “High-resolution image synthesis and semantic manipulation with conditional GANs,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8798–8807.

[26] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2223–2232.

[27] Y. Zhou, K. Gu, and T. Huang, “Unsupervised representation adversarial learning network: From reconstruction to generation,” 2018, *arXiv:1804.07353*. [Online]. Available: https://arxiv.org/abs/1804.07353

[28] M. Li, H. Huang, L. Ma, W. Liu, T. Zhang, and Y. Jiang, “Unsupervised image-to-image translation with stacked cycle-consistent adversarial networks,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 184–199.

[29] C. Ledig *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4681–4690.

[30] H. Zhang, V. Sindagi, and V. M. Patel, “Image de-raining using a conditional generative adversarial network,” 2017, *arXiv:1701.05957*. [Online]. Available: https://arxiv.org/abs/1701.05957

[31] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, “Generative visual manipulation on the natural image manifold,” in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 597–613.

[32] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “Neural photo editing with introspective adversarial networks,” 2016, *arXiv:1609.07093*. [Online]. Available: https://arxiv.org/abs/1609.07093

[33] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, “Context encoders: Feature learning by inpainting,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 2536–2544.

[34] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 4401–4410.

[35] Q. Mao, H.-Y. Lee, H.-Y. Tseng, S. Ma, and M.-H. Yang, “Mode seeking generative adversarial networks for diverse image synthesis,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 1429–1437.

[36] P. Wang and X. Bai, “Thermal infrared pedestrian segmentation based on conditional GAN,” *IEEE Trans. Image Process.*, vol. 28, no. 12, pp. 6007–6021, Dec. 2019.

[37] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.
[39] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 21–37.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[41] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.
[42] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*. [Online]. Available: https://arxiv.org/abs/1411.1784
[43] T. Miyato and M. Koyama, "cGANs with projection discriminator," 2018, *arXiv:1802.05637*. [Online]. Available: https://arxiv.org/abs/1802.05637
[44] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 694–711.

Juhua Liu received the M.S. and Ph.D. degrees in graphic communication from the School of Printing and Packaging, Wuhan University, Wuhan, China, in 2007 and 2014, respectively. He is currently an Associate Professor with the School of Printing and Packaging, Wuhan University. His research interests mainly include image processing, computer vision, and machine learning.

Chaoyue Wang received the bachelor's degree from Tianjin University, China, and the Ph.D. degree from the University of Technology Sydney, Australia. He is currently a Postdoctoral Researcher in machine learning and computer vision with the School of Computer Science, The University of Sydney. His research interests mainly include machine learning, deep learning, and generative models. He received the Distinguished Student Paper Award in the 2017 International Joint Conference on Artificial Intelligence (IJCAI).

Hai Su received the B.S. and Ph.D. degrees in graphic communication from the School of Printing and Packing, Wuhan University, Wuhan, China, in 2010 and 2015, respectively. He is currently a Lecturer with the School of Software, South China Normal University, Guangzhou, China. His research interests mainly include pattern recognition and image processing.

Bo Du (M'10–SM'15) received the Ph.D. degree in photogrammetry and remote sensing from the State Key Lab of Information Engineering in Surveying, Mapping and Remote sensing, Wuhan University, Wuhan, China, in 2010. He is currently a Professor with National Engineering Research Center for Multimedia Software, Institute of Artificial Intelligence, School of Computer Science, Wuhan University, Wuhan. He has more than 60 research papers published in the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEM (TNNLS), the IEEE TRANSACTIONS ON IMAGE PROCESSING (TIP), the IEEE TRANSACTIONS ON MULTIMEDIA (TMM), the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING (TGRS), the IEEE JOURNAL OF SELECTED TOPICS IN EARTH OBSERVATIONS AND APPLIED REMOTE SENSING (JSTARS), and the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS (GRSL); moreover, thirteen of them are ESI hot papers or highly cited papers. His major research interests include pattern recognition, hyperspectral image processing, machine learning, and signal processing. He received the Highly Cited Researcher 2019 Award from the Web of Science Group, the Distinguished Paper Award from IJCAI 2018, the Best Paper Award of the IEEE Whispers 2018, and the Champion Award of the IEEE Data Fusion Contest 2018. He received the Best Reviewer Awards from the IEEE GRSS for his service to IEEE the JOURNAL OF SELECTED TOPICS IN EARTH OBSERVATIONS AND APPLIED REMOTE SENSING (JSTARS) in 2011, and the ACM Rising Star Awards for his academic progress in 2015. He serves as an Associate Editor for the *Pattern Recognition and Neurocomputing*, a Senior PC for IJCAI/AAAI/KDD, and as the Area Chair for ICPR and IJCNN. He was the Session Chair for both the International Geoscience and Remote Sensing Symposium (IGARSS) 2018/2016 and the 4th IEEE GRSS Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS). He also serves as a Reviewer for 20 Science Citation Index (SCI) magazines, including the IEEE TGRS, TIP, JSTARS, and GRSL.

Dacheng Tao (F'15) is currently a Professor of computer science and an ARC Laureate Fellow with the School of Computer Science, Faculty of Engineering and Information Technologies, The University of Sydney, where he is also the Inaugural Director of the UBTECH Sydney Artificial Intelligence Centre. He mainly applies statistics and mathematics to artificial intelligence and data science. His research interests spread across computer vision, data science, image processing, machine learning, and video surveillance. His research results have been expounded in one monograph and over 500 publications in prestigious journals and at prominent conferences, such as the IEEE T-PAMI, T-NNLS, T-IP, JMLR, IJCV, NIPS, ICML, CVPR, ICCV, ECCV, ICDM, and ACM SIGKDD, with several Best Paper Awards, such as the Best Theory/Algorithm Paper Runner Up Award in the IEEE ICDM07, the Best Student Paper Award in the IEEE ICDM13, the Distinguished Student Paper Award in the 2017 IJCAI, the 2014 ICDM 10-Year Highest-Impact Paper Award, and the 2017 IEEE Signal Processing Society Best Paper Award. He was a recipient of the 2015 Australian Scopus-Eureka Prize, the 2015 ACS Gold Disruptor Award, and the 2015 UTS Vice-Chancellors Medal for Exceptional Research. He is a fellow of the AAAS, OSA, IAPR, and SPIE.