MÁSTER UNIVERSITARIO
EN VISIÓN ARTIFICIAL

TRABAJO FIN DE MÁSTER

Detection of minimum repeatable tile in fabric textures using CNN activations.

Autor: Vicente Gilabert Maño

Tutora: Elena Garcés García

Curso académico 2022/2023

Acknowledgements

I would like to take this opportunity to express my sincere appreciation to the individuals and organizations who have played a crucial role in my journey within the field of engineering, computer vision and artificial intelligence. Their unwavering support and guidance have been instrumental in shaping my professional growth and achievements.

First and foremost, I would like to extend my heartfelt gratitude to my family. Their constant encouragement, belief in my abilities, and unconditional support have been the driving force behind my success. Their unwavering commitment to my aspirations has provided me with the foundation and strength to pursue my career in this challenging field. I am forever grateful for their love and encouragement.

Furthermore, I would like to express my gratitude to Elena Garcés and the team at SEDDI, the company I am currently working for. Their support, trust, and belief in my abilities have provided me with an exceptional platform to apply my skills and contribute to groundbreaking projects. The collaborative environment at SEDDI has fostered my growth as a professional, allowing me to continuously learn, innovate, and push the boundaries of what is possible in the field of engineering and artificial intelligence. I am grateful for the opportunities they have provided me and the invaluable experiences I have gained through my association with the company.

Last but not least, I would also like to express my deep appreciation to my professors. Their wealth of knowledge, passion for teaching, and mentor-ship have been invaluable throughout my academic journey.

I am honored to have had the privilege of learning from and working with such exceptional individuals and organizations. Their support has not only shaped my career but has also inspired me to continually strive for excellence.

Resumen

La industria de la moda está adoptando la digitalización para poder abordar su ineficiencia, el gran desperdicio textil y los daños ambientales. Las herramientas digitales como el modelado 3D, la realidad virtual y la inteligencia artificial permiten a los diseñadores y fabricantes crear y visualizar diseños con precisión y rapidez, reduciendo tiempo y costos. El renderizado de texturas y la digitalización de tejidos son cada vez más importantes en la moda para realizar simulaciones y renderizados de gran realismo que se utilizan en la prueba virtual de prendas y la visualización de productos textiles.

Sin embargo, un problema abierto en visión por computador es la detección del patrón mínimo repetible de un tejido, que se refiere a la unidad más pequeña de un patrón textil que puede repetirse sin problemas para crear un patrón mayor. El mapeado de texturas es un aspecto crucial del renderizado de tejidos y requiere texturas que puedan repetirse y que estén optimizadas (bajo consumo de memoria) para un renderizado eficaz en tiempo real. Identificar el patron mínimo repetible es fundamental para crear mapas de texturas realistas y de alta calidad para aplicaciones como videojuegos y arte digital.

Existen varios enfoques tradicionales y modernos para detectar la repetición de patrones. En este trabajo nos centramos en un enfoque moderno que utiliza una red neuronal convolucional para extraer el mapa de activaciones de las imágenes téxtiles. El objetivo de nuestro trabajo es investigar y desarrollar Auto-tiling, es decir un proceso automático para la detección del patrón mínimo repetible. Esto implica un proceso de tres etapas (autoalineación, detección del patrón mínimo repetible y *stitching*). Se ha creado y desplegado una versión demo en línea para que los usuarios la prueben. Este trabajo se ha realizado con la compañia SEDDI, ya que se pretende integrar este producto en SEDDI Textura.

Summary

The fashion industry is undergoing digitalization to address its inefficiency, high textile waste, and environmental damage. Digital tools such as 3D modeling software, virtual reality, and artificial intelligence enable designers and manufacturers to create and visualize designs rapidly and precisely, reducing time and expenses. Texture rendering and digitizing fabrics are increasingly significant in the fashion industry, allowing for highly realistic simulations and renderings that can be used for various applications such as virtual clothing try-on and textile product visualization.

However, an open problem in computer vision is detecting the minimum repeatable tile (MRT) of a fabric, which refers to the smallest unit of a textile pattern that can be seamlessly repeated to create a larger pattern. This is essential for creating accurate digital representations of textile patterns, allowing the pattern to be scaled up or down without distorting its overall appearance. Texture mapping is crucial in fabric rendering and requires tileable and memory-optimized textures for efficient real-time rendering. Identifying the minimum repeatable tile is critical to creating realistic and high-quality texture maps for various applications, including video games and digital art.

Various traditional and modern approaches exist for detecting pattern repetition. This work focuses on a modern approach that uses a CNN to extract the activation maps of texture images. The aim of our work is to research and develop auto-tiling, i.e. an automatic process for the detection of the minimum repeatable pattern. This involves a three-stage process (auto-alignment, detection of the minimum repeatable tile and stitching), and an online demo has been created and deployed for users to test it. This work has been done with the company SEDDI, as it is intended to integrate this product into SEDDI Textura.

Contents

List of Figures **11**

List of Tables **15**

1 Introduction **19**

1.1 Problem and motivation 20

1.2 Context of SEDDI 21

2 Background and related work **23**

2.1 Convolutional Neural Network (CNN) 23

2.1.1 AlexNet 26

2.1.2 Feature Visualization 27

2.2 Multi-Scale Structural Similarity Index 28

2.3 Methods to estimate the minimum repeatable tile 30

2.3.1 Traditional approaches 30

2.3.2 Modern approaches 31

3 Proposed method **35**

3.1 Image alignment 35

3.2 Minimum repeatable tile 38

3.2.1 Overview 38

3.2.2 Preprocess image 39

3.2.3 Filter extraction from AlexNet 40

3.2.4 Creation of Distance map 41

3.2.5 Selection of best size pattern 42

3.3 Stitching 44

3.4 Implementation 47

3.4.1 Programming language 47

3.4.2 Packages 48

4 Experiments and results **51**

4.1 Dataset 51

4.2 Execution time 53

4.3 Evaluation 57

4.4 Correct results 59

4.5 Incorrect results 65

5 Auto-tiling demo **71**

5.1 Manual-tiling 72

5.2 Auto-tiling 75

5.3 Video examples 76

6 Conclusions and future lines **77**

Bibliography **79**

List of Figures

1.1 Images a) and b) are from our algorithm that identified the minimum pattern for two fabrics, shown in red in the inset, and we used it to create a tiled image. Images c) and d) are non-tileable images. 21

2.1 Convolution process: a matrix operation is performed to obtain a new value for the central pixel. It then shifts to the right to repeat the process. . . . 24

2.2 Representation of the architecture of a CNN for object detection. 24

2.3 Representation of the AlexNet architecture. 26

2.4 Features learned by a convolutional neural network (Inception V1) trained on the ImageNet data. The features range from simple features in the lower convolutional layers (left) to more abstract features in the higher convolutional layers (right). Figure from Olah, et al. [2] 28

2.5 The MS-SSIM results for different images are shown from left to right in the following order: original image, blurred image (kernel=5), blurred image (kernel=21), and ImageNET image. It can be observed that the MS-SSIM scores are decreasing from left to right, indicating that the similarity between the original image and the blurred images is decreasing as the amount of blurring increases. 29

2.6 Examples of three fabrics with the minimum pattern found by their algorithm (inset, in green), and a tiled image created using minimum pattern found. Figure from Rodriguez-Pardo et al. [18] 32

3.1 Complete pipeline of auto-tiling algorithm. 35

3.2 Top figure from [20]. (a) A directional texture rotated at 45°, (b) variance of projections at different angles. Both local maxima of the variance of the projections are correct. Bottom figure shows the angles direction (θ) used in radon transform algorithm. 37

3.3 Overview of complete pipeline of MRT algorithm and stitching step. In tiled image result (gray), is shown the original image containing the minimum repeatable tile marked by a red box, and the final seamless tile after apply the stitching correction. 39
3.4 The images displayed from left to right include the original image and randomly selected image from each convolutional layer of the network starting from layer 1 to layer 5 in increasing depth. 40
3.5 The images displayed from left to right include the original image, selected image from convolutional layer, the Non-Maximum Suppression (NMS) algorithm applied to selected image and distance map. 41
3.6 Displayed from left to right are the following images: the original image, the distance map, the non-maximum suppression of the distance map, and the final distance map with the detected peaks. 42
3.7 The images are presented from left to right as follows: the original image, the distance map with detected peaks, and for each peak, the resulting tiled image. 43
3.8 The images displayed from left to right: the original image containing the minimum repeatable tile marked by a red box, the minimum repeatable tile is cropped with an overlap, and the area of interest is highlighted with a red box, the stitched image composed of a 2x2 grid with seamless tile highlighted in green, and the final seamless tile. 45
3.9 The sequence of images from left to right displays the following: a stitched image consisting of a 2x2 grid, with marked in orange the overlapped section, followed by the absolute difference of the overlap section, a blurred version of the previous image, the detected seam mask, and the dilated seam mask. 46
3.10 Tiled image with no correction and with stitching correction. 46

4.1 Radom images selected from our fabrics dataset. 52
4.2 Radom images were deleted from our fabric dataset. 53
4.3 Execution time in seconds of all steps of auto-tiling algorithm. We use CPU with ONNX in MRT step. The number in the box is the mean execution time of all dataset. 54

4.4 Execution time in seconds of all steps of MRT algorithm. We use CPU with ONNX in MRT step. The number in the box is the mean execution time of all dataset. 54
4.5 Execution time in seconds of CPU and GPU inference using torch, and CPU and GPU using ONNXruntime. The number in the box is the mean execution time of all dataset. 56
4.6 MS-SSIM=0.604. 59
4.7 MS-SSIM=0.959. 59
4.8 MS-SSIM=0.771. 60
4.9 MS-SSIM=0.924. 60
4.10 MS-SSIM=0.939. 61
4.11 MS-SSIM=0.939. 61
4.12 MS-SSIM=0.944. 62
4.13 MS-SSIM=0.857. 62
4.14 MS-SSIM=0.885. 63
4.15 MS-SSIM=0.889. 63
4.16 MS-SSIM=0.908. 64
4.17 MS-SSIM=0.870. Incorrect result due to MRT failure. 65
4.18 MS-SSIM=0.878. Incorrect result because of MRT failure. 65
4.19 MS-SSIM=0.676. Incorrect result due to MRT failure. 66
4.20 MS-SSIM=0.531. Incorrect result due to MRT failure 66
4.21 MS-SSIM=0.852. Incorrect result due to misalignment of the fabric. 67
4.22 MS-SSIM=0.649. Incorrect result due to misalignment of the fabric. 67
4.23 MS-SSIM=0.903. Incorrect result due to stitching error. 68
4.24 MS-SSIM=0.903. Incorrect result due to stitching error. 68
4.25 MS-SSIM=0.485. Incorrect result due to auto-alignment failure. 69
4.26 MS-SSIM=0.807. Incorrect result due to misalignment of the fabric. 69
4.27 MS-SSIM=0.682. Incorrect result due to misalignment of the fabric. 70
4.28 MS-SSIM=0.797. Incorrect result due to misalignment of the fabric. 70

5.1 Homepage of auto-tiling application. An image has been uploaded. 72
5.2 Homepage of auto-tiling application. An image has been uploaded. 72
5.3 Step 2: Manual and automatic angle rotation correction. 73
5.4 Step 3: Manual and automatic minimum repeatable tile detection. 74

5.5 Result of manual tiling. 75
5.6 Execution of automatic tiling. 75
5.7 The left image displays the detected best repeatable pattern while the right image shows the resulting pattern after tiling. 76

List of Tables

4.1 Results of manual revision of auto-tiling algorithm. 57
4.2 Results of MS-SSIM and verification with manual revision. 58

Acronyms

CAD Computer-Aided Design. 19, 22

CNN Convolutional Neural Network. 9, 21, 23, 24, 26, 27, 32, 38, 40, 55

CPU Central Processing Unit. 12, 13, 48, 53–56

DISTS Deep Image Structure and Texture Similarity. 29

DMF distance matching functions. 30, 31

GAN Generative Adversarial Network. 33

GLCM Gray-level co-occurrence matrix. 30

GPU Graphic Processing Unit. 13, 26, 47, 48, 53, 55, 56

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 26

LPIPS Learned Perceptual Image Patch Similarity. 29

MLP Multilayer Perceptron. 23

MRP minimum repeatable pattern. 20

MRT minimum repeatable tile. 7, 9, 12, 13, 20, 30, 35, 38, 39, 53–55, 57, 58, 65, 77

MS-SSIM Multi-Scale Structural Similarity Index. 11, 15, 28–30, 38, 43, 57, 58, 77

NMS Non-Maximum Suppression. 12, 41, 42

ONNX Open Neural Network Exchange. 12, 13, 48, 53–56

OpenCV Open Source Computer Vision Library. 48

PSNR Peak Signal-to-Noise Ratio. 29

RMSprop Root Mean Square Propagation. 25

SDH sum and difference histogram. 31

SGD Stochastic Gradient Descent. 25

SIFT Scale-invariant feature transforms. 31

SSIM Structural Similarity Index Measure. 28, 29

STD standard deviation. 41

SURF Speeded-Up Robust Features. 31

Chapter 1

Introduction

Digitalizing the fashion industry is becoming increasingly important for several reasons. The traditional fashion industry is known for being inefficient, wasteful, and environmentally damaging. Digitalization offers a way to address these issues by increasing efficiency, reducing waste, and promoting sustainability.

The digitization of fashion fabrics provides many advantages for designers and manufacturers. Using digital tools, including 3D modeling software, virtual reality, and artificial intelligence, designs can be created and visualized rapidly and precisely. As a result, time and expenses associated with conventional design processes such as producing physical prototypes and samples can be reduced. Furthermore, digitization enables designers to have more flexibility and creativity in the design process, as they can evaluate various fabrics, colors, and styles without requiring physical materials.

In recent years, computer vision and computer graphics have played an increasingly significant role in the fashion industry, particularly in the area of textile texture rendering and digitizing fashion fabrics. Rendering textile textures and digital fashion fabrics is an important aspect of the field of digital textile design. With the advent of Computer-Aided Design (CAD) and advanced imaging technologies, it is now possible to create highly realistic textile simulations and renderings that can be used for a variety of applications, such as virtual clothing try-ons, textile product visualization, and designing clothes in a 3D environment.

The process of rendering textile textures involves creating a digital representation of the visual and tactile qualities of a particular fabric. This is achieved by capturing high-resolution images of the fabric and using computer algorithms to analyse and interpret the texture and colour data. Once the data has been processed, it can be used to create

digital models of the fabric that can be incorporated into various design applications.

Despite the significance and vast scale of the textile industry, numerous unresolved issues continue to be the subject of ongoing research. In this work, we will focus on the detection of the minimum repeatable tile of a fabric which refers to the smallest unit of a textile pattern that can be seamlessly repeated to create a larger pattern. This is essential for creating accurate digital representations of textile patterns, as it allows for the pattern to be scaled up or down without distorting its overall appearance.

1.1 Problem and motivation

Texture mapping is a crucial aspect of rendering, particularly in real-time rendering engines. Texture mapping involves mapping a 2D texture onto a 3D model in a way that makes the texture appear to be part of the model's surface. To be effective, texture mapping requires the texture to be tileable, meaning that it can be repeated seamlessly across the surface of the 3D model without producing visible artifacts at the boundaries. This is important because if the texture is non tileable, it can lead to visible seams or distortions in the final rendered image, as shown in images c) and d) of Figure 1.1. These artifacts can detract from the overall realism of the model. In addition to being tileable, textures must also be optimized for low memory consumption to ensure that they can be rendered quickly and efficiently in real-time rendering engines.

Detecting the minimum repeatable tile (MRT) of a texture, also called minimum repeatable pattern (MRP), is an open problem in computer vision, particularly in the context of texture analysis and synthesis. Textures are an essential aspect of many visual applications, including digital art, video games, and virtual reality. The MRT is the smallest section of a texture that can be repeated without creating visible seams or distortions, as shown in images a) and b) of Figure 1.1. By identifying it, it is possible to map the texture onto a 3D object accurately, producing a more realistic and visually pleasing result. Using a MRT ensures that the texture tiles without any visible seams or repetitions. This creates a more realistic appearance, as the fabric's texture appears continuous and uninterrupted. This process is essential for creating high-quality texture maps for video game assets, digital art, and other applications where realistic textures are critical.

In the state of the art, there are many ways to detect a pattern repetition using

traditional and modern approaches. We used a modern approach that use a Convolutional Neural Network (CNN) to extract the activation maps of texture images.

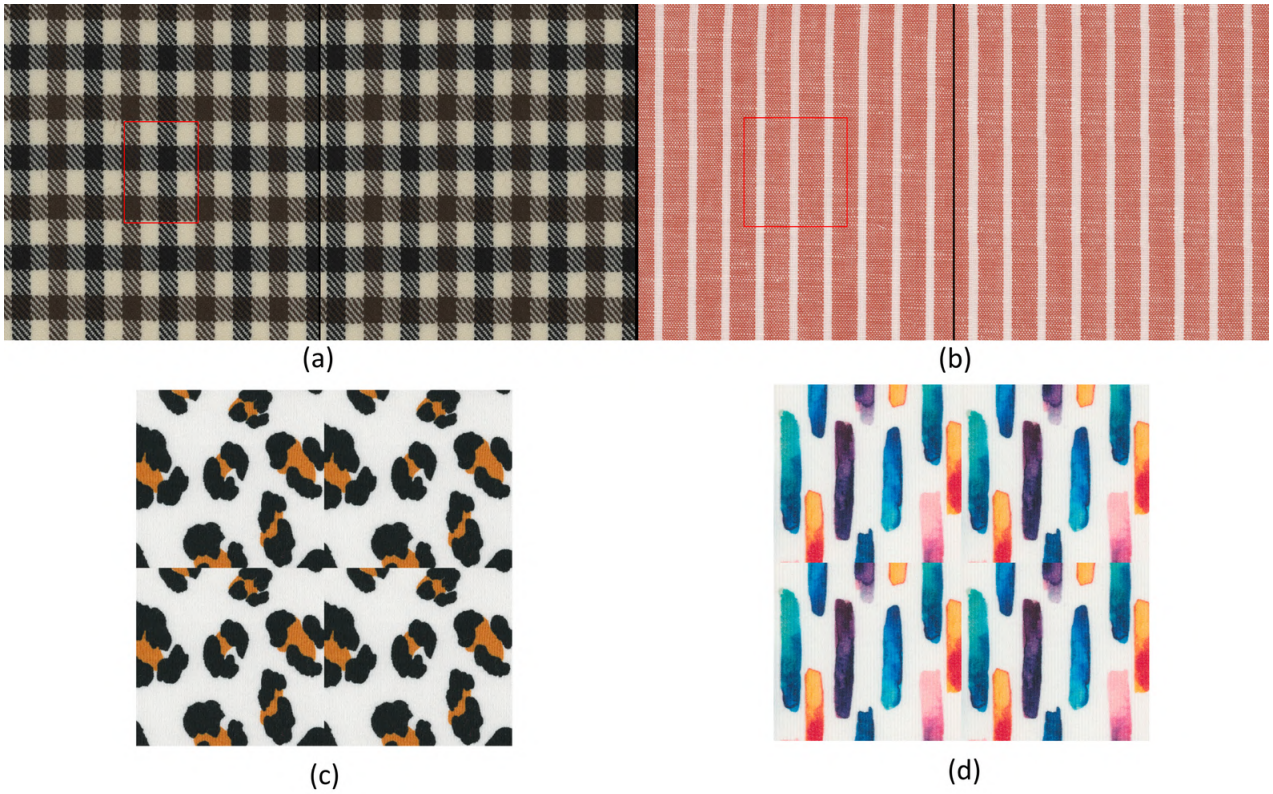


Figure 1.1: Images a) and b) are from our algorithm that identified the minimum pattern for two fabrics, shown in red in the inset, and we used it to create a tiled image. Images c) and d) are non-tileable images.

1.2 Context of SEDDI

The fashion industry has been undergoing a significant transformation, with the rise of "Fast Fashion" resulting in a highly negative environmental impact. Production waste, including fiber, yarn, and fabric waste, is a significant contributor to this waste, and to mitigate this problem, production rates need to be reduced and production needs to be more precise. According to [1] studies showed 15% of fabric used in garment manufacturing is wasted, and some estimates even place textile waste during garment manufacturing at 25–30%.

To reduce the fashion industry's impact on the environment, textile companies should prioritize reducing their environmental impact, and larger corporations are beginning to

make small changes in their policies to demonstrate their commitment to this change. A tool to generate reliable simulations of textiles, garments, and bodies is needed, which would enable companies to create new garments without the use of physical materials and aid users in finding the garment that best fits their size and figure. Overall, the fashion industry needs new, straightforward tools to reduce its environmental impact, even if only partially.

SEDDI's primary objective is to digitize the textile industry to minimize its environmental impact. By utilizing physical science and data, SEDDI seeks to simulate textiles, garments, and bodies with an unprecedented level of accuracy, thereby revolutionizing the fashion industry.

SEDDI Author is a CAD software for pattern design and grading, as well as automated cutting systems and fabric spreaders, which enable fashion companies to reduce waste and improve accuracy in garment production, while also saving time and costs associated with manual cutting. These eco-friendly solutions eliminate the need for physical samples, reducing natural resource consumption, and can increase efficiency and productivity in the fashion industry.

SEDDI Textura aims to simulate textiles, garments, and bodies with unmatched accuracy by applying physical science and data in a unique way. Their optics team is responsible for capturing and interpreting materials to ensure accurate simulation. I have been working with them to do this work.

SEDDI's solutions also help to improve accuracy and precision, reducing errors and waste in the production process. The products and services of SEDDI are a game-changer in the fashion industry, introducing new technologies that improve efficiency, accuracy, and sustainability, boosting productivity and output while saving costs.

Chapter 2

Background and related work

This chapter provides an overview of the theoretical background relevant to our work, as well as a review of related works in the field.

2.1 Convolutional Neural Network (CNN)

Neural networks are machine learning models inspired by biological networks of neurons, and they have the capability to solve a wide range of problems, including supervised and unsupervised learning tasks. In many domains, neural networks have emerged as state-of-the-art models. In supervised learning scenarios, neural networks are typically organized as a sequence of layers, where each layer receives input data, performs a mathematical operation on it, and passes the output to the next layer. This architecture is commonly seen in Multilayer Perceptron (MLP), but is limited in domains in which the data has patterns that depend on local structures.

Since its introduction by Yann LeCun in the early 1990s, CNN has shown excellent performance in image processing tasks, achieving impressive results in image recognition and deeply changing the state of the art in computer vision.

CNNs, a variation of MLPs, are well-suited for image processing due to their matrix-based operations. They shine in pattern recognition tasks, such as object, face, and scene recognition, by directly learning from image data and eliminating the need for manual feature extraction. While traditional techniques are still used in some cases, CNNs use convolutional layers as their key components to capture local patterns. These layers consist of square filters (Figure 2.1), applied through the convolution operation, sliding across the input image or previous layer's activation to generate feature maps.

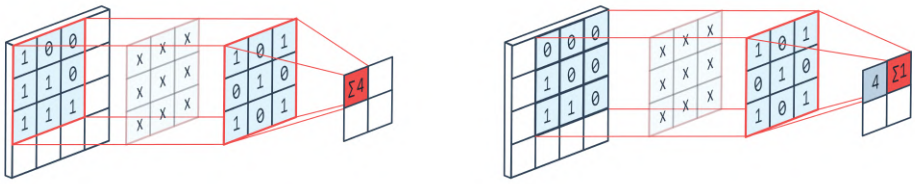


Figure 2.1: Convolution process: a matrix operation is performed to obtain a new value for the central pixel. It then shifts to the right to repeat the process.

Each feature map signifies the presence or absence of specific visual features, like edges, corners, or textures, within specific image regions. Training the network involves optimizing these features through gradient descent, improving classification accuracy. A notable advantage of CNNs lies in their hierarchical learning capability. By stacking multiple convolutional layers, the network learns features at different levels of abstraction. Lower layers focus on simpler features like edges, while deeper layers extract more intricate characteristics like textures, enabling effective representation of complex visual information.

In CNNs, the architecture consists of an input layer, an output layer, and hidden layers, with certain layers dedicated to performing convolutional operations. The convolutional operation involves the application of masks or filters, which significantly impact the results obtained, enabling tasks such as edge detection, noise removal, or profiling in image processing.

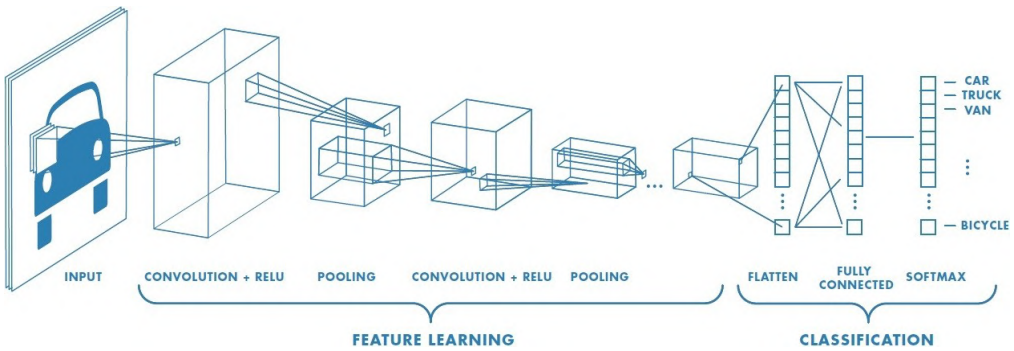


Figure 2.2: Representation of the architecture of a CNN for object detection.

CNN models usually include three different layers, as depicted in Figure 2.2:

- Convolutional layers: These layers apply masks or filters to the input image, allowing for feature extraction and pattern recognition.

- Pooling layers: These layers reduce the spatial size of the image by taking either the maximum or average value within each block, enabling downsampling and retaining important information.

- Fully connected layers: These layers are typically positioned at the end of the model for image classification tasks. However, they ignore spatial information because they connect all neurons from the previous layer to the next layer, treating the data in a flattened way.

Depending on the application, a model can be created from scratch (without prior training) or transfer learning can be used, which means using a model that has already been trained with a dataset that can be useful for your project. Another way of working is to use a model already designed by scientists and train it with your own data to detect what you are looking for.

When we talk about training, we mean to learn by adjusting the weights and biases of neurons. During training, the network is presented with a large set of input data, along with the desired output or label for each input. The network processes the input data through its layers of neurons and produces an output. The difference between the output produced by the network and the desired output is measured using a loss function, which quantifies the difference or error between the predicted and actual output. The goal of training is to minimize this error or loss function by adjusting the weights and biases of the neurons. This is achieved through an iterative process called backpropagation, which calculates the gradient of the loss function with respect to the network parameters and uses this information to update the weights and biases in the opposite direction of the gradient.

The optimization process involves the use of optimization algorithms such as Stochastic Gradient Descent (SGD), Adam, or Root Mean Square Propagation (RMSprop), which are used to update the weights and biases in the network at each iteration. These algorithms use the gradient of the loss function with respect to the network parameters to determine the direction and magnitude of the weight updates. The choice of optimization algorithm and the settings used for learning rate, batch size, and other hyperparameters can significantly impact the training process and the performance of the neural network.

2.1.1 AlexNet

AlexNet is a Convolutional Neural Network (CNN) that was developed by a team of researchers led by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. It was designed to compete in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which is an annual competition that evaluates algorithms for object detection and classification in large datasets of images.

AlexNet was a game-changer for several reasons. First, it was much deeper than previous CNNs, with eight layers of neurons. This allowed it to learn more complex features of images and improve its accuracy. Second, it used a technique called "dropout" to prevent overfitting, which is when the network becomes too specialized for the training data and performs poorly on new data. Dropout randomly drops out some of the neurons during training, forcing the network to learn more robust features that are less dependent on the specific training data. Third, AlexNet used a technique called "data augmentation" to artificially expand the size of the training dataset. This involves randomly cropping, flipping, and rotating the images to create new variations, which helps the network generalize better to new images.

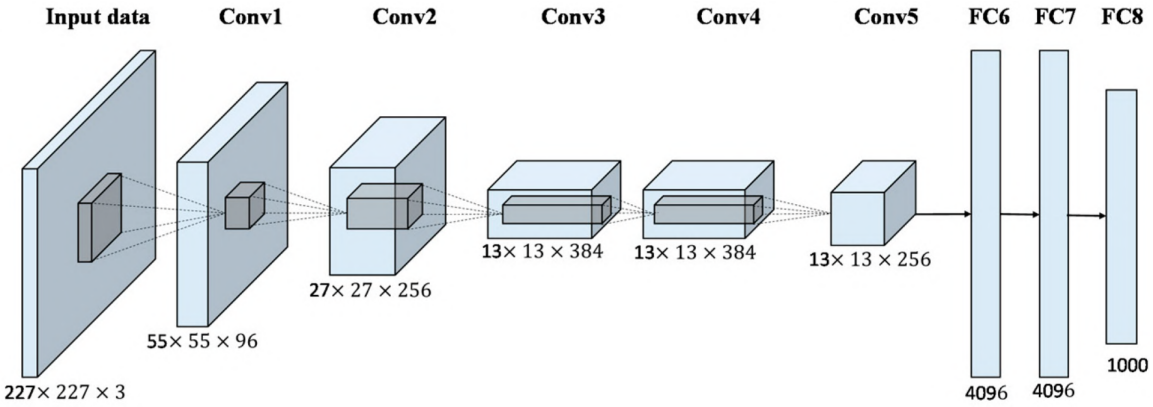


Figure 2.3: Representation of the AlexNet architecture.

When AlexNet was introduced, it achieved a top-5 error rate of 15.3% on the ILSVRC dataset, which was a significant improvement over the previous year's winning entry. It was the first architecture that used GPU to boost training performance. It also demonstrated the potential of deep learning and CNNs for computer vision tasks, and inspired a new wave of research in the field. Since then, CNNs have become the dominant approach for image classification, and deep learning has been applied to a wide range of other tasks in computer vision, natural language processing, and other domains.

AlexNet plays a crucial role in our work, serving as a powerful feature extractor by utilizing the activation maps of its convolutional layers. Through this approach, we can effectively analyze and extract features from input images, providing a deeper understanding of the unique characteristics and variations present in different fabric textures. Using these learned features, we improve the accuracy and effectiveness of our algorithm in identifying and extracting the minimum repeatable tile from complex fabric textures, ultimately improving the performance of our extraction process.

2.1.2 Feature Visualization

Feature visualization [2] is the process of generating and visualizing images that can activate specific features or neurons in a neural network. It is a technique used to gain insights into what a neural network has learned and how it represents different concepts in its internal layers. In a CNN, each layer learns to represent different features or patterns in the input data. For example, in an image classification network, the first layer might learn to recognize edges and corners, while later layers might learn to recognize more complex shapes and objects. By visualizing the patterns of activation in each layer, we can gain insights into what the network has learned and how it is making decisions.

Feature visualization is a powerful technique used to gain insights into neural networks by generating and visualizing images that activate specific features or neurons. It can be utilized in several ways, such as understanding the network's performance by identifying what features each layer is sensitive to, and troubleshooting the network by detecting issues in the activation patterns of each layer. Additionally, feature visualization can be applied creatively to generate artistic visualizations like DeepDream or style transfer, which can provide new insights and creative work.

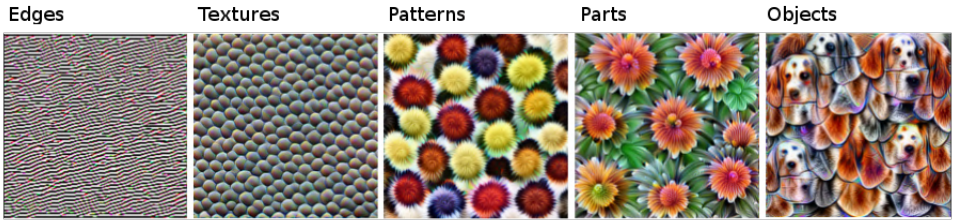


Figure 2.4: Features learned by a convolutional neural network (Inception V1) trained on the ImageNet data. The features range from simple features in the lower convolutional layers (left) to more abstract features in the higher convolutional layers (right). Figure from Olah, et al. [2]

Some of the techniques used in feature visualization include activation maximization, deconvolution, guided backpropagation, DeepDream, and style transfer. These techniques allow us to generate images that maximize the activation of specific neurons or layers or transfer the style of one image onto the content of another. By visualizing the patterns of activation in each layer, we can gain insights into what the network has learned and how it represents different concepts.

In our research, we employ an optimization process to extract the feature map from the convolutional layers of AlexNet. Consistent sizing of the activation maps is essential for effective analysis of the extracted features. By resizing the activation maps to a uniform size, we enable a comprehensive and accurate examination of these features.

2.2 Multi-Scale Structural Similarity Index

Multi-Scale Structural Similarity Index (MS-SSIM) is a popular algorithm for image quality assessment that measures the structural similarity between two images. MS-SSIM is an extension of the original Structural Similarity Index Measure (SSIM) algorithm [3] that operates at multiple scales, allowing it to capture more complex image structures.

The SSIM index is a method for measuring the similarity between two images based on luminance, contrast, and structural distortions. To evaluate the similarity, the MS-SSIM algorithm first decomposes both images into multiple scales with a Gaussian pyramid. Then, it calculates the SSIM index for each scale and combines them using a weighted average to obtain a final similarity score. The weighting assigned to each scale is inversely proportional to the scale, indicating that smaller image structures have a greater effect

on the final score.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.1}$$

where x and y are the two images being compared, μ_x and μ_y are the mean pixel intensities of the two images, σ_x^2 and σ_y^2 are the variances of pixel intensities of the two images, σ_{xy} is the covariance between the pixel intensities of the two images, and c_1 and c_2 are small constants added for numerical stability.

$$MS - SSIM(x, y) = \left[\prod_{i=1}^{N} SSIM_i\,(x, y)^{w_i}\right]^{\gamma} \tag{2.2}$$

where N is the number of scales; $SSIMi(x, y)$ is the SSIM index at scale i; wi is a weight coefficient that gives more importance to scales with more perceptual relevance; and γ is a parameter that controls the overall sensitivity of the algorithm.

Figure 2.5: The MS-SSIM results for different images are shown from left to right in the following order: original image, blurred image (kernel=5), blurred image (kernel=21), and ImageNET image. It can be observed that the MS-SSIM scores are decreasing from left to right, indicating that the similarity between the original image and the blurred images is decreasing as the amount of blurring increases.

One of the advantages of MS-SSIM is its ability to accurately capture the perceptual quality of images, particularly in scenarios where traditional metrics like PSNR and SSIM fail. MS-SSIM has been shown to correlate well with human judgments of image quality, making it a useful tool for evaluating the effectiveness of image processing algorithms. Another advantage is its computational efficiency. The algorithm only requires a small number of calculations compared to other modern perceptual metrics like Learned Perceptual Image Patch Similarity (LPIPS) [4] or Deep Image Structure and Texture Similarity (DISTS) [5], making it well-suited for real-time applications.

In our work, we use MS-SSIM to evaluate the similarity between the tiled images and the original image, taking into account both structural and perceptual information. By comparing the MS-SSIM scores, we can quantify how well the tiled images preserve the visual characteristics and details of the original image, ensuring that the resulting tiled images closely resemble the original texture with minimal distortions or artifacts.

2.3 Methods to estimate the minimum repeatable tile

Finding repeating patterns is a challenging task in computer vision, and it has been an active research area for many years. This problem arises in many applications, such as image retrieval, object recognition, and video analysis, among others. Repeating patterns can be defined as visual elements that occur multiple times within an image. These elements can be simple, such as textures or geometric shapes, or more complex, such as objects or scenes. Traditional approaches to textile pattern analysis rely on human expertise and require significant manual effort, which is extremely costly and time-consuming. To meet the needs of digital manufacturing in the textile industry, it's essential to find a way to accurately extract the minimum repeatable tile from the printed fabric automatically.

2.3.1 Traditional approaches

There is a large amount of work in the field of periodic pattern analysis. Generally, they can be mainly divided into the following 3 categories: (1) spectral-based methods, (2) statistical-based methods, (3) traditional learning-based methods. For example, early in 1980, Terzopoulus et al. [6] tried to analyze periodic textures structure by using the Gray-level co-occurrence matrix (GLCM). However, it was computationally intensive and had poor robustness [7]. The approach using auto-correlation function to calculate peaks was proposed by Lin et al. [8]. This method had a certain tolerance to local distortion, but it required smoothing the auto-correlation function, which led to a long calculation time.

Oh et al. [9] used the distance matching function to analyse periodic texture's structure, which has a small amount of calculation consumption. However, the distance

matching functions (DMF) method is very sensitive to noise such as geometric distortion. A method of using sum and difference histogram (SDH) for texture analysis was proposed by Unser et al. [10]. The advantage of this method was that it greatly reduced the calculation time and had a stable effect. Matsuyama et al. [11] used the Fourier transform to analyze the textures. The period of repeat pattern primitives could be determined by the pulse distribution of the Fourier spectrum. However, the peak of the Fourier spectrum was not prominent when the pattern contained fewer periodic primitives, which made the method based on the Fourier transform could only segment the periodic pattern primitives of the woven fabric.

Feature extraction methods aim to extract representative features from the input images, which can be used to describe the repeating patterns. These features can be based on local descriptors, such as Scale-invariant feature transforms (SIFT) or Speeded-Up Robust Features (SURF), or on global descriptors, such as bag-of-features or deep features. In terms of image descriptors used to represent image information such as shapes or colors, we find a vast number of methods [12, 13, 14], that rely on SIFT [15] as feature descriptors.

Kuo et al. [16] converted the printed fabric into a full-color image and employed the fuzzy mean clustering algorithm to segment the pattern to obtain the pattern of the printed fabric. Finally, the proposed approach using the Hough transform completed the segmentation of periodic pattern primitives. However, this method was only suitable for patterns with relatively clear color boundaries and had trouble effectively identifying patterns with rich colors and complex textures.

The above methods often require the manual design of feature extractors, which have poor adaptability and are only effective for some simple periodic pattern primitives. Therefore, it is hard to adapt to complex printed fabrics, which are unable to meet the needs of the textile industry.

2.3.2 Modern approaches

In recent years, the application of deep learning algorithms has become increasingly widespread, which has promoted the rapid development of computer vision, especially in image classification, object detection, and semantic segmentation. Thanks to the contribution of the academic community in sharing their work and pre-trained deep convolutional neural networks, new opportunities have been created in this field. These

approaches have the advantage of being able to learn discriminative features directly from the input data, without the need for manual feature engineering. CNNs are a type of neural network that has been shown to be highly effective at image recognition tasks and has been successfully applied to fabric pattern recognition. The AlexNet model, for instance, has been used to classify fabric patterns with high accuracy, outperforming traditional computer vision methods. However, few researchers apply it to the pattern primitive's extraction of printed fabrics.

In particular, Lettry et al. [17] present a method to find repetitive patterns in natural scenes by looking for spatial regularities in the activations of filters in a pre-trained neural network. However, the primitives of periodic patterns in natural scenes are relatively simple and are usually composed of regular geometric objects with clear boundaries. Their method is capable of finding the most probable size of the repeating pattern in the image, as well as fitting a grid that can segment those repeating patterns. A main disadvantage of this method is that the repeating pattern can only be of one size, and it works better when the grid in which the repeating pattern lies is aligned with the axis of the image and when the image is fronto-planar. Despite those disadvantages, we find their work to be a suitable starting point that inspired our algorithm for geometrically regular input textures.

As extensions of the previous article applied to fabrics, we have Rodriguez-Pardo et al. [18] and Xiang et al. [19]. First one, propose an end-to-end pipeline capable of finding the size of the minimal repeating pattern in single images using AlexNet as well as obtaining the single repetition that, when tiled, produces the most similar synthesis to the complete image.

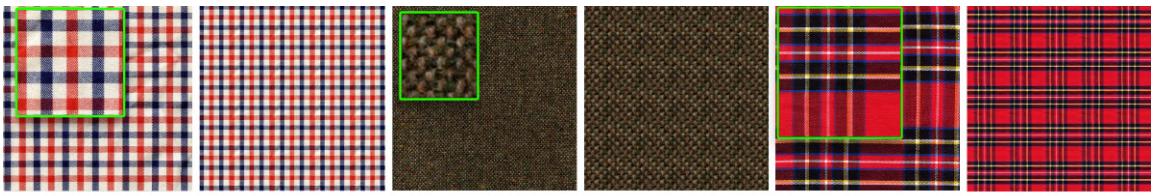


Figure 2.6: Examples of three fabrics with the minimum pattern found by their algorithm (inset, in green), and a tiled image created using minimum pattern found. Figure from Rodriguez-Pardo et al. [18]

Xiang et al. [19] propose the same pipeline of [17], but modify the last pooling layer by Spatial Pyramid Pooling helping the network learn multi-scale features and improving

the accuracy of the network classification of whether it contains periodic primitives. They also retrain AlexNet using fabric samples, which helps them achieve better results.

Another approach is to use Generative Adversarial Network (GAN) to generate synthetic fabric samples that can be used to learn about the repeating patterns. This has been used to analyze and synthesize various types of fabrics, including woven, knitted, and printed fabrics. GANs have also been used for textile defect detection by training on images of defective fabrics and generating synthetic images that can be used to train a classifier.

We have based our work on two articles in particular, which are [17] and [18].

Chapter 3

Proposed method

We propose an algorithm to get a minimum repeatable tile (MRT) using three stages:

- Auto-alignment: Detect angle rotation to correct image input.
- Detection of minimum repeatable tile (MRT): Detect best crop to tile image.
- Stitching: Correction of MRT crop to avoid seams (artifacts) of tiled image.

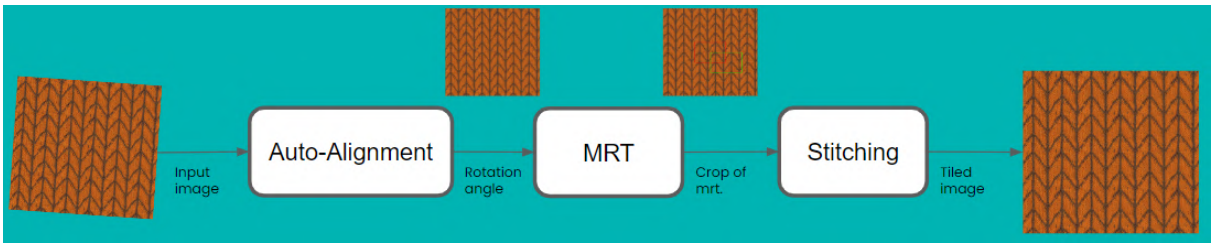


Figure 3.1: Complete pipeline of auto-tiling algorithm.

3.1 Image alignment

The initial crucial step is to align the input texture with the main axis. Without proper alignment, identifying recurring patterns becomes more challenging. Since the input image in this case is a scanned fabric, it is necessary to ensure that it is aligned as accurately as possible. However, it can be difficult to achieve perfect alignment as the fabrics are placed manually during the collection process.

Image alignment using Radon transform can also be used to align a single image with itself. This technique is known as self-alignment or self-registration, and it is often used

to correct for misalignments or distortions in the image due to factors such as camera motion or lens distortion. The Radon transform is a mathematical technique that maps an image into a set of one-dimensional projections, which represent the sum of the pixel intensities along a line in the image. By comparing the projections of two images, we can determine the alignment between them.

$$Rf(\rho, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\delta(\rho - x\cos\theta - y\sin\theta)\, dx\, dy \tag{3.1}$$

where $f(x, y)$ is the input image, ρ is the distance parameter from the origin and θ is the angle parameter. The delta function δ enforces that the line integral in the transform passes through the point (x, y) at angle θ and distance ρ from the origin. The resulting 2D array from the transform displays ρ values along the rows and θ values along the columns. For every angle θ and distance t, the transform generates a set of projections.

In our case it has been used to align an image of fabrics as in [20], taking into account that the yarns of the fabrics must be aligned vertically or horizontally. The process of self-alignment using Radon transform that we used involves the following steps:

Algorithm 1 Image Alignment using Radon Transform

1: Filter the input image using a Laplacian filter operator to emphasize edges.
2: Compute the two-dimensional Radon transform of the image along a set of different angles $Rf(\rho, \theta)$ where $\theta \in [-15, 15] \cup [75, 105]$ with a step size of 0.25.
3: Calculate the variance of each Radon transform result for each angle.
4: Select the angle θ_{misalign} with the highest variance.
5: Apply a rotation matrix $R_{\text{misalign}}(\theta_{\text{misalign}})$ to the original image to correct the misalignment.

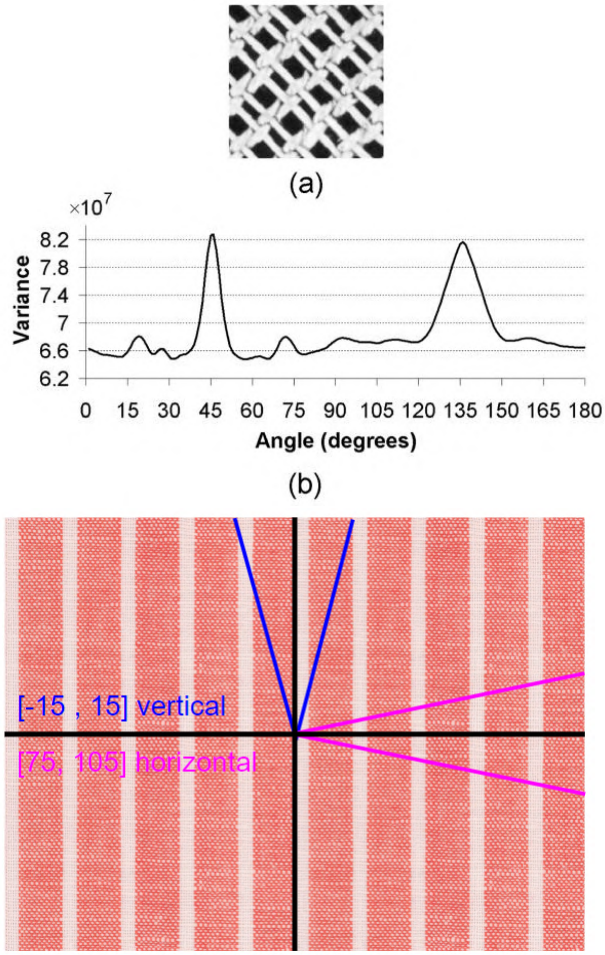


Figure 3.2: Top figure from [20]. (a) A directional texture rotated at 45°, (b) variance of projections at different angles. Both local maxima of the variance of the projections are correct. Bottom figure shows the angles direction (θ) used in radon transform algorithm.

Algorithm 1 is the standard process for one image, but we have added a multi-scale evaluation. The multi-scale is composed of an original image size, a centre crop of 512 and a centre crop of 256. With these central crops, we aim to look at the structure of the yarn in order to better align the fabric. The above process is performed for each scale and the final result is decided using the median.

The auto-alignment stage is crucial and can determine the success of the entire algorithm. It is important that the fabric image has a vertical or horizontal orientation of yarns rather than a diagonal one.

3.2 Minimum repeatable tile

The goal of this section is to describe the entire process of detecting the minimum repeatable tile. It involves four main steps, namely image pre-processing, filter extraction, creation of distance map, and finding the best crop tile.

3.2.1 Overview

The minimum repeatable tile (MRT) algorithm is an approach designed to extract the smallest repeating pattern from fabric images. This subsection provides an overview of the MRT algorithm, highlighting its key steps and their significance in achieving accurate pattern extraction. The algorithm combines image preprocessing, feature extraction using AlexNet, distance map creation, selection of the best size pattern, and stitching to produce a tiled image using the minimum repeatable tile. The algorithm steps are presented and described below, accompanied by a comprehensive pipeline in Figure 3.3.

1. **Pre-process image:** apply image normalization using mean and std from ImageNET dataset.

2. **Feature Extraction:** We use AlexNet, a popular pre-trained CNN model, to extract feature maps from convolutional layers. We selectively retain only the layers that exhibit the highest standard deviation, ensuring that we capture the most significant and distinctive features for further analysis.

3. **Distance Map:** After selecting feature maps with high standard deviation, we generate a distance map by accumulating the distances between local peaks within each layer. This histogram, also known as the distance map, illustrates the most frequently occurring distances among potential repeating patterns in the image.

4. **Select best crop size of repeated pattern:** We select the top 5 most repeated points from the histogram (distance map) and generates a tiled image using these points. The candidate patterns in the tiled image are then evaluated using the MS-SSIM metric to measure their similarity to the original image. By analyzing different pattern sizes, the algorithm identifies the size that maximizes both pattern similarity and occurrence, leading to the selection of the most suitable minimum repeatable tile.

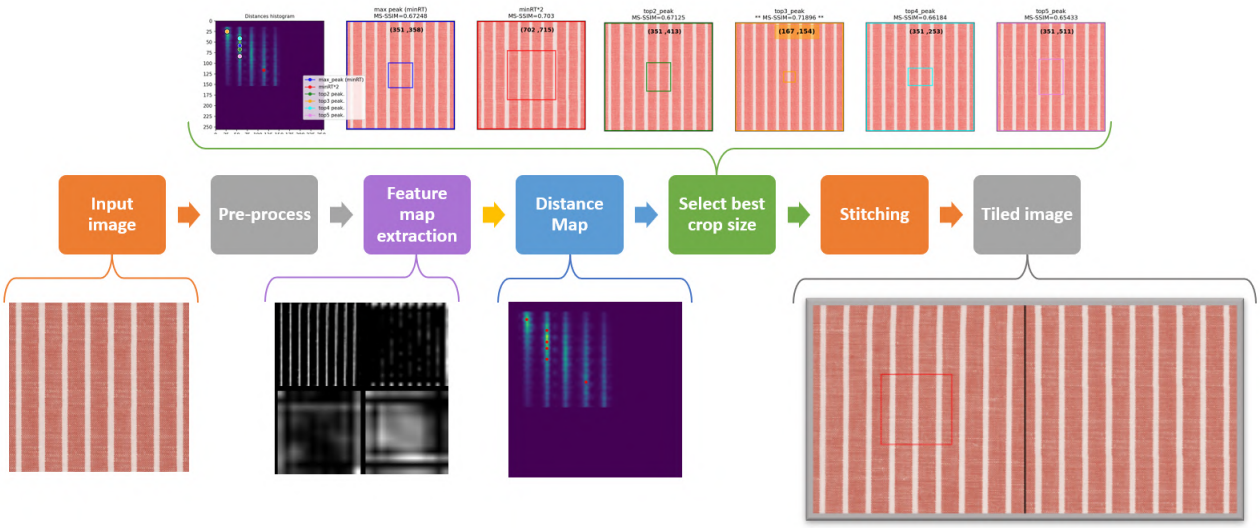


Figure 3.3: Overview of complete pipeline of MRT algorithm and stitching step. In tiled image result (gray), is shown the original image containing the minimum repeatable tile marked by a red box, and the final seamless tile after apply the stitching correction.

3.2.2 Preprocess image

In this step, a pre-processing is applied to the image. We applied a image normalization (also called standardization) that involves adjusting the pixel values of an image to make them more consistent and easier to work with for machine learning algorithms. Normalization using the mean and std from a dataset is useful because it ensures that the pixel values have similar statistical properties across all images in the dataset. This makes it easier for deep learning algorithms to learn features that are relevant to the task, and can improve the overall performance of the model.

The method of normalization used has been the mean and standard deviation (std) of the pixel values from a dataset (ImageNet). The pre-trained network that we use is AlexNet, and is trained with dataset normalization. In order to obtain the most similar results to the original work we use the same process.

$$I' = \frac{I - m}{std}$$

where $m = [0.485, 0.456, 0.406]$ and $std = [0.229, 0.224, 0.225]$ are the mean and standard deviation of the imageNET dataset for each channel (rgb).

3.2.3 Filter extraction from AlexNet

Convolutional Neural Network (CNN) have the ability to learn hierarchical representations of features that prove useful in a variety of tasks, such as image classification, object detection, and style transfer. These features are learned through convolutional filters that detect specific patterns within images. Initial layers typically learn to identify low-level features like edges, while deeper layers learn more complex features such as objects or intricate shapes. One major advantage of using deep CNNs instead of manually designing filters is that they can learn filters from data, making them more resilient and less reliant on human assumptions or prior knowledge of the problem.

Our method takes advantage of the multi-level capabilities of CNN filters. The purpose of this step is to detect repetitions in the image by examining the activation peaks in the filter space. By analyzing the distances between these peaks in feature space, we can determine the most likely size of the minimum repeatable tile. The common distance that consistently appears is the predominant size of the repeating pattern. This information is crucial for accurately identifying and extracting the minimum repeatable tile from the image.

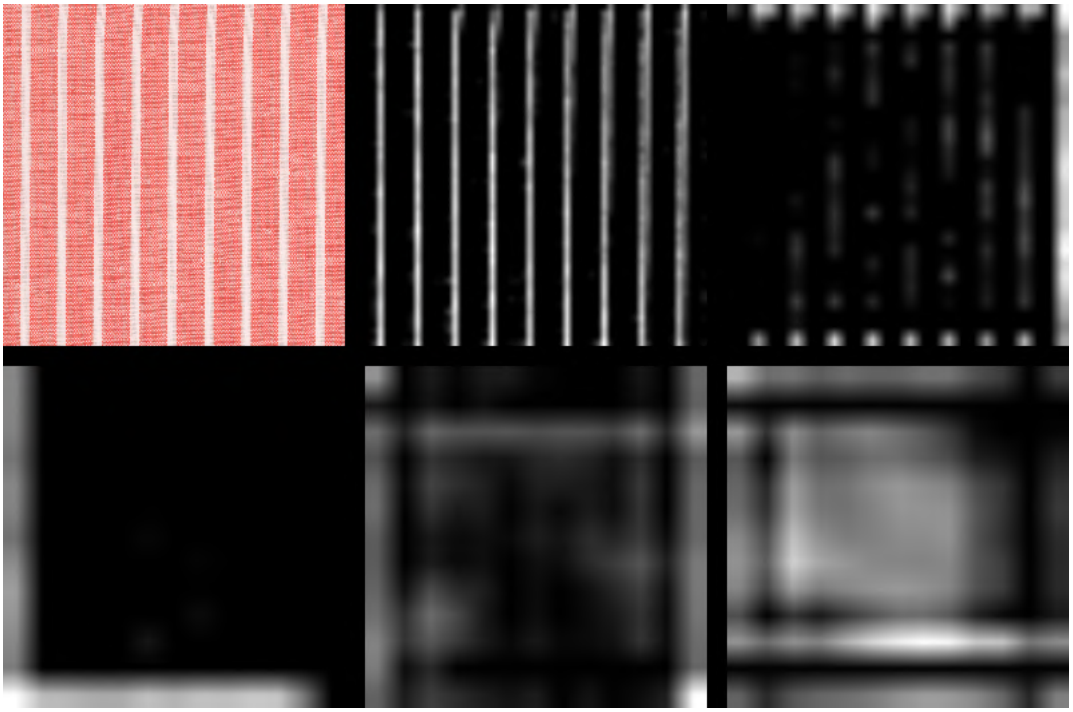


Figure 3.4: The images displayed from left to right include the original image and randomly selected image from each convolutional layer of the network starting from layer 1 to layer 5 in increasing depth.

In order to optimize the computational efficiency of our algorithm, we select a subset of filters F_l based on their standard deviation (STD). Specifically, we only keep the filters with a std value greater than 85% of the distribution of std values for the entire layer. This ensures that we are only including filters that have a significant response and ignoring those that do not contribute much to the overall computation.

3.2.4 Creation of Distance map

After selecting the filters with the highest standard deviation, the next step is to create a distance map.

In each layer of the network, the input image generates different local peaks (Figure 3.5 (b)), each representing a specific distance vector. These distance vectors capture the spatial relationships between the peaks. To accurately determine the appropriate size of the pattern primitives, the algorithm employs the Hough voting space, where we accumulate the Euclidean distances between each peak and the others. This accumulation is done in a 2D histogram (x,y), called as a distance map and is displayed in Figure 3.5 (d). This process enables the algorithm to effectively identify the optimal size of the pattern primitives because all of this information is extracted from the feature map of the image.

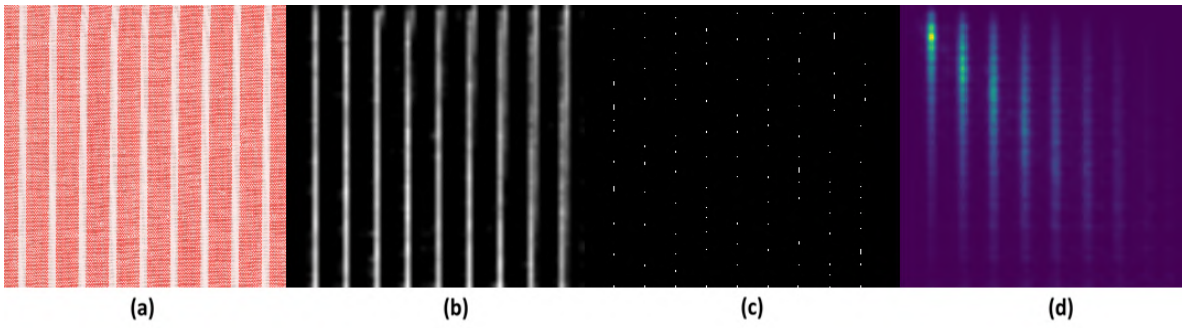


Figure 3.5: The images displayed from left to right include the original image, selected image from convolutional layer, the Non-Maximum Suppression (NMS) algorithm applied to selected image and distance map.

Algorithm 2 Distance Map Calculation

1: **for** each feature map **do**
2: Apply NMS algorithm to generate regular activation peaks.
3: Obtain the centroids of the activation peaks using Connected-Component analysis.
4: Calculate Euclidean distance between all centroids.
5: Accumulate the distances in the Hough space voting (distance map).

After the algorithm completes the process for all the selected filters, it generates a 2D histogram known as a distance map. A regularisation is applied on the map to weight the squared results. This map is used to identify the most common distances between the peaks of the filters, which are represented by the peaks in the histogram. These peaks are considered the most repeated distances and are used to identify the size of the periodic pattern primitive.

3.2.5 Selection of best size pattern

To detect peaks in the histogram or distance map, the algorithm employs a technique used in last step known as Non-Maximum Suppression (NMS). This technique involves searching for the maximum value in a local neighborhood and setting all other values in that neighborhood to zero. This ensures that only the highest values or peaks are retained in the histogram. After the NMS is applied, the algorithm obtains the centroids of the remaining activation peaks. The centroids correspond to the positions in the histogram where the peaks are situated (x, y), where x refers to the width and y to the height, while the values at those locations indicate the sizes of the most frequently recurring patterns.

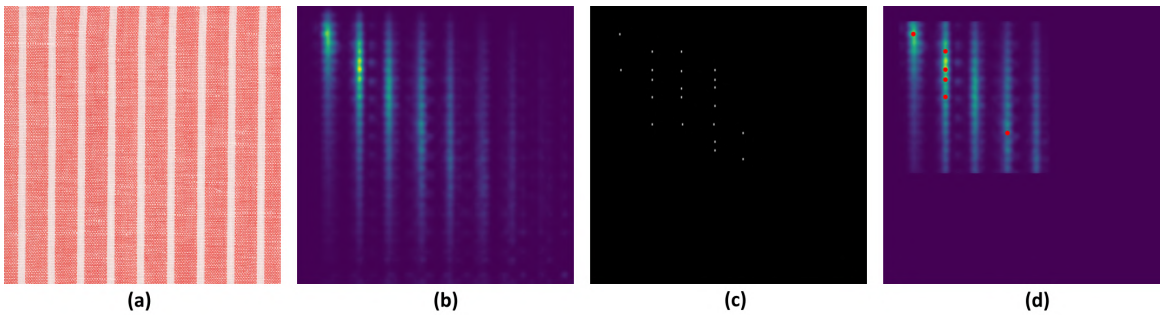


Figure 3.6: Displayed from left to right are the following images: the original image, the distance map, the non-maximum suppression of the distance map, and the final distance map with the detected peaks.

Figure 3.6 (d) show the applied constraint on the distance map. To prevent excessively large distance vectors, we impose restrictions at both the lower and upper bounds of the distance map size. To guarantee a minimum distance vector size, we enforce a minimum size of $\frac{dim_{dm}}{20}$, where dim_{dm} denotes the distance map size. Additionally, we place a maximum restriction of $\frac{3}{5} \cdot dim_{dm}$.

Unlike previous works that relied on selecting the maximum peak of the histogram [18, 17, 19], we select the top N peaks, where N is set to 5. The process involves selecting the top 5 peaks from the extracted image and cropping the original image at the center using the (x, y) coordinates equal to its width and height. These coordinates correspond to the most frequently repeated pattern in the original image. Then, a tiled image is created by repeating the crop until the size of the input image is obtained. Finally, we evaluate the original image with the tiled image, and choose the image that achieves the highest score according to the Multi-Scale Structural Similarity Index (MS-SSIM) index. In Section 2.2 provides a detailed explanation of this metric.

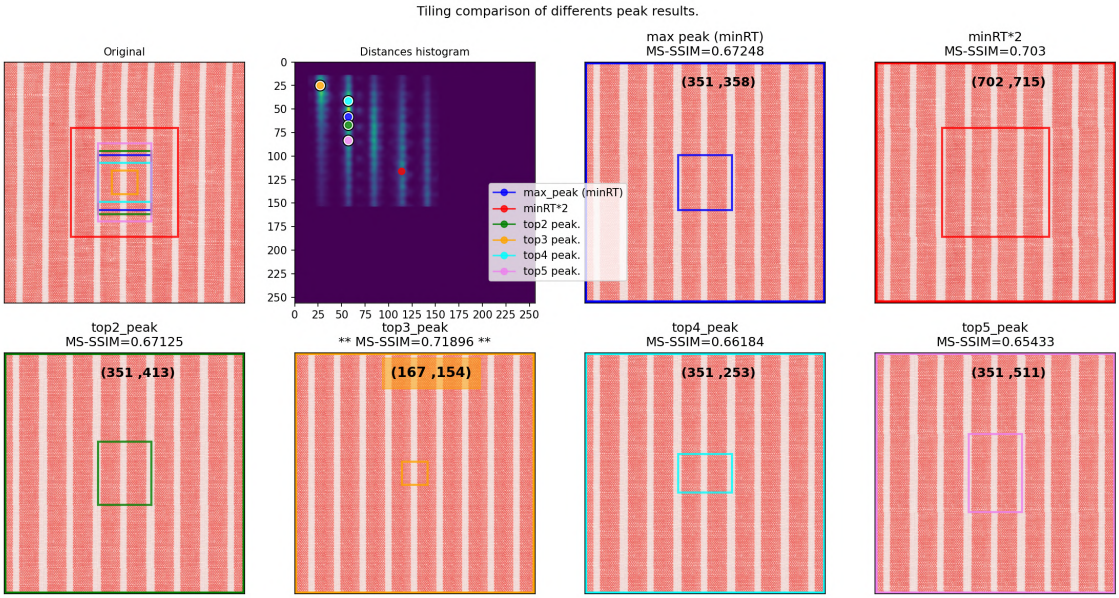


Figure 3.7: The images are presented from left to right as follows: the original image, the distance map with detected peaks, and for each peak, the resulting tiled image.

Based on our experiments, we have found that in some cases, it is beneficial to add an additional centroid to the histogram (red dot in Figure 3.7). This is because a larger pattern can better capture the variance of the yarns in the fabric texture. The additional centroid is determined by doubling the value of the maximum peak in the histogram,

provided that it is not in close proximity to any other peak or outside the specified constraints.

3.3 Stitching

In this section, we will discuss the final step of the Auto-tiling algorithm, which is known as stitching. The primary objective of this step is to address or prevent any imperfections that may arise due to the tiling process. After detecting the minimum repeatable tile of a fabric, the stitching step ensures that the final output is seamless and visually appealing. This is particularly important for applications where the quality and appearance of the texture are critical, such as in digital art, video game assets, and virtual clothing try-on. By stitching the tiles together seamlessly, the Auto-tiling algorithm creates a natural and cohesive overall texture that maintains the pattern's visual continuity. The stitching step can also involve various techniques such as blending, feathering, and cloning, depending on the specific requirements of the application. Overall, the stitching step is a crucial aspect of the Auto-tiling algorithm, ensuring that the final texture output meets the high standards required by modern digital fashion applications.

The main algorithm used have been seam-carving [21, 22] to correct or avoid imperfections caused by the tiling process. Seam-carving involves removing or adding seams of pixels in an image to resize or distort it while preserving the image's essential features. This technique can be used to eliminate undesirable patterns or distortions that may have occurred during the tiling process, using the minimum energy to detected a seam and applied a blending in this area to avoid artifacts in final image. The algorithm create a smoother transition between the tiles and correct any misalignments that may have occurred during the tiling process. This technique can be particularly useful when working with large fabric textures or repeating patterns, where misalignments can be more noticeable.

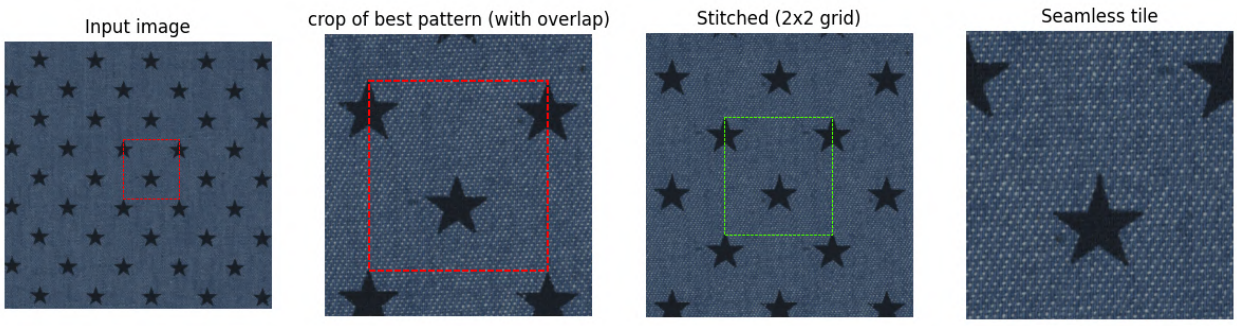


Figure 3.8: The images displayed from left to right: the original image containing the minimum repeatable tile marked by a red box, the minimum repeatable tile is cropped with an overlap, and the area of interest is highlighted with a red box, the stitched image composed of a 2x2 grid with seamless tile highlighted in green, and the final seamless tile.

The steps of this algorithm are as follows:

- First, the minimum repeatable tile is cropped with an overlap margin, and this cropped tile is repeated 2x2 times.

- The absolute difference in the overlap region is then calculated, and a blur filter is applied to remove any noise present in the image.

- Next, the mask is inverted, and the detected seam (mask) is dilated to increase the blending area.

- Finally, a blending of the overlapped images is performed within the masked area using the previously generated mask.

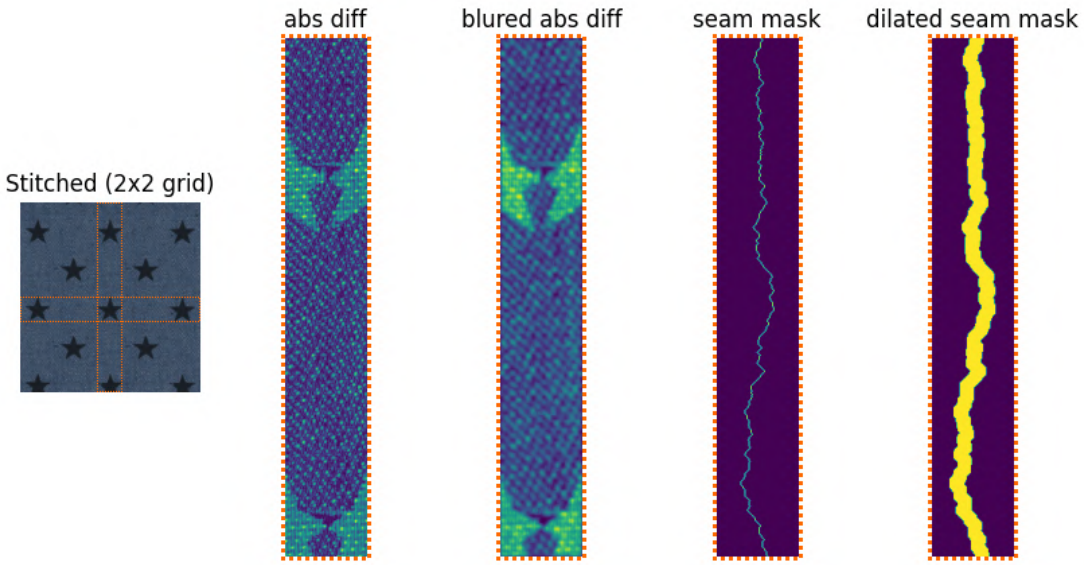


Figure 3.9: The sequence of images from left to right displays the following: a stitched image consisting of a 2x2 grid, with marked in orange the overlapped section, followed by the absolute difference of the overlap section, a blurred version of the previous image, the detected seam mask, and the dilated seam mask.

By following these steps, the stitching algorithm is able to detect and correct any imperfections that may have occurred during the tiling process, ensuring that the final output is seamless and visually appealing.

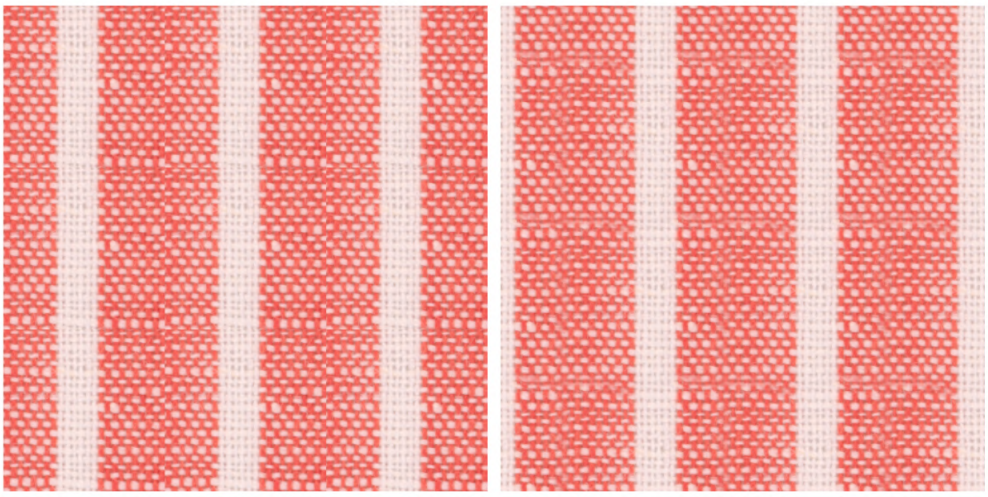


Figure 3.10: Tiled image with no correction and with stitching correction.

The implementation of this code have been developed by SEDDI. I have incorporated a portion of their code into my work with their permission.

3.4 Implementation

3.4.1 Programming language

The programming language chosen for the project is Python. According to Wikipedia, this programming language is considered high-level and is characterized by a philosophy that emphasizes a syntax that favors readable code. It is a multi-paradigm programming language, as it supports object-oriented programming, imperative programming, and, to a lesser extent, functional programming. It is an interpreted language, with strong typing (data type violations are not allowed), dynamic, and cross-platform.

Python is the most widely used programming language for artificial intelligence and data science, particularly for deep learning. This is thanks to its simplicity and the number of existing libraries for these purposes. Being an interpreted language abstracts the user from all low-level complexities, allowing the user to focus on the problem and how to quickly translate ideas into code. Thanks to libraries for machine learning development, advanced knowledge is not necessary to initiate and develop the code.

Visual Studio Code (VS Code) was used to develop entire project. It is a free source-code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It also has a wide range of extensions available, which can add features like language support, debugging tools, and code analysis. VS Code is designed to be highly customizable, allowing users to personalize their experience with different themes, keyboard shortcuts, and settings. It is widely used in software development, particularly for web development and data science.

In order to execute some test of code we also used a platform called Google Colab. This is a cloud service that provides us with a Jupyter Notebook which can be accessed using a web browser, regardless of whether we use Windows, Linux or Mac. This platform allows us to activate a GPU for training, has pre-installed libraries used in data science, and can be linked to our Google Drive account. It offers all of this for free, but with some

limitations that we will discuss during our work.

3.4.2 Packages

- **Scientific computing: NumPy and Scipy**. NumPy is a library that provides efficient numerical arrays and a wide range of mathematical operations to work with them. Scipy is another library that builds on NumPy and provides additional tools for scientific and technical computing, such as optimization, signal processing, and statistics.

- **Pytorch**: PyTorch is an open-source machine learning library developed by Facebook's artificial intelligence research group. It provides an efficient and easy-to-use platform for building and training deep neural networks.

- **Open Source Computer Vision Library (OpenCV)**: is an open-source computer vision and machine learning software library. It provides a common infrastructure for computer vision applications and has support for various programming languages such as C++, Python, and Java. It contains over 500 functions that cover a wide range of areas in the vision process, such as object recognition (facial recognition), camera calibration, stereo vision, and robotics vision.

- **Open Neural Network Exchange (ONNX)**: is an open-source platform that enables interoperability between different deep learning frameworks by defining a common format for exchanging trained models. It allows for seamless integration of models across different software and hardware platforms.

- **ONNX Runtime**: is an open-source high-performance engine for deploying machine learning models. It is optimized to run models in various hardware configurations, including CPUs, GPUs, and edge devices.

- **Pandas**: is a Python library used for data manipulation and analysis. It provides data structures for efficiently storing and manipulating large, heterogeneous datasets, as well as functions for cleaning and transforming data.

- **Matplotlib**: is a data visualization library for creating static, animated, and interactive visualizations in Python. It provides a wide range of plotting options

and customization features to create high-quality and informative graphs, charts, and figures.

- **Streamlit**: is an open-source library that allows you to create interactive web applications for data science and machine learning projects quickly. It provides a simple and intuitive way to build a user interface around your Python code, enabling you to easily create and share interactive data apps.

Chapter 4

Experiments and results

4.1 Dataset

Textile digitization is a complex process that involves capturing the mechanical and optical characteristics of fabrics and converting them into digital data. This conversion enables the creation of accurate 3D models. However, due to the intricacies of textiles, achieving reliable and precise digitization often requires advanced and costly solutions.

At SEDDI, we possess a comprehensive dataset of fabrics, which has been obtained using sophisticated equipment to accurately determine the optical and mechanical properties of each fabric. The textile industry tends to prefer more accessible and affordable digitization tools like traditional scanners for their fabric digitization needs. Despite the difference in equipment, our algorithms can predict the properties of fabrics using scanned images with minimal errors.

To achieve this, in this project we used 133 RGB images captured by the scanner (see Figure 4.1).

Figure 4.1: Radom images selected from our fabrics dataset.

The images used have been chosen to meet the algorithm's requirements, which are based on detecting the minimum repeatable tile. For the algorithm to work accurately, it is crucial to satisfy particular criteria. The algorithm has limitations, as the fabric image cannot contain non-rigid transformations or distortions, and it requires more than two repetitions of the same pattern.

As shown in Figure 4.2, some examples were excluded due to the lack of a repetitive pattern, less than three repetitions, or distortion in the image capture.

Figure 4.2: Radom images were deleted from our fabric dataset.

4.2 Execution time

This section focuses on evaluating the performance of our code across various devices and packages. First, with a general analysis of the algorithm and going deeper into the part of MRT with the different tests performed, we compare the execution times of ONNX Runtime and PyTorch for both CPU and GPU inference.

We have the complete pipeline of the auto-tilting algorithm as shown in Figure 4.3. We can achieve a result for auto-tiling (excluding auto-alignment) in approximately 2 seconds, which is a very satisfactory outcome for a practical application. The MRT algorithm is executed on the CPU using ONNX, we will focus on this later.

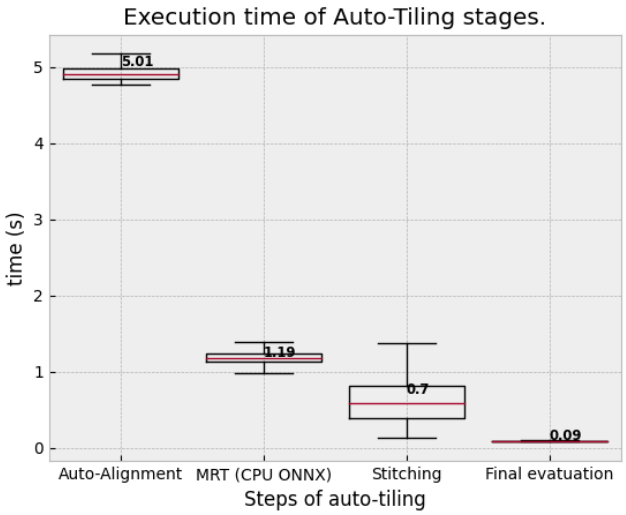


Figure 4.3: Execution time in seconds of all steps of auto-tiling algorithm. We use CPU with ONNX in MRT step. The number in the box is the mean execution time of all dataset.

Figure 4.4 shows the execution time of the complete MRT algorithm. We can see how feature map extraction and selecting the best crop size are on average the slowest steps, but they are actually an affordable time for a real application.

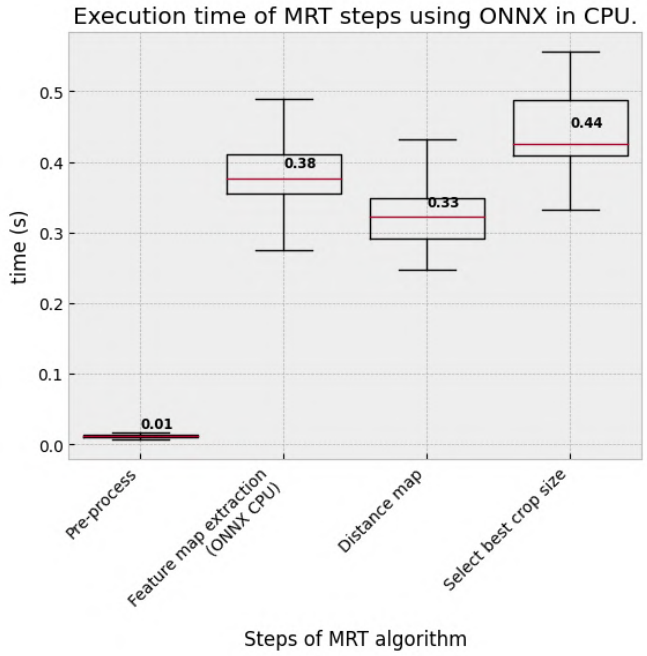


Figure 4.4: Execution time in seconds of all steps of MRT algorithm. We use CPU with ONNX in MRT step. The number in the box is the mean execution time of all dataset.

The last point that remains to be explained is the comparison of libraries and devices used and why we finally chose CPU ONNX.

PyTorch is a powerful deep learning framework that allows developers to create and train complex neural networks. However, when it comes to deploying models in production, it is often not the best choice due to its heavy footprint and lack of support for certain platforms. ONNX, on the other hand, is a lightweight and efficient format for representing deep learning models and allows easy conversion between different deep learning frameworks, including PyTorch, which means that you can train your models in PyTorch and then convert them to ONNX format for deployment. ONNX Runtime is specifically optimized for running ONNX models on different devices, including CPUs, GPUs, and edge devices, further enhancing performance and flexibility. This can help speed up your code and make it more efficient, especially on resource-constrained devices like mobile phones and IoT devices.

The purpose of this application is to bring this work to the product of SEDDI Textura. To achieve this goal, it is necessary to implement the algorithm as quickly as possible and make it executable on the front-end with JavaScript. We have evaluated various options for this purpose, including setting up a back-end to run on the GPU, but it has been dismissed due to its cost. Therefore, we have arrived at the ONNX Runtime, which allows us to perform inference on the front-end and many other platforms.

The part of the code that will be used with ONNX is the part that performs the AlexNet inference to extract the filters from the CNNs. Our initial implementation of the code employed an optimization algorithm to perform feature visualization (Section 2.1.2), as the output image of each filter in each layer differed and was smaller in size. The goal was to obtain an image of the original size to enable a search for repetitive patterns. The optimization algorithm produced good results, but it was discarded due to the high computational requirements. The final implementation resized the image to the original size, resulting in less accurate outcomes but with the advantage of being able to run in nearly real-time.

Figure 4.5 illustrates the execution times of the complete minimum repeatable tile (MRT) algorithm using different devices and libraries, in which AlexNet inference is integrated.

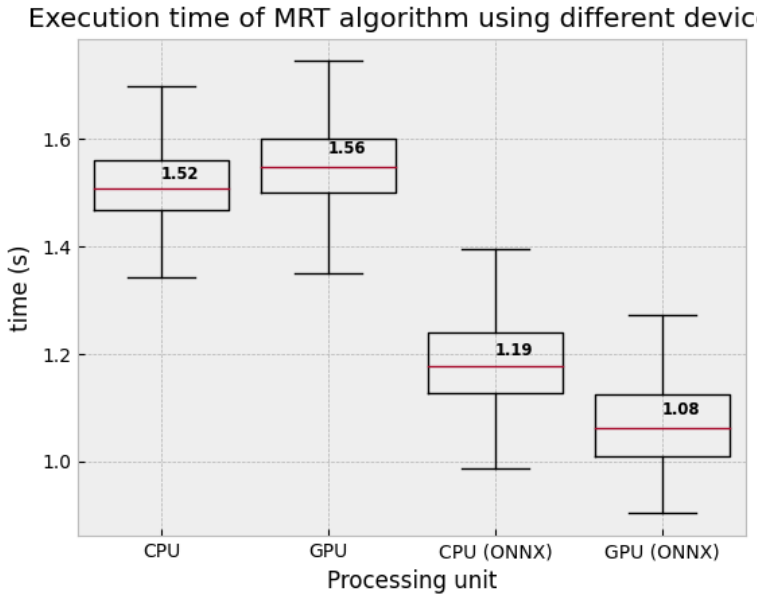


Figure 4.5: Execution time in seconds of CPU and GPU inference using torch, and CPU and GPU using ONNXruntime. The number in the box is the mean execution time of all dataset.

The results obtained align with our expectations, as ONNX is optimized for inference and has managed to outperform Torch. One surprising finding is that in Torch, the GPU performs slower than the CPU. This is because the image needs to be moved to the internal memory of the GPU for execution, which, in some cases, can be detrimental to its performance. However, since the images in this study are not very large (256 pixels in the inference step), the results are relatively consistent across all cases.

For achieving fast performance in browser deployment, the final implementation of the algorithm uses the ONNX CPU framework.

If the fabric is properly aligned, the algorithm will perform accurately and quickly. However, auto-alignment is not yet practical for real-world scenarios as it was only used for demonstration purposes and not for production. This is due to the need for multiple Radon transformations at various angles to determine the rotation angle. This could be a potential avenue for future research.

4.3 Evaluation

To evaluate the effectiveness of our method, we compared the original fabric image with the final tiled output using the MS-SSIM metric, which measures the structural similarity between two images. The MS-SSIM score ranges from 0 to 1, with a score of 1 indicating a perfect match between the two images.

It is important to note that this result does not necessarily indicate success, as the MS-SSIM metric only indicates the similarity between the two evaluated images, which is what we are looking for. However, there may be cases where the images are similar but the tiled result is not correct. Therefore, a manual review should be performed to certify its correctness. The manual revision of results will provide confirmation that the algorithm's output is accurate and valid. The table below presents the results of the manual revision process for a sample of tiled images.

Algorithm	**Success rate (%)**	**Error rate (%)**
Auto-tiling	**55**	45
MRT	-	**15**
Other	-	30

Table 4.1: Results of manual revision of auto-tiling algorithm.

The Table 4.1 shows the success rate and error rate of the algorithm, broken down into two categories: minimum repeatable tile (MRT) and other. The auto-tiling algorithm had an overall success rate of 55%, indicating that were accurate after manual revision. The error rate was 45%, meaning that the tiled result was found to be incorrect after manual revision. The table also displays the error rate of the MRT, which is our main contribution, at 15%. The *Other* error represents any error that is not related to the MRT, such as misalignment in the auto-alignment algorithm, stitching issues, distortions in the fabric image, or fabrics with two different yarn orientations.

In the following table, the results of MS-SSIM with different thresholds are verified with the results of the manual verification, which determines the correctness of the output.

	MS-SSIM	After manual review
threshold	**Success rate (%)**	**Success rate (%)**
0.6	89.5	59.4
0.7	78.9	59.4
0.8	63.9	54.8

Table 4.2: Results of MS-SSIM and verification with manual revision.

From the Table 4.2, we can observe that as the MS-SSIM threshold increases, the success rate decreases. This indicates that a higher threshold is more restrictive in determining the similarity between the tiled image and the original image.

However, it is important to note that the success rates obtained from the MS-SSIM metric alone do not align perfectly with the success rates from the manual review. This suggests that the MS-SSIM metric may not capture all the details and complexities involved in assessing the correctness of the tiled results.

Furthermore, it should be noted that in the Table 4.1, there is a separate category indicating an error rate of 30% for reasons unrelated to the minimum repeatable tile (MRT). This implies that there are other factors contributing to errors in the auto-tiling algorithm, which may include issues such as auto-alignment problems, stitching imperfections, or distortions in the fabric images. In more ideal conditions, where these additional factors are minimized or eliminated, it is possible that the overall success rate could be improved.

The lower success rates after the manual review indicate that the algorithm's performance is not perfect and there is room for improvement. Further refinements and enhancements to the algorithm may be necessary to increase the success rate and ensure more accurate tiled results.

4.4 Correct results

This section presents our correct results of auto-tiling process. Each image is displayed with the original version on the left, while the minimum repeatable tile is highlighted in red. On the right, the final tiled image is shown.

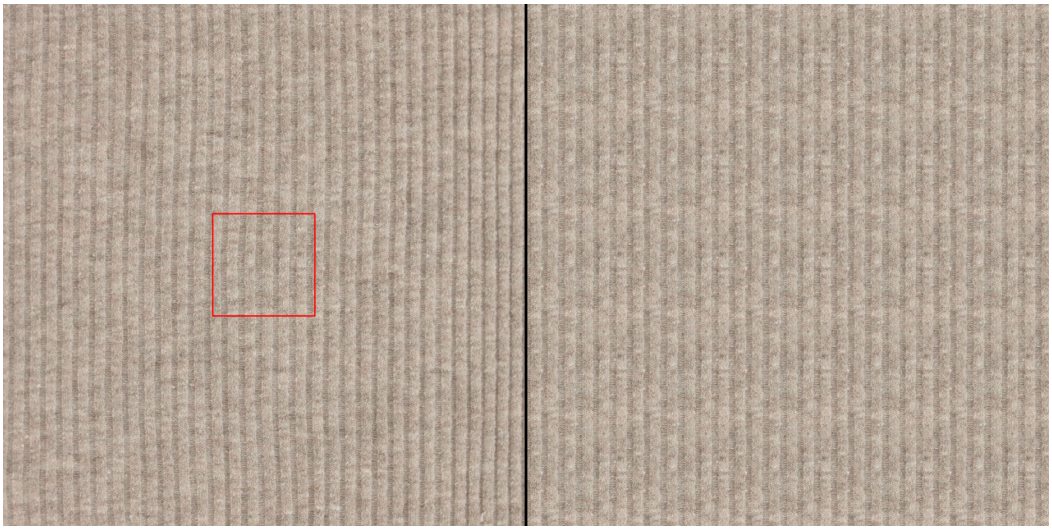


Figure 4.6: MS-SSIM=0.604.

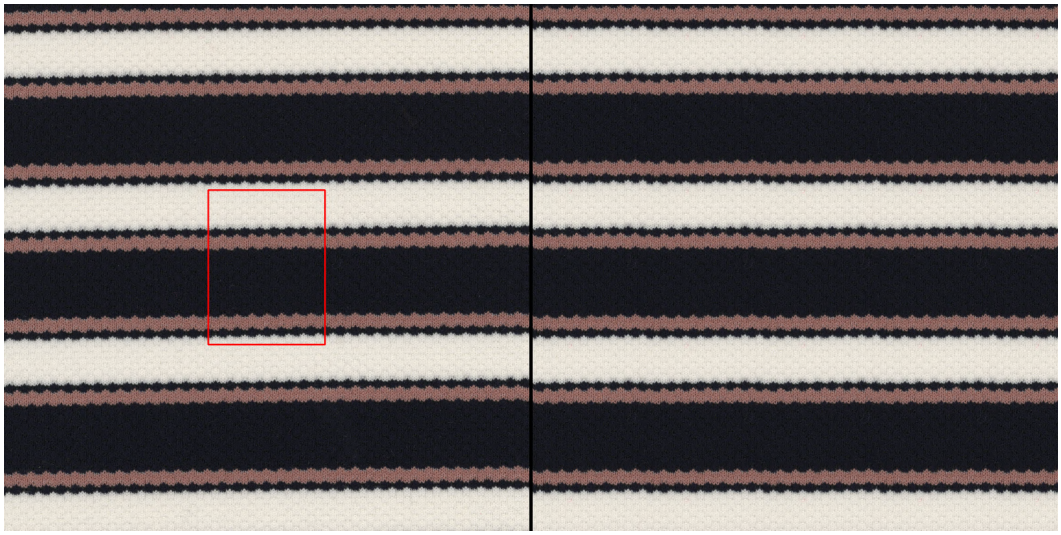


Figure 4.7: MS-SSIM=0.959.

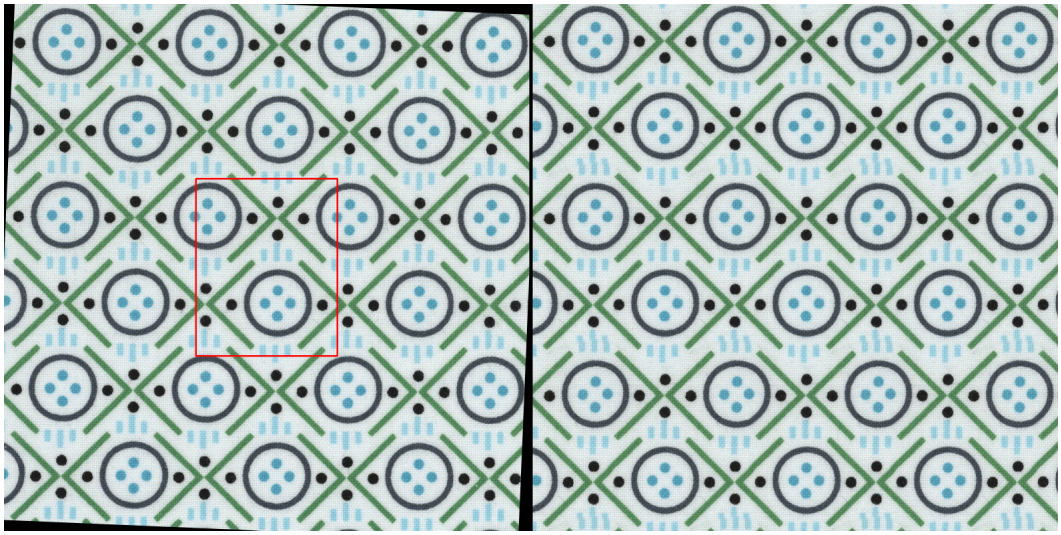


Figure 4.8: MS-SSIM=0.771.

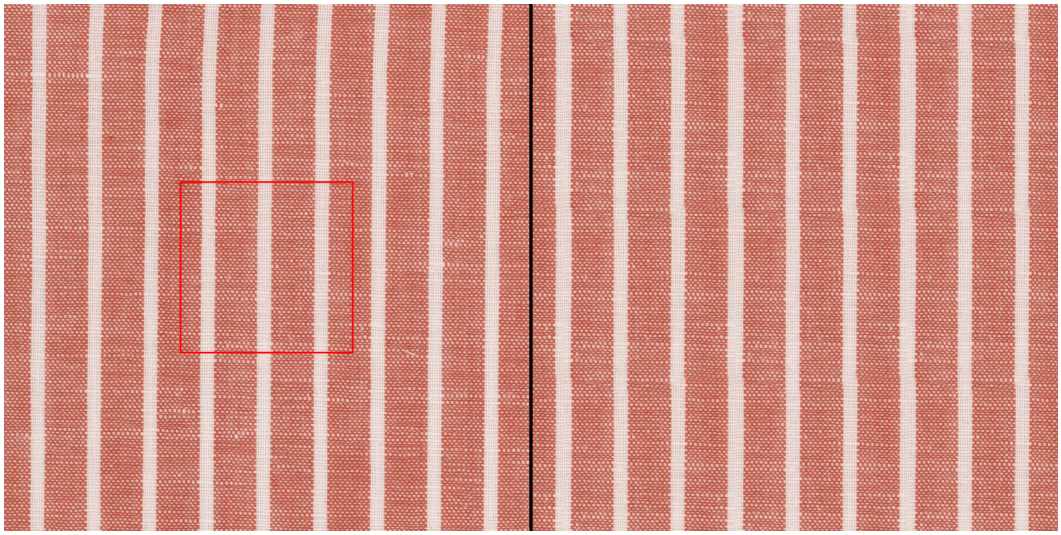


Figure 4.9: MS-SSIM=0.924.

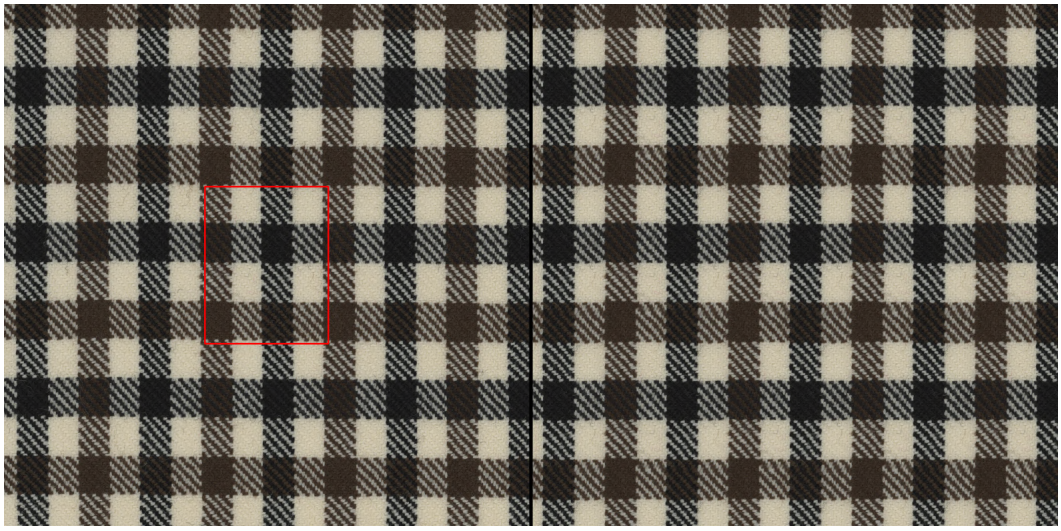


Figure 4.10: MS-SSIM=0.939.

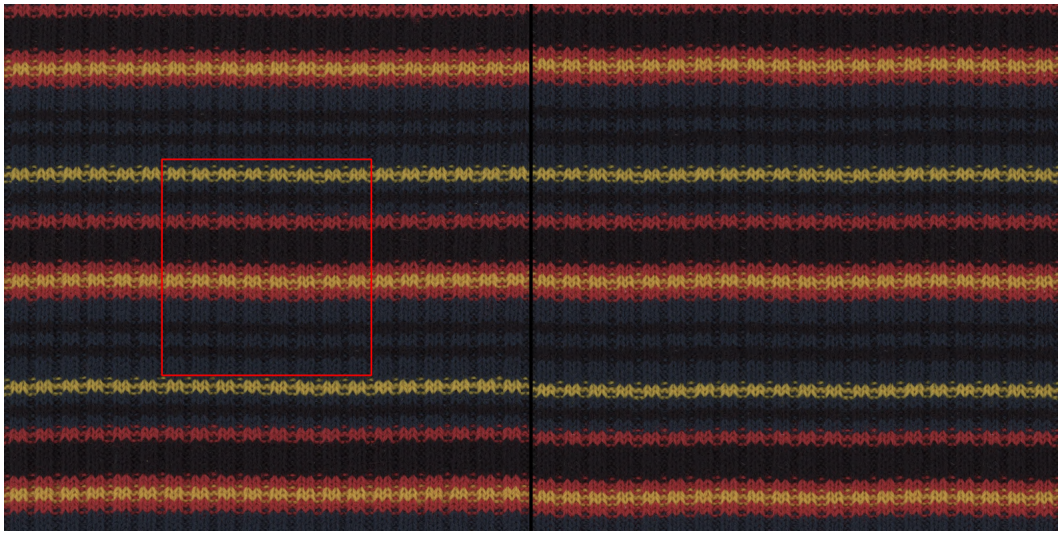


Figure 4.11: MS-SSIM=0.939.

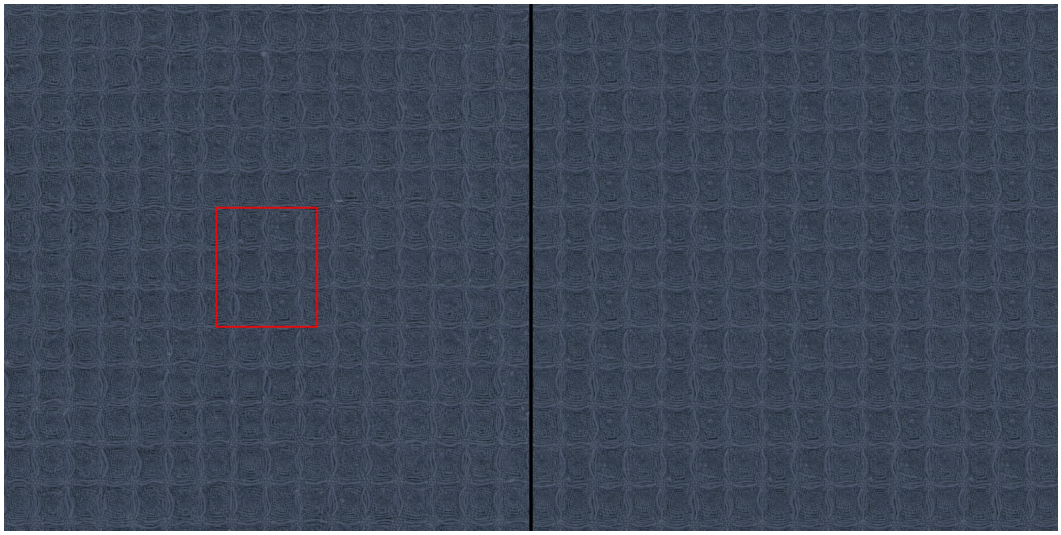


Figure 4.12: MS-SSIM=0.944.

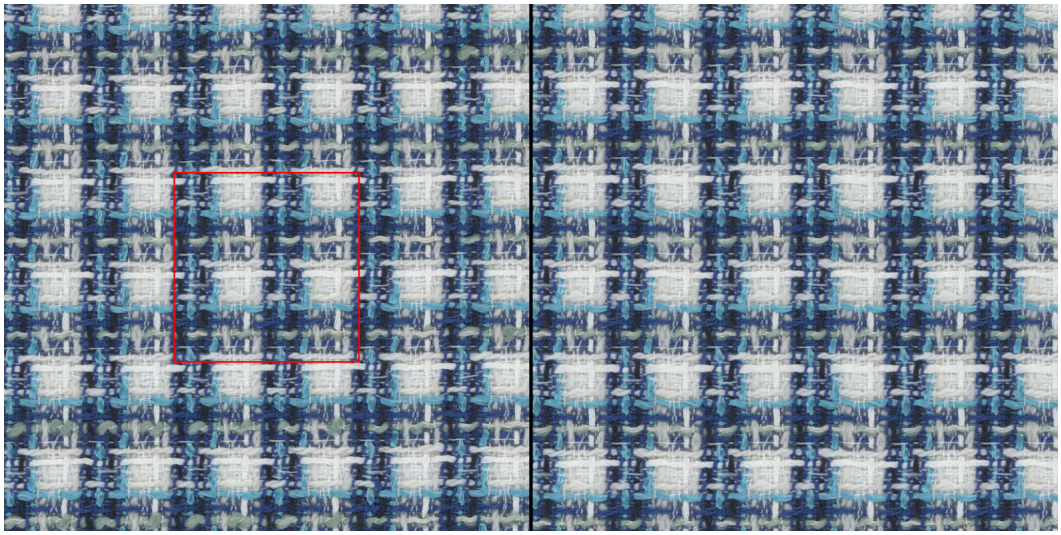


Figure 4.13: MS-SSIM=0.857.

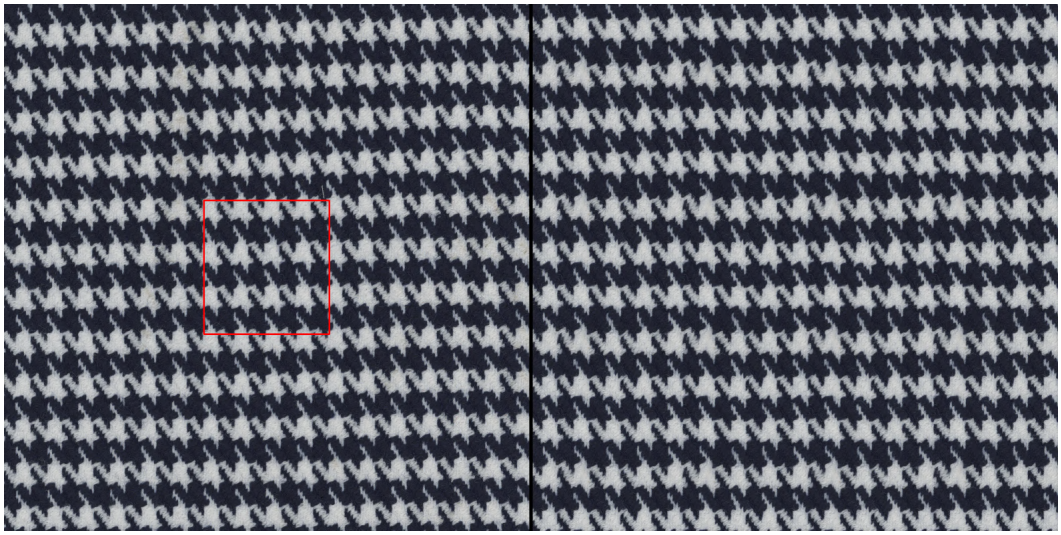


Figure 4.14: MS-SSIM=0.885.

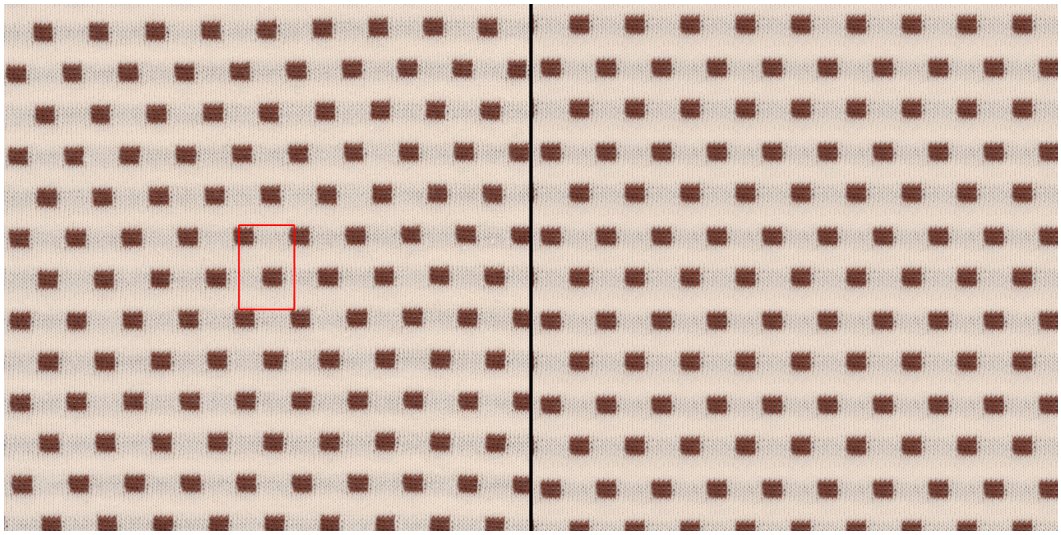


Figure 4.15: MS-SSIM=0.889.

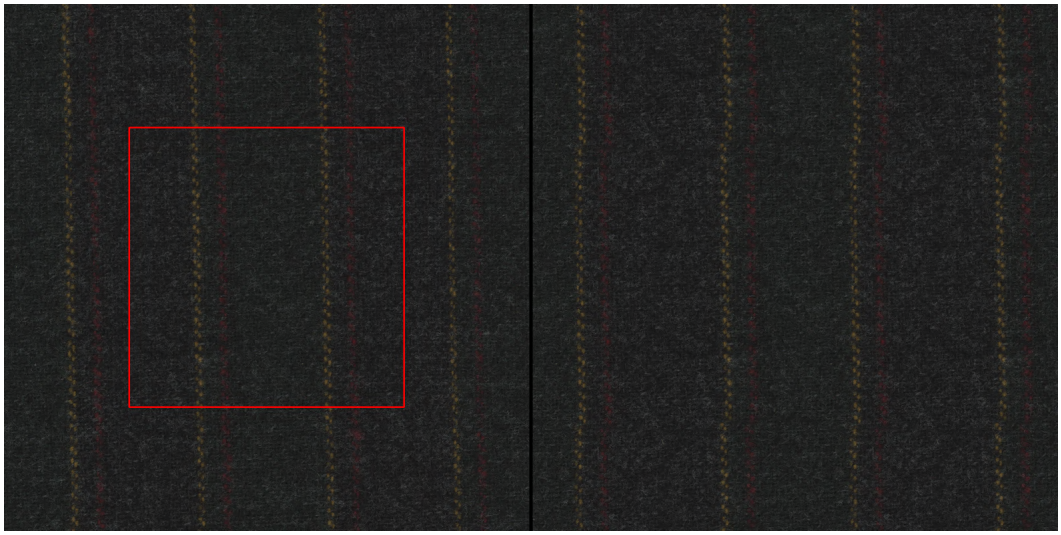


Figure 4.16: MS-SSIM=0.908.

4.5 Incorrect results

This section presents some inaccurate results of the auto-tiling process. Each image is accompanied by a commentary on the potential error in each case, whether it's caused by fabric distortion, auto-alignment, MRT, or stitching. The original version of each image is displayed on the left, with the minimum repeatable tile highlighted in red. On the right, the final tiled image is shown.

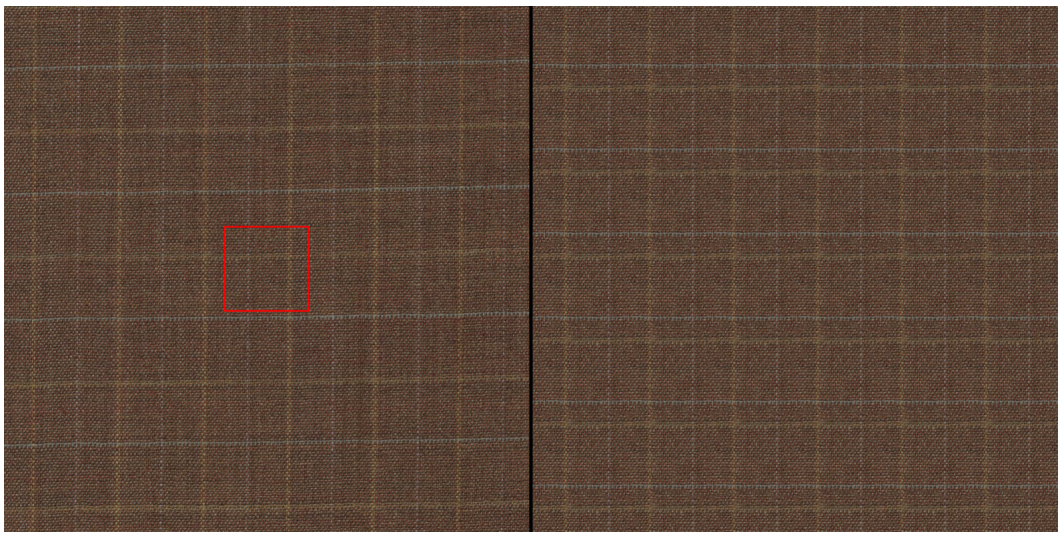


Figure 4.17: MS-SSIM=0.870. Incorrect result due to MRT failure.

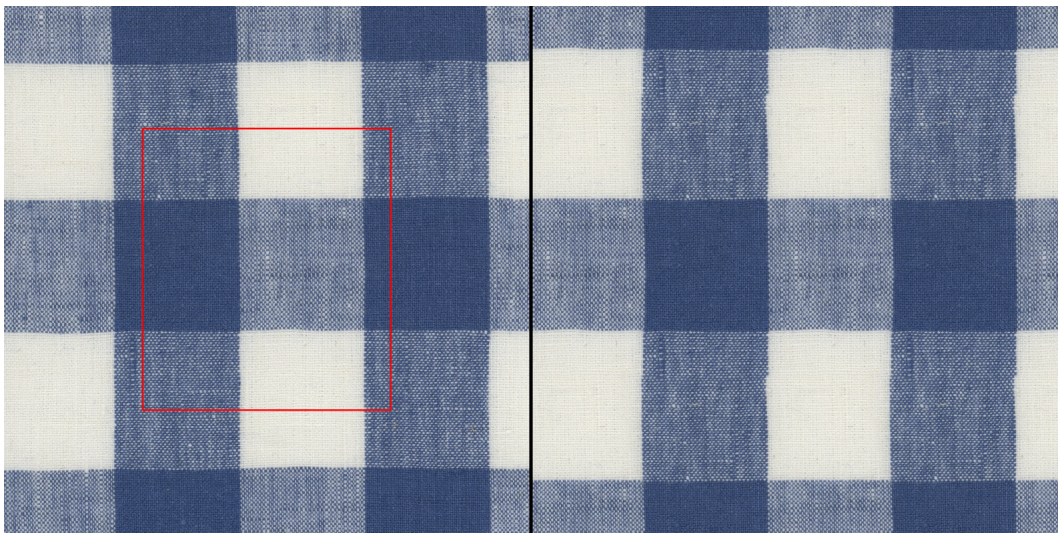


Figure 4.18: MS-SSIM=0.878. Incorrect result because of MRT failure.

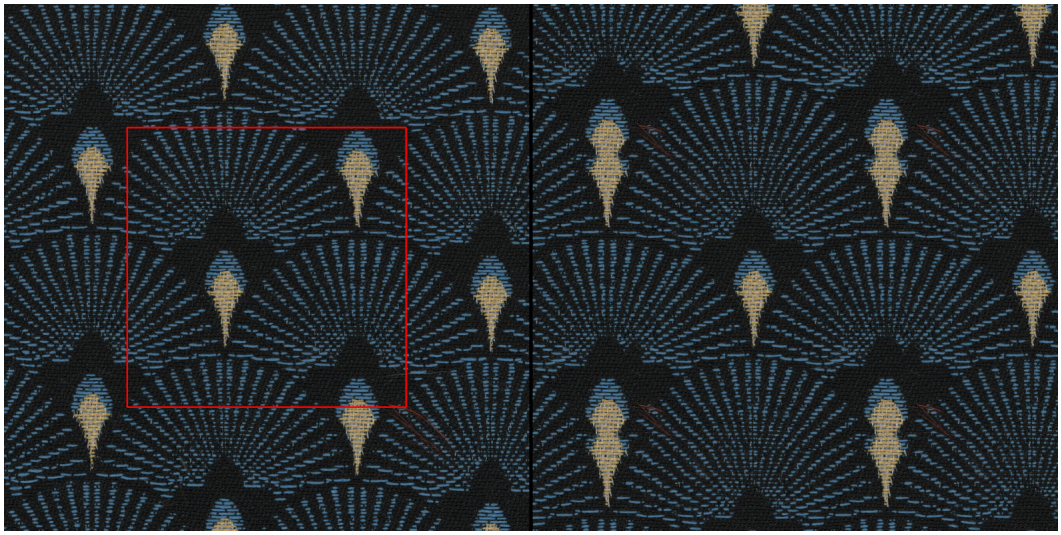


Figure 4.19: MS-SSIM=0.676. Incorrect result due to MRT failure.

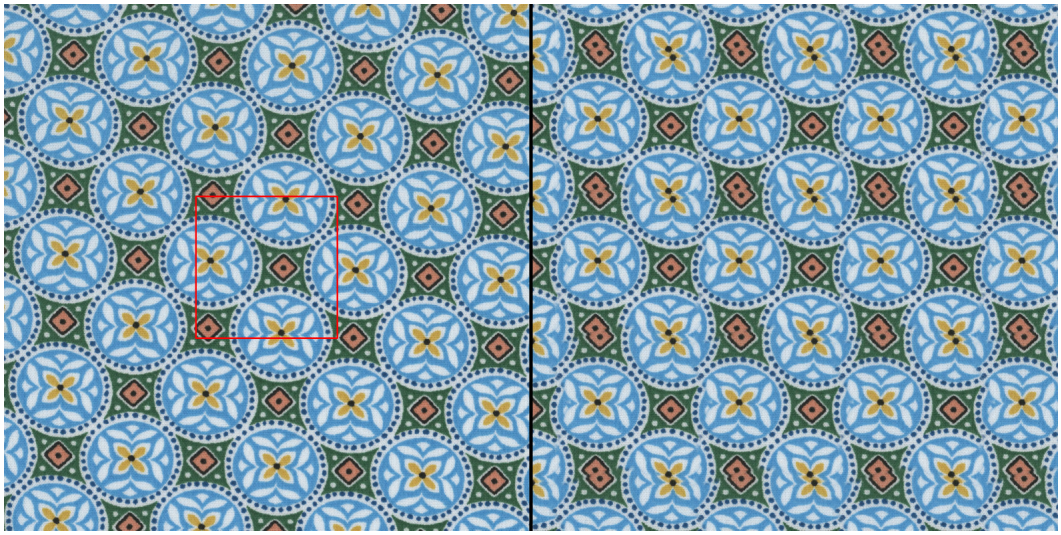


Figure 4.20: MS-SSIM=0.531. Incorrect result due to MRT failure

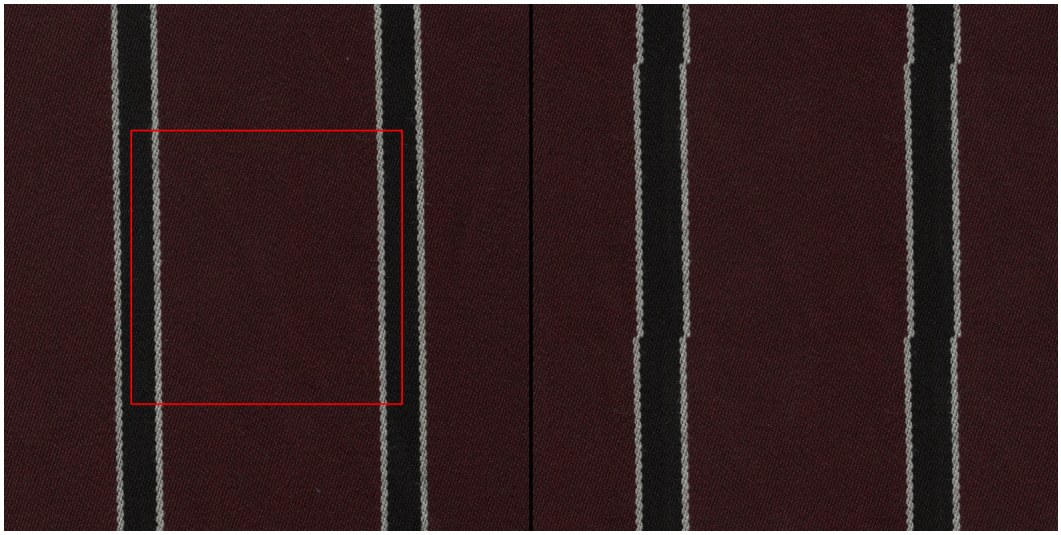


Figure 4.21: MS-SSIM=0.852. Incorrect result due to misalignment of the fabric.

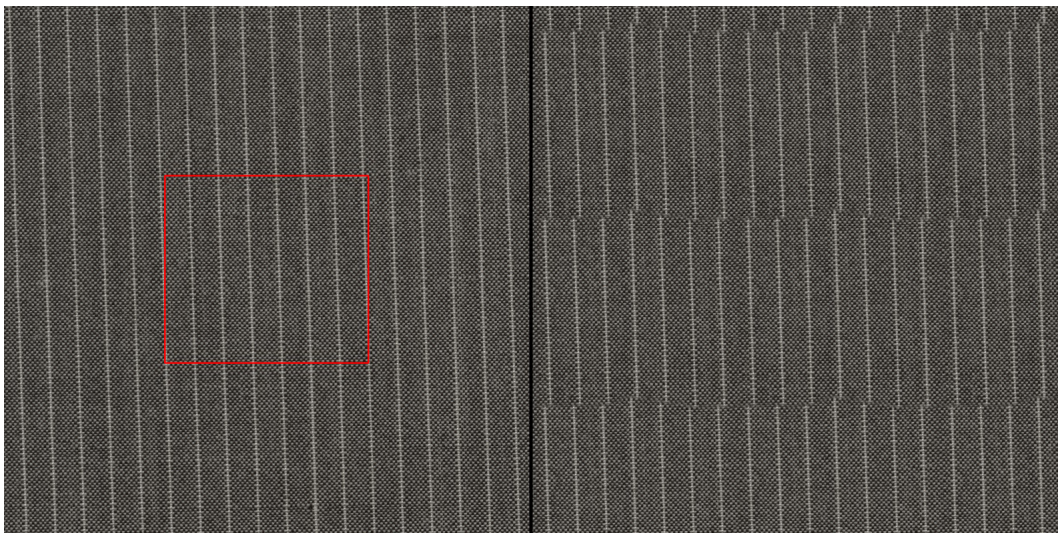


Figure 4.22: MS-SSIM=0.649. Incorrect result due to misalignment of the fabric.

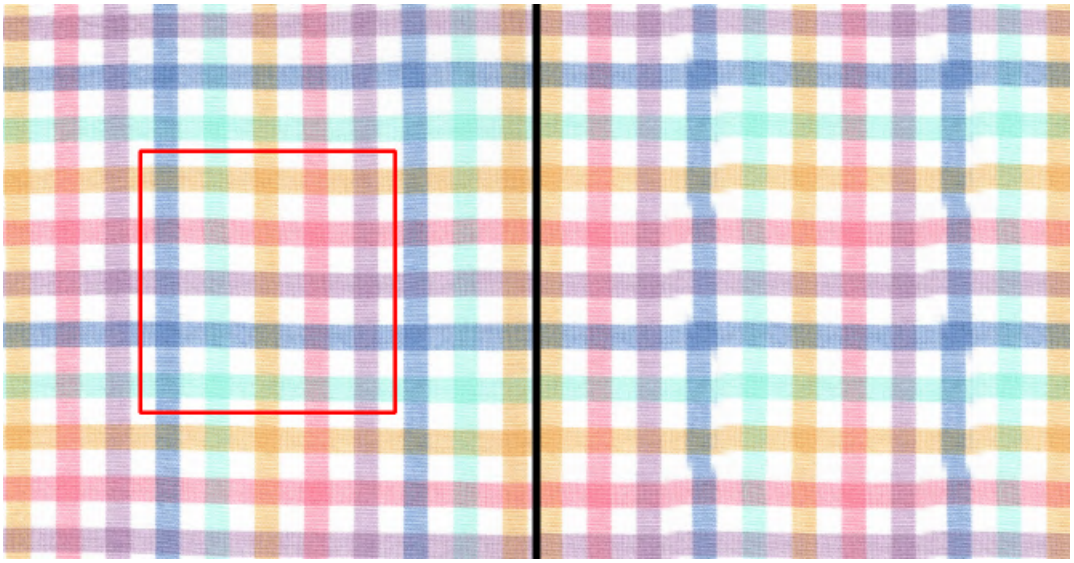


Figure 4.23: MS-SSIM=0.903. Incorrect result due to stitching error.

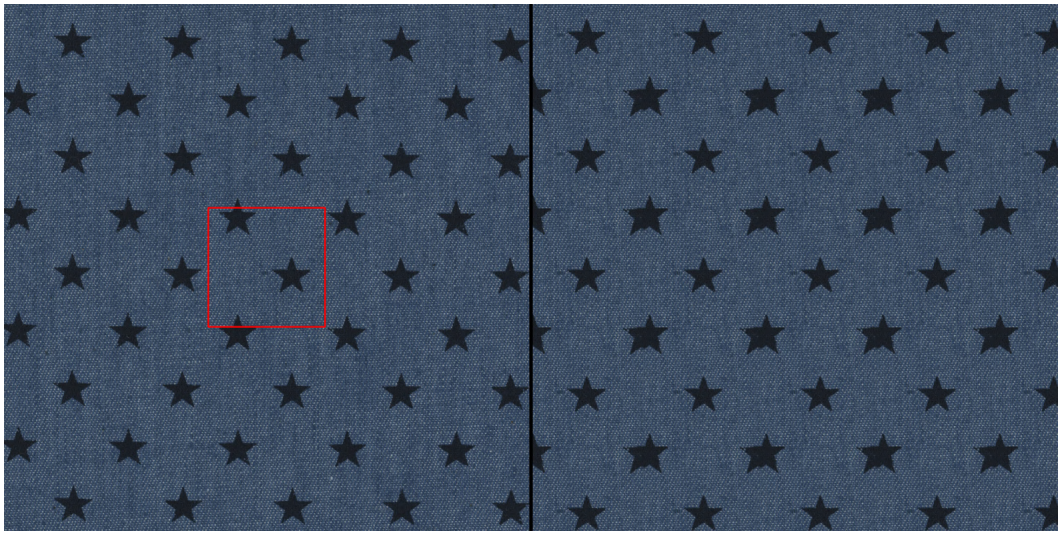


Figure 4.24: MS-SSIM=0.903. Incorrect result due to stitching error.

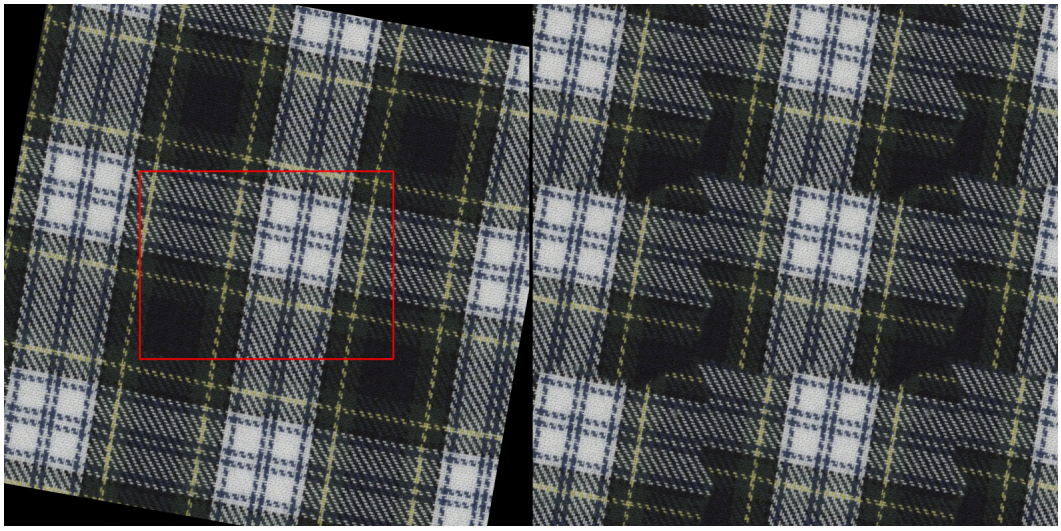


Figure 4.25: MS-SSIM=0.485. Incorrect result due to auto-alignment failure.

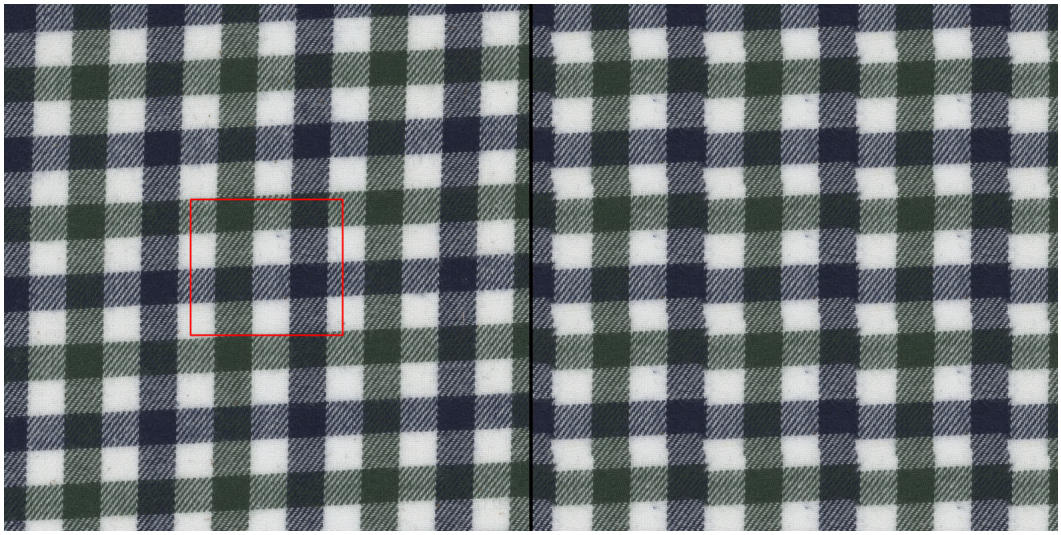


Figure 4.26: MS-SSIM=0.807. Incorrect result due to misalignment of the fabric.

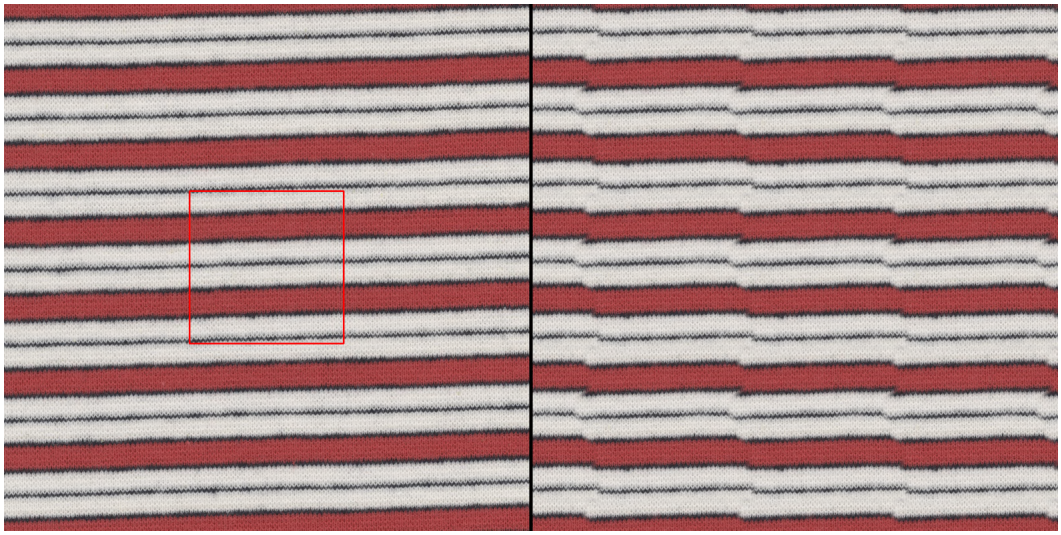


Figure 4.27: MS-SSIM=0.682. Incorrect result due to misalignment of the fabric.

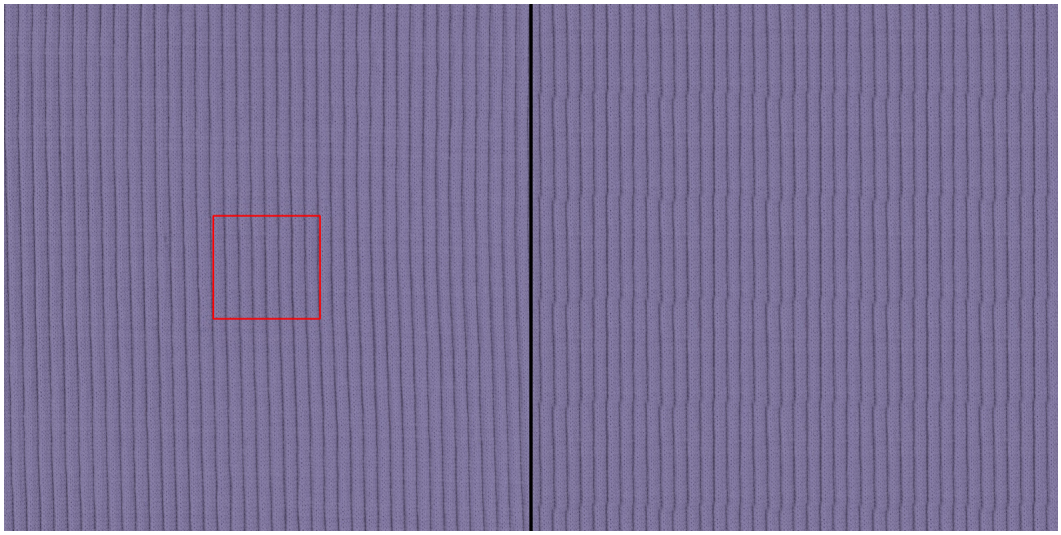


Figure 4.28: MS-SSIM=0.797. Incorrect result due to misalignment of the fabric.

Chapter 5

Auto-tiling demo

This chapter is dedicated to describing the online demo developed using streamlit to showcase our auto-tiling assistant. The primary purpose of this demonstration is to evaluate the effectiveness and feasibility of the final tool as a product. This evaluation will enable the department in charge of implementing the tool in the final product to analyze and identify potential issues or areas for improvement. Additionally, the testing process will help ensure that the tool meets the necessary requirements and specifications to effectively address the intended use case.

This application was developed using the streamlit framework, which is a free and open-source tool for quickly creating and sharing machine learning and data science web applications. By using a Docker container, the application and its dependencies can be easily packaged and deployed on various systems without requiring manual installation and configuration of dependencies. These approach enables users to run the application with ease, even without technical expertise. The user-friendly interface provided by streamlit makes it easy for users to interact with the application, while the back-end server processes the data submitted by the user using the appropriate algorithms and models. This results are presented in an easily understandable and readable format.

The tool offers two different working modes:

- **Manual-tiling**: You have the flexibility to manually select the rotation angle to correct the image in step 2 and the specific area of the image that you want to tile in step 3. The tool provides automatic assistance for both of these steps to simplify the process.

- **Auto-tiling**: The app automatically detects the rotation angle and the optimal

area of the image to tile, offering a quick and efficient solution. After the final result is displayed, you can revisit the previous steps (2 and 3) to examine the outcome of each stage of the algorithm and make any necessary modifications to achieve the desired outcome.

To start using the application, the initial step is to upload an image on the main page. After uploading the image, the user can select the desired mode of operation for the application.

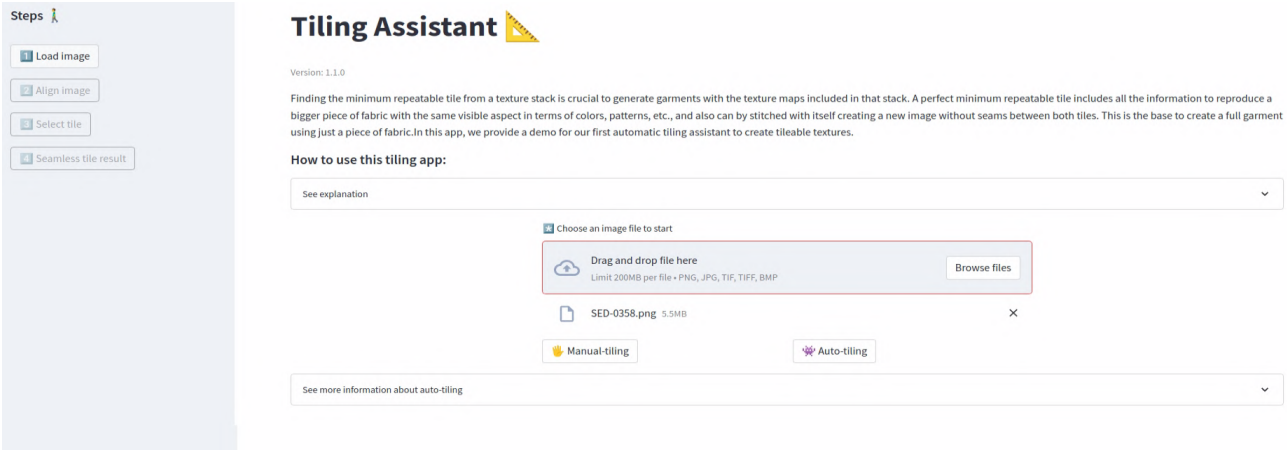


Figure 5.1: Homepage of auto-tiling application. An image has been uploaded.

5.1 Manual-tiling

After uploading the image, we can proceed to the second step of the application by clicking on the button to activate the manual mode for performing rotation correction.

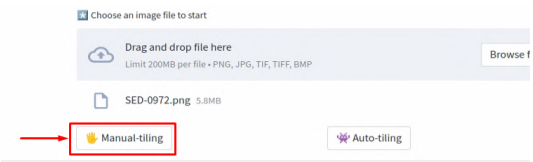
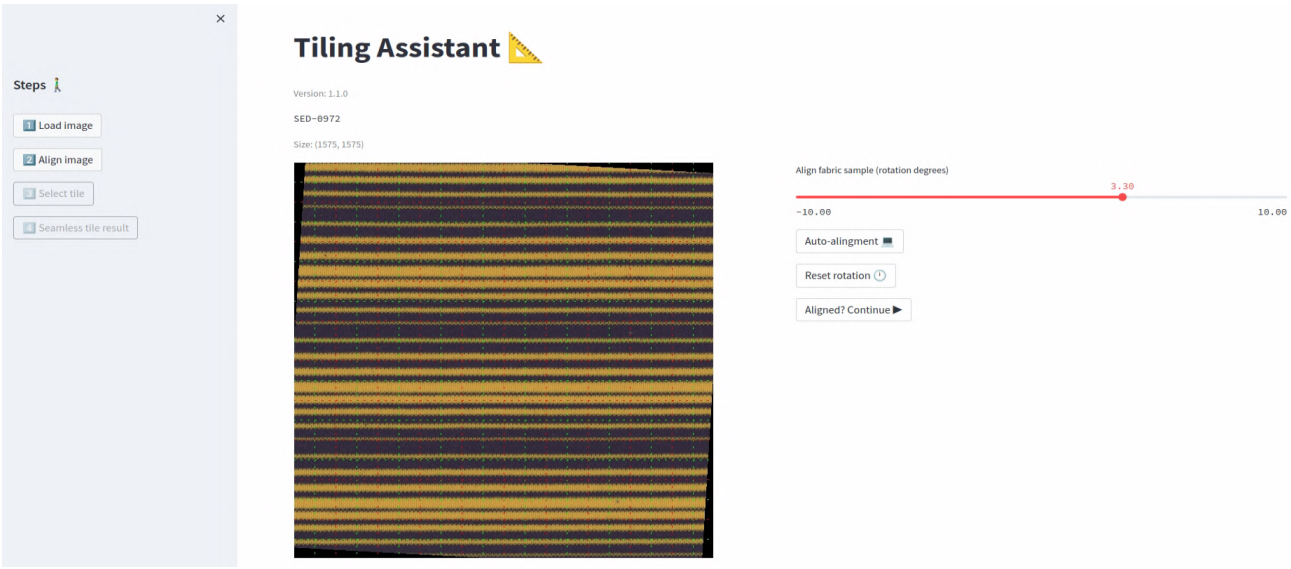


Figure 5.2: Homepage of auto-tiling application. An image has been uploaded.

In this step, we are presented with the original image view and a slider to adjust the rotation correction and preview the updated image. Additionally, there is an automatic search button available for performing the correction automatically.



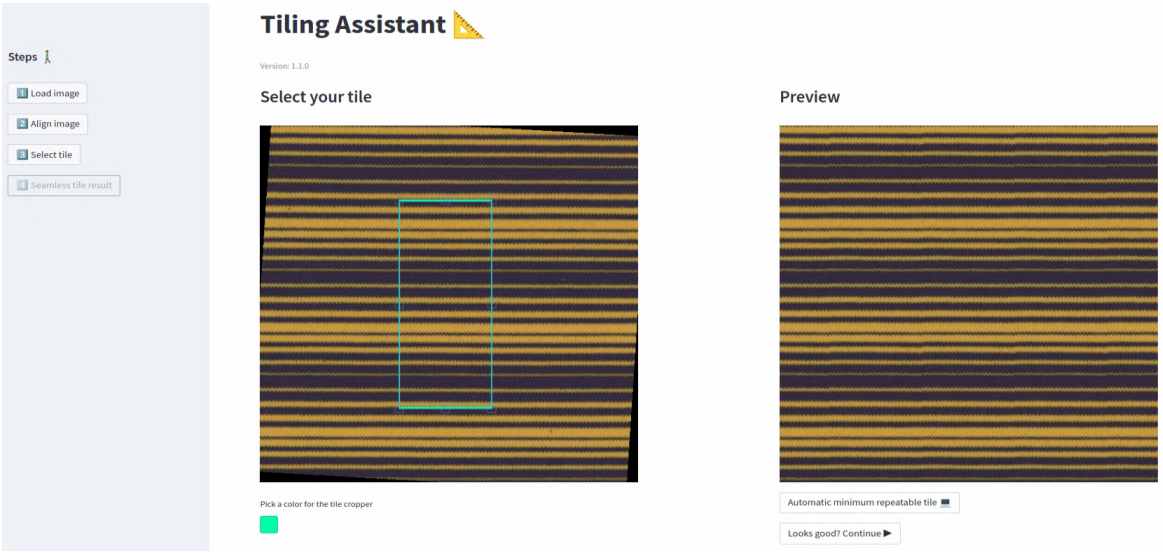
(a) Select manually the angle rotation using the slider.

(b) Detect automatically the angle rotation.

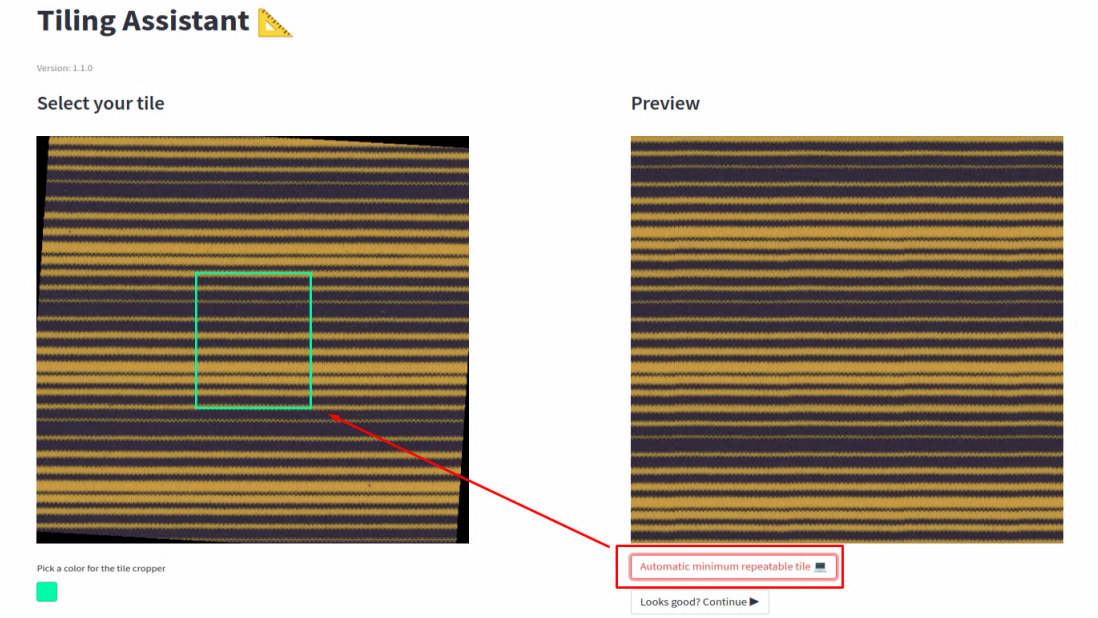
Figure 5.3: Step 2: Manual and automatic angle rotation correction.

After obtaining the accurate angle, we can proceed by clicking the ”continue” button. Moving on to step 3, we will now select either the minimum repeatable pattern cutout or any other cutout that we find suitable for creating the repetition. We also have an automatic option available to perform the minimum repeatable pattern. On the left side of the image, we have a box representing the the selected crop of minimum repeatable pattern. We can modify it as necessary, and on the right side of the image, we can see

the resulting tile in real-time.



(a) Select manually the best repeatable pattern.



(b) Detect automatically detect the best repeatable pattern.

Figure 5.4: Step 3: Manual and automatic minimum repeatable tile detection.

After obtaining the accurate the best repeatable pattern, we can proceed by clicking the ”continue” button. The next and final step is Step 4, where the application displays the results of the best repeatable pattern and the resulting tiled image after going through the stitching algorithm. We can make modifications by going back to the previous steps, and there is also an option to download the resulting image.

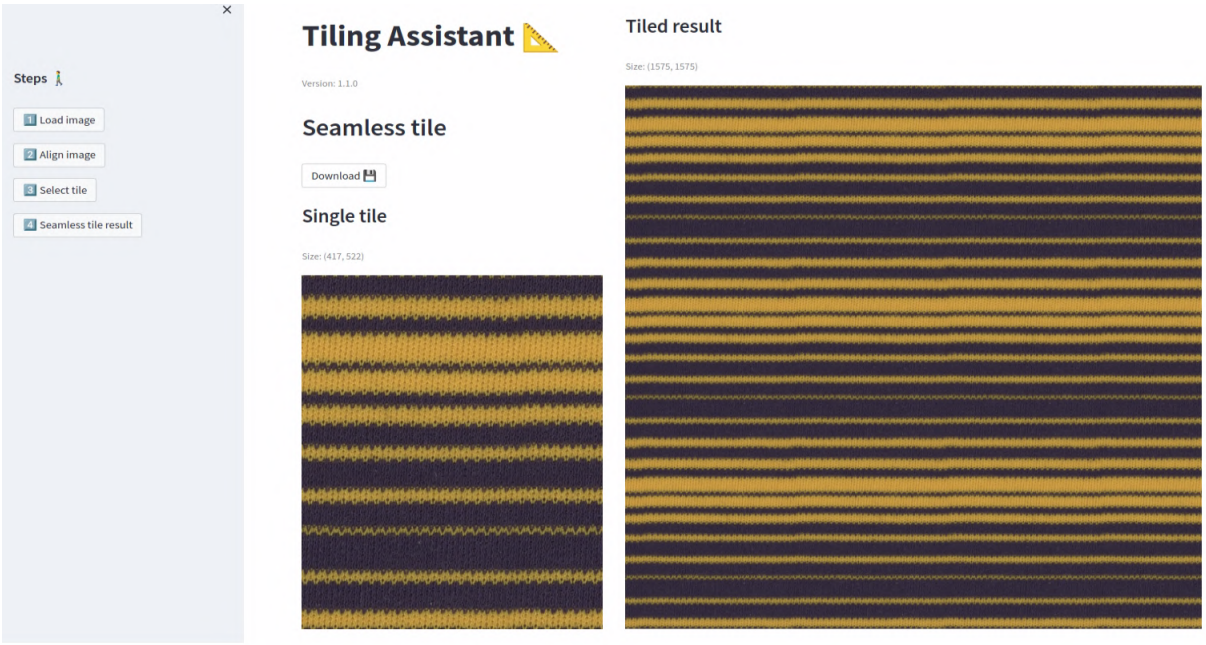


Figure 5.5: Result of manual tiling.

5.2 Auto-tiling

After the image is uploaded, we can obtain the final result by clicking on the button to activate the automatic mode.

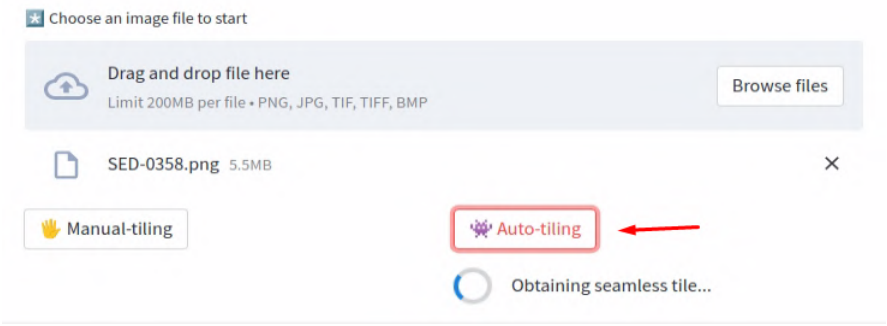


Figure 5.6: Execution of automatic tiling.

Figure 5.7: The left image displays the detected best repeatable pattern while the right image shows the resulting pattern after tiling.

We can make modifications by going back to the previous steps, and there is also an option to download the resulting image.

5.3 Video examples

In this subsection, we provide links to video examples that showcase the functionality of the tool in action. These videos serve as visual demonstrations of how the tool works and highlight its features and capabilities. Whether it's demonstrating the tool's accuracy, speed, or user-friendly interface, these video examples serve as valuable resources for users to explore and evaluate the tool before utilizing it in their own projects or workflows.

- Manual-tiling: `https://youtu.be/CEO48BHs54I`
- Auto-tiling: `https://youtu.be/nnrYfhDhZmY`
- Auto-tiling with alignment correction: `https://youtu.be/jeCHdvwQWDM`

Chapter 6

Conclusions and future lines

In order to do this work, a review of the state of the art has been carried out, and SEDDI's previous work in this area has been extended [18]. The algorithm has been implemented in an optimised way and prepared for potential migration to the front-end by making different tests with frameworks and processing units.

The auto-tiling algorithm, featuring a minimum repeatable tile as its main contribution, presents promising outcomes for real-world use. The algorithm has proven to be successful in tiling images with a minimum repeatable tile and achieving high MS-SSIM scores, which signifies a considerable similarity to the original image. Additionally, this process is executed quickly, taking only around 1.5 seconds, making it a viable recommendation for users to discover the MRT.

Furthermore, it is important to note that the algorithm's performance is heavily dependent on the conditions and alignment of the input images and must have more than two repetition patterns. In cases where the fabric is correctly aligned, the minimum repeatable tile can be found. However, it is important to note that the MS-SSIM score does not guarantee a correct tiling result in all cases. There may be situations where the images appear similar but the tiling is incorrect.

Additionally, we developed and deployed an online demo using Streamlit and Docker for our product department to test the algorithm. This demo allows for both automatic and manual tiling, and the results can be reviewed and modified at each step of the algorithm.

Overall, the auto-tiling algorithm with a minimum repeatable tile has demonstrated significant potential for real-world applications in the context of SEDDI Textura, an AI-powered fabric texture digitization tool equipped with numerous tiling features.

In future work, it would be useful to explore additional improvements to the algorithm's performance, such as integrating more robust and efficient alignment capabilities to expand its applicability to a wide range of scenarios. Additionally, re-training the AlexNET network using fabric images, as demonstrated in this paper [19], could improve feature extraction and enhance the overall performance of the algorithm. Using only the first layers of the CNN network can also be helpful, and it could be faster, as they contain important information about the borders and shapes. Futhermore, another path of study would be diffusion models, to try to obtain a tileable image in the output as these authors do with GANs [23].

Bibliography

[1] Kirsi Niinimäki et al. “The environmental price of fast fashion”. In: *Nature Reviews Earth & Environment* 1 (Apr. 2020), pp. 189–200. DOI: 10.1038/s43017-020-0039-9.

[2] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature Visualization”. In: *Distill* (2017). https://distill.pub/2017/feature-visualization. DOI: 10.23915/distill.00007.

[3] Zhou Wang et al. “Image Quality Assessment: From Error Visibility to Structural Similarity”. In: *Image Processing, IEEE Transactions on* 13 (May 2004), pp. 600 –612. DOI: 10.1109/TIP.2003.819861.

[4] Richard Zhang et al. “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric”. In: (Jan. 2018).

[5] Keyan Ding et al. “Image Quality Assessment: Unifying Structure and Texture Similarity”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (Dec. 2020), pp. 1–1. DOI: 10.1109/TPAMI.2020.3045810.

[6] Steven W. Zucker and Demetri Terzopoulos. “Finding structure in Co-occurrence matrices for texture analysis”. In: *Computer Graphics and Image Processing* 12.3 (1980), pp. 286–308. ISSN: 0146-664X. DOI: https://doi.org/10.1016/0146-664X(80)90016-7. URL: https://www.sciencedirect.com/science/article/pii/0146664X80900167.

[7] J. Parkkinen, K. Selkäinaho, and Erkki Oja. “Detecting texture periodicity from the cooccurrence matrix”. In: *Pattern Recognition Letters* 11 (Jan. 1990), pp. 43–50. DOI: 10.1016/0167-8655(90)90054-6.

[8] Hsin-Chih Lin, Ling Wang, and Shi-Nine Yang. “Extracting periodicity of a regular texture based on autocorrelation functions”. In: *Pattern Recognition Letters* 18 (May 1997), pp. 433–443. DOI: 10.1016/S0167-8655(97)00030-5.

[9] Gyuhwan Oh, Seungyong Lee, and Sung Shin. “Fast determination of textural periodicity using distance matching function”. In: *Pattern Recognition Letters* 20 (Feb. 1999), pp. 191–197. DOI: 10.1016/S0167-8655(98)00140-8.

[10] Michael Unser. “Sum and Difference Histograms for Texture Classification”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8 (Feb. 1986), pp. 118 –125. DOI: 10.1109/TPAMI.1986.4767760.

[11] Takashi Matsuyama, Shu-Ichi Miura, and Makoto Nagao. “Structural analysis of natural textures by Fourier transformation”. In: *Computer Vision, Graphics, and Image Processing* 24 (Sept. 1983), pp. 347–362. DOI: 10.1016/0734-189X(83)90060-9.

[12] Jingchen Liu and Yanxi Liu. “GRASP recurring patterns from a single view”. In: June 2013. DOI: 10.1109/CVPR.2013.261.

[13] James Pritts, Ondrej Chum, and Jiri Matas. “Rectification, and Segmentation of Coplanar Repeated Patterns”. In: June 2014, pp. 2973–2980. DOI: 10.1109/CVPR.2014.380.

[14] Akihiko Torii et al. “Visual Place Recognition with Repetitive Structures”. In: vol. 37. June 2013, pp. 883–890. DOI: 10.1109/CVPR.2013.119.

[15] Tony Lindeberg. “Scale Invariant Feature Transform”. In: vol. 7. May 2012. DOI: 10.4249/scholarpedia.10491.

[16] Jeffrey Kuo, Chung-Yang Shih, and Jiunn-Yih Lee. “Repeat Pattern Segmentation of Printed Fabrics by Hough Transform Method”. In: *Textile Research Journal* 75 (Nov. 2005), pp. 779–783. DOI: 10.1177/0040517505058848.

[17] Louis Lettry et al. “Repeated Pattern Detection Using CNN Activations”. In: Mar. 2017, pp. 47–55. DOI: 10.1109/WACV.2017.13.

[18] Carlos Rodríguez Pardo et al. “Automatic Extraction and Synthesis of Regular Repeatable Patterns”. In: *Computers & Graphics* 83 (Oct. 2019), pp. 33–41. DOI: 10.1016/j.cag.2019.06.010.

[19] Zhong Xiang et al. “Periodic Pattern Detection of Printed Fabric Based on Deep Learning Algorithm”. In: *Journal of Physics: Conference Series* 2148 (Jan. 2022), p. 012013. DOI: 10.1088/1742-6596/2148/1/012013.

[20] Kourosh Jafari-Khouzani and Hamid Soltanian-Zadeh. “Radon Transform Orientation Estimation for Rotation Invariant Texture Analysis”. In: *IEEE transactions on pattern analysis and machine intelligence* 27 (July 2005), pp. 1004–8. DOI: 10.1109/TPAMI.2005.126.

[21] Shai Avidan and Ariel Shamir. "Seam Carving for Content-Aware Image Resizing". In: *SIGGRAPH* 26 (July 2007). DOI: 10.1145/1276377.1276390.

[22] Michael Rubinstein, Ariel Shamir, and Shai Avidan. "Improved seam carving for video retargeting". In: *ACM Trans. Graph.* 27 (Aug. 2008). DOI: 10.1145/1399504.1360615.

[23] Carlos Rodriguez-Pardo and Elena Garces. "SeamlessGAN: Self-Supervised Synthesis of Tileable Texture Maps". In: *IEEE Transactions on Visualization and Computer Graphics* (2022).