Lab 1 Report: Fabric Anomaly Detection Using Autoencoder

Abstract: This report elucidates a method implemented for detecting anomalies in fabric images utilizing an autoencoder model, a specialized form of a neural network known for its adeptness in handling reconstruction tasks. Various loss functions, namely Mean Squared Error (MSE), Structural Similarity Index (SSIM), and Multi-Scale Structural Similarity (MS-SSIM), were evaluated to gauge the efficacy of the model in discerning normal and anomalous fabric patterns. Preliminary results indicated that, of the loss functions assessed, MSE exhibited a promising outcome in accurately identifying diverse fabric anomalies. The application of MSE enabled the autoencoder to detect anomalies based on the deviation in reconstruction error effectively, thus proving its viability as a robust metric for fabric anomaly detection. The findings from this investigation hold substantial implications in advancing scalable and accurate anomaly detection mechanisms within the domain of fabric manufacturing, potentially mitigating associated financial ramifications and ensuring product quality.

1. Introduction:

Anomalies in fabric manufacturing can result in financial losses and customer dissatisfaction. Traditional methods, such as manual inspection, have their limitations in terms of accuracy, speed, and scalability. Hence, a machine learning-based approach can provide an efficient and reliable solution.

2. Training Procedure:

2.1 Data Collection:

The experiment employs a custom dataset composed of images labeled either as ‘fault’ or ‘non_fault’. The dataset is partitioned into training and testing subsets, with images allocated to each subset in a manner preserving the distribution of the different categories of images within the overall dataset.

	Training set	Testing set
fault		8
non_fault	196	20

2.2 Data Pre-processing:
Each image in the dataset undergoes a preprocessing phase where it is resized to 256x256 pixels and transformed to tensor format, utilizing the **transforms** module from **torchvision**. This process ensures uniformity in input data dimensions, allowing for seamless training and testing of the neural network model.

2.3 Model Architecture:
The Autoencoder model used in this experiment consists of an encoder and a decoder. The encoder compresses the input image into a lower-dimensional latent space representation, while the decoder reconstructs the original image from this representation. Autoencoders are integral for anomaly detection as they are trained to learn the normal data distribution and can thus differentiate anomalies based on reconstruction errors.

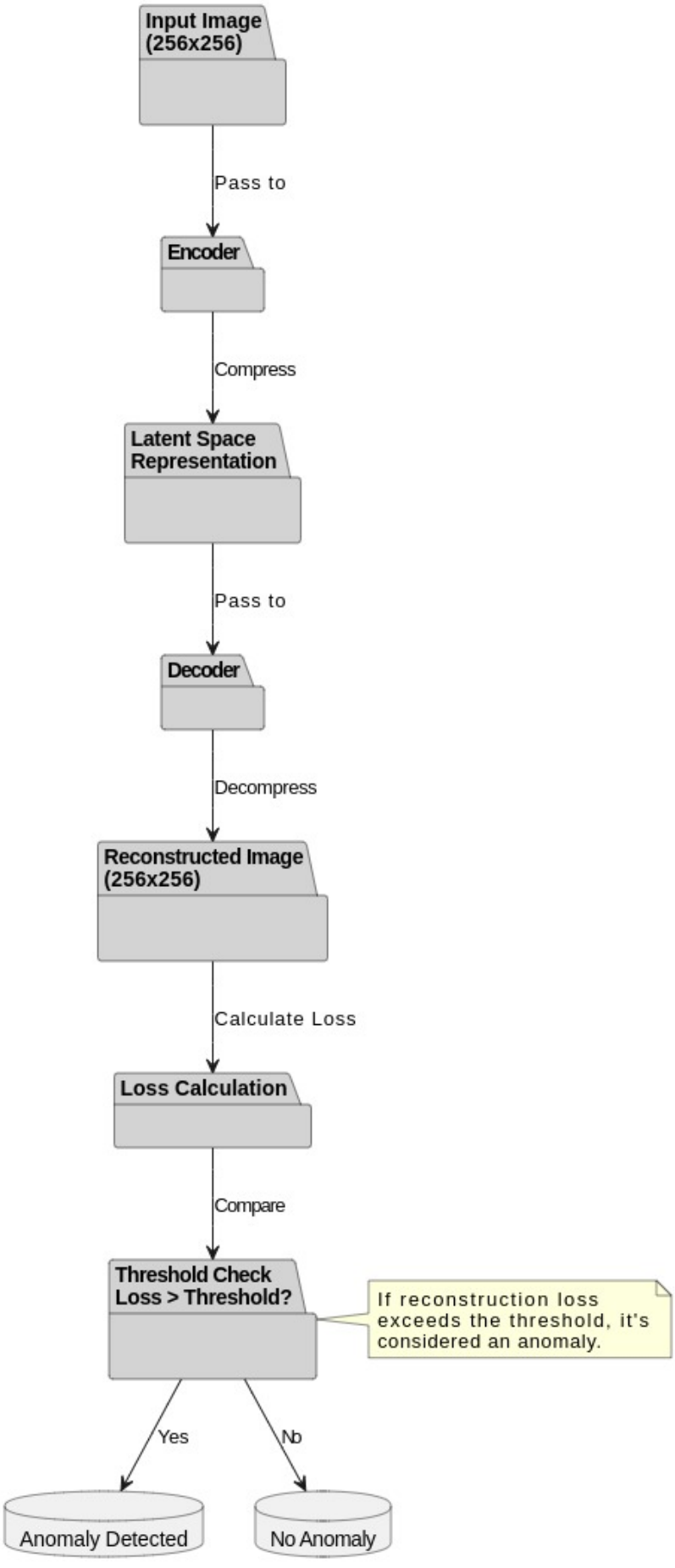
2.4 Hyperparameters
The model training involves a set of predefined hyperparameters including a batch size of 8, 50 training epochs, and a learning rate of 0.001. These parameters were empirically chosen to optimize the learning process.

2.5 Optimization and Loss Functions
Adam optimizer is employed to minimize the chosen loss functions during training. The model is separately trained using three different loss functions: MSE for measuring the mean squared error between the reconstructed and original images, SSIM for assessing the structural similarity, and MS-SSIM for evaluating multi-scale structural similarity between the original and reconstructed images. These loss functions were selected to provide a comprehensive understanding of the model's performance from different perspectives.

3. Testing Procedure:

3.1 Heatmap Generation
Post-training, the model is evaluated on the test dataset, and anomaly localization heatmaps are generated. The residual image is computed by subtracting the reconstructed image from the original input, and the heatmap is derived from this residual by summing the absolute values along the color channels, followed by application of a colormap. Each heatmap is saved with the corresponding input image name.

3.2 Anomaly Detection and Results Documentation
The reconstruction loss is calculated for each test image to determine the anomaly status based on a predetermined threshold. The filename, loss, anomaly status, and category of each test image are documented in a result text file, enabling further analysis of the model's performance.


Input Image (256x256)
Pass to
Encoder
Compress
Latent Space Representation
Pass to
Decoder
Decompress
Reconstructed Image (256x256)
Calculate Loss
Loss Calculation
Compare
Threshold Check Loss > Threshold?
If reconstruction loss exceeds the threshold, it's considered an anomaly.
Yes
No
Anomaly Detected
No Anomaly

4. Results:

	Shallow Autoencoder			Shallow-Neutral Autoencoder			Neutral Autoencoder			Deep Autoencoder		
	MSE	SSIM	MS-SSIM	MSE	SSIM	MS-SSIM	MSE	SSIM	MS-SSIM	MSE	SSIM	MS-SSIM
fault_1	0.0023	0.0001	0	0.0033	0.0002	0.0001	0.0039	0.0001	0	0.0048	0.0055	0.0004
fault_2	0.0025	0.0001	0	0.0034	0.0002	0.0001	0.0043	0.0001	0	0.006	0.0061	0.0005
fault_3	0.0025	0.0001	0	0.0034	0.0002	0.0001	0.0044	0.0001	0	0.0059	0.0059	0.0005
fault_4	0.0024	0.0001	0	0.0035	0.0002	0.0001	0.0042	0.0001	0	0.005	0.0057	0.0005
fault_5	0.0022	0.0001	0	0.003	0.0001	0.0001	0.0037	0.0001	0	0.0046	0.0055	0.0004
fault_6	0.0022	0.0001	0	0.0031	0.0001	0.0001	0.0037	0.0001	0	0.0046	0.0056	0.0005
fault_7	0.0021	0.0001	0	0.003	0.0001	0.0001	0.0036	0.0001	0	0.0044	0.0055	0.0004
fault_8	0.0023	0.0001	0	0.0031	0.0001	0.0001	0.0037	0.0001	0	0.0047	0.0057	0.0004
non-fault_1	0.0025	0.0001	0	0.0031	0.0002	0.0001	0.0038	0.0001	0	0.0043	0.0057	0.0004
non-fault_2	0.0024	0.0001	0	0.0031	0.0002	0.0001	0.0037	0.0001	0	0.0044	0.0061	0.0005
non-fault_3	0.0023	0.0001	0	0.003	0.0001	0.0001	0.0036	0.0001	0	0.0041	0.0056	0.0004
non-fault_4	0.0024	0.0001	0	0.0031	0.0002	0.0001	0.0036	0.0001	0	0.0045	0.0061	0.0005
non-fault_5	0.0024	0.0001	0	0.0031	0.0002	0.0001	0.0035	0.0001	0	0.0044	0.0057	0.0004
non-fault_6	0.0022	0.0001	0	0.0028	0.0001	0.0001	0.0032	0.0001	0	0.0041	0.0055	0.0004
non-fault_7	0.0023	0.0001	0	0.0031	0.0002	0.0001	0.0036	0.0001	0	0.0043	0.0056	0.0004
non-fault_8	0.0022	0.0001	0	0.003	0.0001	0.0001	0.0034	0.0001	0	0.0044	0.0054	0.0004
non-fault_9	0.0022	0.0001	0	0.0029	0.0001	0.0001	0.0035	0.0001	0	0.0042	0.0055	0.0004
non-fault_10	0.0024	0.0001	0	0.0032	0.0002	0.0001	0.0037	0.0001	0	0.0046	0.0059	0.0004
non-fault_11	0.0022	0.0001	0	0.0028	0.0001	0.0001	0.0032	0.0001	0	0.004	0.0055	0.0004
non-fault_12	0.0023	0.0001	0	0.0031	0.0002	0.0001	0.0037	0.0001	0	0.0046	0.0058	0.0005
non-fault_13	0.0022	0.0001	0	0.003	0.0002	0.0001	0.0035	0.0001	0	0.0044	0.0059	0.0004
non-fault_14	0.0021	0.0001	0	0.0029	0.0002	0.0001	0.0035	0.0001	0	0.0042	0.0058	0.0005
non-fault_15	0.0021	0.0001	0	0.0029	0.0001	0.0001	0.0034	0.0001	0	0.0042	0.0056	0.0004
non-fault_16	0.0022	0.0001	0	0.003	0.0001	0.0001	0.0035	0.0001	0	0.0044	0.0058	0.0004
non-fault_17	0.0022	0.0001	0	0.0028	0.0002	0.0001	0.0034	0.0001	0	0.0046	0.0058	0.0004
non-fault_18	0.0024	0.0001	0	0.003	0.0002	0.0001	0.0036	0.0001	0	0.0046	0.0062	0.0005
non-fault_19	0.0023	0.0001	0	0.0029	0.0002	0.0001	0.0036	0.0001	0	0.0045	0.006	0.0005
non-fault_20	0.0024	0.0001	0	0.003	0.0002	0.0001	0.0036	0.0001	0	0.0047	0.0063	0.0005

Condensed Results

This study explored the efficacy of several Autoencoder models, including Shallow-Neutral, Neutral, and Deep Autoencoders, using Mean Squared Error (MSE) for fabric anomaly detection. All models, except for the non-performing Shallow Autoencoder, reliably identified around 50% of the faulty images without misclassifying the non-fault ones.

5. Future Plan:

Given the promising yet moderate success of the autoencoder models in anomaly detection within fabric images, the next steps will focus on addressing the limitations encountered in the initial phases, particularly the influence of the size of the training data on model performance. One promising avenue to overcome this limitation is the exploration and implementation of few-shot image generation techniques.

Few-Shot Image Generation:

Few-shot image generation can potentially alleviate the constraints imposed by limited training data. This technique aims to generate new, diverse, and high-quality samples from a small set of initial images, enabling the model to learn more generalized and robust features from limited examples. By leveraging few-shot learning, it is anticipated that the autoencoder models can be trained more effectively, enhancing their ability to distinguish anomalies with higher accuracy and reliability.

Exploration of Advanced Models:

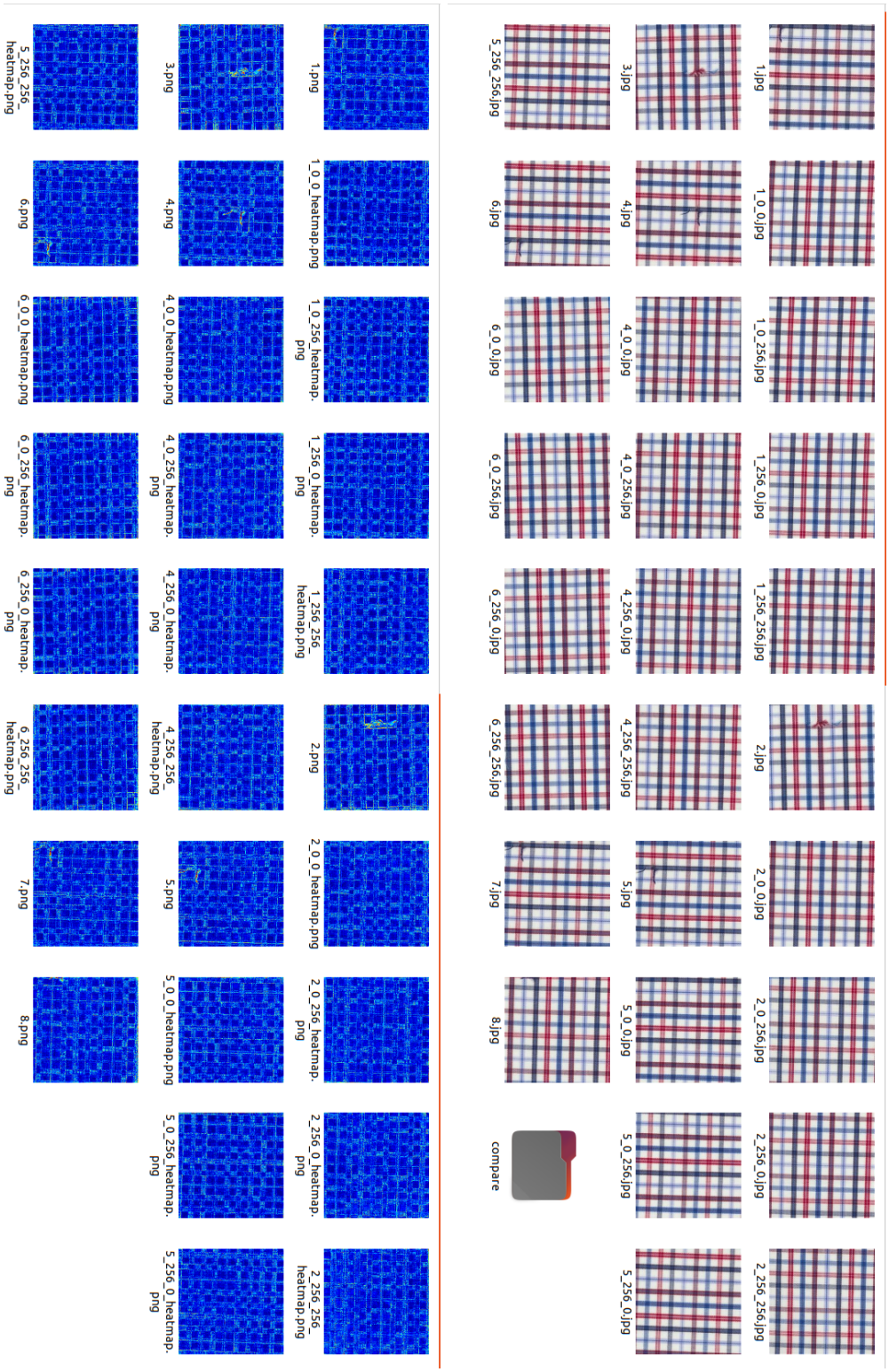
Beyond few-shot image generation and autoencoders, further exploration will also encompass advanced models and techniques in machine learning and computer vision, such as Generative Adversarial Networks (GANs) and Transformer-based models, to investigate their viability and effectiveness in fabric anomaly detection.

Assessment and Validation:

Subsequent to the implementation of the aforementioned strategies, rigorous evaluation and validation will be undertaken to ensure the robustness and reliability of the developed models. The comparative analysis will be performed between the refined models and the initial autoencoder models to quantify the improvements achieved.

By aligning the future plan with the integration of few-shot image generation and exploring advanced models, the goal is to push the boundaries of the current state-of-the-art in fabric anomaly detection, contributing to the evolution of quality assurance mechanisms in the textile industry.

6. Miscellaneous



Images Input and Loss Heatmap

```
""" import torch.nn as nn

class Autoencoder(nn.Module):
def __init__(self):
super(Autoencoder, self).__init__()

# Encoder
self.encoder = nn.Sequential(
nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.Conv2d(512, 1024, kernel_size=4, stride=2, padding=1),
nn.ReLU()
)

# Decoder
self.decoder = nn.Sequential(
nn.ConvTranspose2d(1024, 512, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1),
nn.Sigmoid()
)

def forward(self, x):
x = self.encoder(x)
x = self.decoder(x)
return x """

import torch.nn as nn
```

```
class Autoencoder(nn.Module):
def __init__(self):
super(Autoencoder, self).__init__()
# Encoder
self.encoder = nn.Sequential(
nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
)
# Decoder
self.decoder = nn.Sequential(
nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
nn.ReLU(),
nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1),
nn.Sigmoid()
)
def forward(self, x):
x = self.encoder(x)
x = self.decoder(x)
return x
```

Model Code Examples