ANOMALY DETECTION FOR MANUFACTURING APPLICATIONS USING CONVOLUTIONAL AUTOENCODERS

by

Nourelislam Zouar

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2021

Approved by:

_____
Dr. Min Shin

_____
Dr. Stephen Welch

_____
Dr. Chen Chen

ABSTRACT

NOURELISLAM ZOUAR. ANOMALY DETECTION FOR MANUFACTURING APPLICATIONS USING CONVOLUTIONAL AUTOENCODERS. (Under the direction of DR. MIN SHIN)

Anomaly Detection in manufacturing environments is increasingly gaining popularity among companies and researchers. Computer based visual inspection systems are at the core of this interest. In the last few years, computer vision has made immense advancement thanks to deep learning. More specifically, unsupervised learning has proven its strength in this area, mainly due to its ability to deal with all kinds of anomalies and this is due to the philosophy used in this type of machine learning.

This research work was motivated by the availability of what is arguably the most comprehensive public anomaly detection dataset, the MVTec dataset [1]. We implemented a pipeline that starts by preprocessing different categories of the MVTec dataset and ends by calculating the prediction accuracies and the inference times. This pipeline includes a convolutional autoencoder. We started by implementing a CAE that follows the description provided by [1]. Then, we have run many experiments using different CAE architectures found in recently published papers.

In order to evaluate the performance of all the experimented CAEs, we used a brute force logic to find the best threshold. We used several portions of the calculated accuracies to find the best threshold. These portions were made using different percentages of the calculated accuracies, ranging from 70% to 100%.

DEDICATION

I dedicate my dissertation work to my beloved mother Nacera Guettaf who has always supported me and to my father Hocine Zouar who never saw this adventure.

I dedicate this work to my sisters Amina and Amel, and to my brothers Mohamed and Hichem.

I also dedicate my dissertation to all my elementary, middle school, high school, and university professors who made of me what I am today.

ACKNOWLEDGEMENTS

I thank my committee members, Dr. Min Shin, Dr. Stephen Welch, and Dr. Chen Chen. I also want to thank Dr. Zachary Wartell for his continuous support and assistance.

Special thanks goes to the AMIDEAST-Fulbright scholarship team.

TABLE OF CONTENTS

LIST OF TABLES ..... *viii*

LIST OF FIGURES ..... *ix*

CHAPTER 1: INTRODUCTION *1*

Thesis Statement 2

CHAPTER 2: BACKGROUND *3*

2.1 Unsupervised Learning 3

2.2 Convolutional Autoencoder 4

2.3 Anomaly Detection 7

CHAPTER 3: RELATED WORK *9*

CHAPTER 4: METHODOLOGY *14*

4.1 Building the L2 convolutional autoencoder pipeline 14

4.1.1 Training dataset 16

4.1.2 Test dataset 17

4.1.3 L2 CAE 19

4.1.4 Reconstruction 21

4.1.5 Calculating distances 24

4.1.6 Thresholding 28

4.1.7 Speed of the CAE 29

4.2 Experimenting other architectures 29

4.2.1 Oh and Yun 2018 30

4.2.2 Chow et al. 2020 30

4.2.3 Gong et al. 2019 33

CHAPTER 5: RESULTS & FUTURE WORK *35*

5.1 Future work 35

REFERENCES *37*

APPENDIX A: RESULTS USING 100% OF THE ACCURACIES *40*

APPENDIX B: RESULTS USING 90% OF THE ACCURACIES *41*

APPENDIX C: RESULTS USING 80% OF THE ACCURACIES *42*

LIST OF TABLES

Table 1: Statistical recap of the MVTec dataset. 9

Table 2: Accuracies obtained for object categories. 11

Table 3: Accuracies obtained for texture categories. 12

Table 4: L2 CAE architecture [18]. 20

Table 5: Samples of L2-distances. 26

Table 6: Grid results of experimenting different picking best threshold logics. 29

Table 7: [27] CAE architecture. 31

Table 8: [28] CAE architecture - Encoder. 32

Table 9: [28] CAE architecture - Decoder. 33

Table 10: [29] CAE architecture - Decoder. 34

Table 11: Results of the evaluated CAEs. For each experimented method, and for each category, good products and defective products accuracies are calculated along with the inference time which was measured in milliseconds. The method with the highest mean of the two accuracies is highlighted in boldface for each category. 36

LIST OF FIGURES

Figure 1: Convolving technique..... 5

Figure 2: A simplistic representation of an autoencoder. 6

Figure 3: MVTec example images of all five textures. 10

Figure 4: MVTec example images of all ten objects. 10

Figure 5: Pipeline of the MVTec L2 CAE. 15

Figure 6: Process of building the training dataset. 16

Figure 7: Making patches without overlap. 17

Figure 8: Making patches with overlap. 18

Figure 9: Building the test dataset using striding. 19

Figure 10: An example of a non-overlapping pixel and overlapping pixel. 22

Figure 11: Illustration of the dividing factors idea. 23

Figure 12: Samples of some reconstructions. 24

Figure 13: Euclidean distance formula. 25

Figure 14: L2 distance between two images formula. 25

Figure 15: A sample of a successful distinction between good and defective. 27

Figure 16: A sample of an unsuccessful distinction between good and defective. 27

CHAPTER 1: INTRODUCTION

Manufacturers are increasingly adopting advanced digital technologies, leading to more intelligent manufacturing processes [2]. Smart manufacturing involves the integration of various sensors, computing platforms, communication technologies, and data intensive modelling in the production process [3]. An example of this integration is the application of the Internet of Things (IoT) technologies in manufacturing, commonly known as Industrial IoT or IIoT [4]. Many manufacturers are investing in modern digital technologies such as IoT to gain competitive advantage. This union of manufacturing and advanced technologies benefit manufacturers in many ways, including managing complex systems, improving production performance, and gaining competitive advantages in the international market [5].

Product defects are an inevitable part of modern manufacturing processes [6]. Detecting issues early in production allows manufacturers to deliver higher quality products, reduce waste, and save cost. Clearly, manufacturers always seek to improve quality in order to increase profits, “there is no better cost to eliminate than the cost of poor quality” [7].

Driven by recent success of deep learning in computer vision since the introduction of AlexNet in 2012 [8], automated anomaly detection is increasingly gaining interest in the scientific community. Deep learning made it easy to solve many problems, such as: classification and segmentation, high accuracies have been obtained in various domains [9]. The high performance of deep learning does come at a cost - deep learning often requires large amounts of data and significant computational power, due to the volume of computation required [10]. For example, for image classification tasks, some practitioners

recommend 1000 examples of each class or more. However, this number can go down substantially if a pre-trained model is used [11].

Thesis Statement

In academic environments, researchers and scientists tend to share datasets for the sake of advancing the overall field. Many outstanding accomplishments made in computer vision were enabled by the introduction of large datasets. For example, stellar achievements were made in image classification thanks to MNIST [12], CIFAR10 [13], and ImageNet [14]. Unlike image classification, unsupervised anomaly detection lacks large public real-world datasets. Further, freely accessible comprehensive datasets are very rare in the manufacturing world, primarily due to the competitive nature of manufacturers [15].

A notable exception to this trend is [1], which consisted of two major contributions. First, the introduction of what is arguably the most complete public anomaly detection dataset, the MVTec Anomaly Detection dataset. The availability of such a dataset to the public opened a number of opportunities. Secondly, setting a baseline to which researchers can compare themselves to. This baseline comprises results obtained by running a series of experiments on the newly introduced dataset, including techniques such as AnoGAN, CNN Feature Dictionary, and Autoencoders.

Using the MVTec dataset, the goal of our research is to present a deep learning model that is capable of yielding competitive results relative to [1]. More precisely, we conduct a series of experiments on the MVTec dataset using various Convolutional Autoencoder architectures.

CHAPTER 2: BACKGROUND

2.1 Unsupervised Learning

Machine Learning (ML) is a branch of artificial intelligence that focuses on algorithms capable of learning from data. Traditionally, programmers write code that captures the programmer's understanding of the task at hand. In ML based approaches, the algorithm does note exclusively rely on the programmer's knowledge to obtain a specific outcome. Instead, the programmer writes multiple components of his learning algorithm, such as the model architecture which is responsible for containing the knowledge learnt, however much of the knowledge gained is not explicitly programmed, it is learned from data.

Numerous machine learning approaches exist including: supervised learning, unsupervised learning, or semi-supervised learning. The present work is focused on autoencoders, which are a type of unsupervised machine learning algorithms.

In supervised learning, the model is guided towards learning a specific pattern using data labelled by humans. In contrast, unsupervised learning leverages "raw" unlabeled data, consequently, the learning model must discover patterns from data alone [16]. Clustering is a canonical example of unsupervised learning. For example, say we have a dataset of persons that consists only of two pieces of information: age and weight. More specifically, the data is a set of tuples such as (65, 160), and the gender of these persons is not given. Given sufficient data, a clustering algorithm will discover the hidden pattern and learn the relation that links age and weight to gender. Using the trained model, we will be able to predict the gender of a person only based on his age and weight. Unsupervised learning is also used by autoencoders. The last-mentioned are a type of deep learning

algorithms that outputs a reconstruction of their input. This idea of exactly reconstructing the input has multiple applications, such as: denoising images and anomaly detection.

The idea of leveraging autoencoders for anomaly detection is based on a key feature of autoencoders which is the ability to reconstruct their input. More precisely, the autoencoder is trained using good products images exclusively. Once trained, the model is supposed to always output a good product image. So, if the input is an image of a good product, then the input and its reconstruction should be very similar. In contrast, if the input is an image of a defective product, the reconstructed image which is an image of a good product should be very different from the input. By measuring the similarity between an input image and its reconstruction anomalies are detected.

2.2 Convolutional Autoencoder

Convolutional neural networks (CNNs) are a type of artificial neural network (ANN) broadly used in analyzing images. Like other ANNs, a CNN is a set of layers of artificial neurons that can be trained to detect patterns in images, these patterns range from simple features like edges, shapes, or textures, to complex features composed of many layers of simpler features such as faces, eyes, or specific objects. What distinguishes CNNs from ANNs is the use of convolution instead of general matrix multiplication [17]. The convolution operation consists of applying filters to images, a filter for each pattern at each layer. In fact, the deeper we dive into a CNN the more sophisticated patterns our CNN is capable of detecting. A filter is a relatively small matrix of learnable parameters called weights. In a convolutional layer a filter is slid over the image and a dot product is computed between the filter itself with the corresponding pixels of the image each time, the result of this convolutional layer is a new matrix of the dot products. This sliding

operation is referred to as convolving. Hence, a convolutional autoencoder is nothing more than an autoencoder that involves convolutional layers.

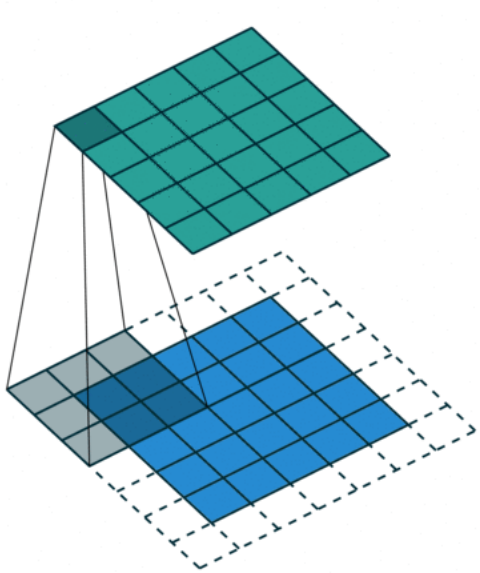

Figure 1: Convolving technique.[1]

Autoencoders are a family of algorithms that fall under the unsupervised learning. These algorithms are meant for dimensionality reduction, the result of this reduction is a representation or an encoding. An autoencoder consists of two parts: an encoder and a decoder. The former is a set of neural network layers, the deeper you go into an encoder the smaller the size of the layer, the last and smallest layer in the encoder is called the bottleneck or the latent space. In the decoder, the shallower you get the greater the size of the layer.

[1] https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

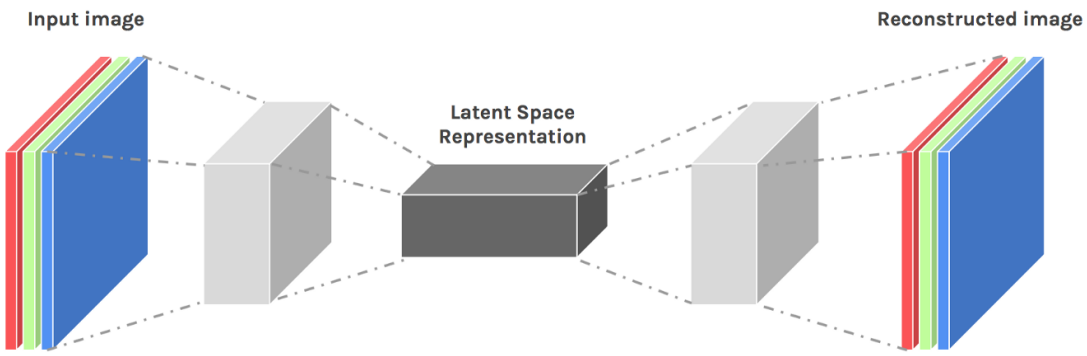

Figure 2: A simplistic representation of an autoencoder.[2]

Autoencoders are characterized by many features such as:

- Data-specific, this means that an autoencoder only works on the data that was trained on. If for example, we trained an autoencoder by using images of bottles, the resulting trained model won't be useful for anything else but images of bottles.
- Although known by outputting their inputs, autoencoders are lossy. Having an output image that looks exactly like the input one is not guaranteed when using autoencoders. Even though these models attempt to reconstruct their input, autoencoders will always output a degraded version of the input.
- As unsupervised models, autoencoders do not need labeled data. Regardless of the preprocessing of the images before inputting them into the autoencoder, nothing else is needed, no labels are required.

Multiple types of autoencoders exist, among them:

- Denoising Autoencoder: When an autoencoder has more nodes in the hidden layer than the input layer, it risks learning the "Identity Function" or "Null Function"

[2] https://towardsdatascience.com/autoencoders-introduction-and-implementation-3f40483b0a85

meaning that the output equals the input which makes the autoencoder completely worthless. To solve this problem, a corrupted (noised) copy of the input is made. Then, this copy will be passed to the autoencoder.

- Vanilla Autoencoder: This is the simplest form of an autoencoder, it consists only of three layers: input, hidden, and output. Obviously, the hidden layer is smaller than the input and output layers, and the input and output layers are of the same size.
- Undercomplete Autoencoder: If an autoencoder has a big latent space, it will most likely perform copying meaning that the autoencoder is not extracting any information, no learning is done. In contrast, an autoencoder with a small hidden layer compared to the input and output layers is called an undercomplete autoencoder.
- Convolutional Autoencoder: Known by its efficiency, convolution has repeatedly proven its strength when it comes to extracting meaningful information from an image. Meanwhile, the compressing performed by the autoencoders causes massive loss in information. Automatically, using convolution in autoencoders has become a reality. In a CAE, the encoding part is done by applying the convolution operation. Contrarily, deconvolution is used while decoding. In our research we particularly gave special attention to this type of autoencoders.

2.3 Anomaly Detection

Usually, data can be divided into groups based on a similarity criteria. Oftentimes, this division creates outliers. An anomaly is a data that is different from the bulk of the data. Depending on the field, this deviation is usually considered negative, it can be: bank

fraud, errors in a text, medical problems or a structural defect. Most of the time, anomalies are not accepted in the industry.

Anomaly detection is the process of identifying novelties in an established pattern. So, for an anomaly to be detected, first we need to establish a model, we have to answer the following question: what is normal? For instance, imagine we have a group of vehicles: nine cars and a motorcycle. Eight out of the nine cars are red and the remaining one is blue, the motorcycle is red too. If we decide that the normal is being a car, then the motorcycle will be considered as an anomaly. On the other hand, if we decide that the anomalous vehicle is the one that has a different color than the rest of vehicles, then the blue car will be considered as an outlier.

In manufacturing, defining the norm is not a hard task. Factories are built to produce good products, so by nature good products are the norm and any defective product is considered an anomaly.

CHAPTER 3: RELATED WORK

[1] presents two major contributions, the introduction of a new dataset and setting baseline results for many state-of-the-art unsupervised anomaly detection algorithms.

The MVTec dataset comprises 5354 high-resolution images, 3629 of them are for training and validation purposes, the remaining 1725 images are for testing. While the training dataset consists of only defect-free images, the testing dataset contains both: flawless images and images with several types of defects. The MVTec dataset covers 15 categories, five of them are for texture images: regular (carpet, grid) and random (leather, tile, wood), and the remaining ten categories are for different objects like: bottle, cable, and hazelnut. Additionally, 73 distinct defect types are present in the testing dataset, namely, scratches, dents, cracks, etc. Table 1 presents a statistical recap of the MVTec dataset.

	Category	# Train	# Test (good)	# Test (defective)	Image side length
Textures	Carpet	280	28	89	1024
	Grid	264	21	57	1024
	Leather	245	32	92	1024
	Tile	230	33	84	840
	Wood	247	19	60	1024
Objects	Bottle	209	20	63	900
	Cable	224	58	92	1024
	Capsule	219	23	109	1000
	Hazelnut	391	40	70	1024
	Metal Nut	220	22	93	700
	Pill	267	26	141	800
	Screw	320	41	119	1024
	Toothbrush	60	12	30	1024
	Transistor	213	60	40	1024
	Zipper	240	32	119	1024
	Total	3629	467	1258	-

Table 1: Statistical recap of the MVTec dataset.

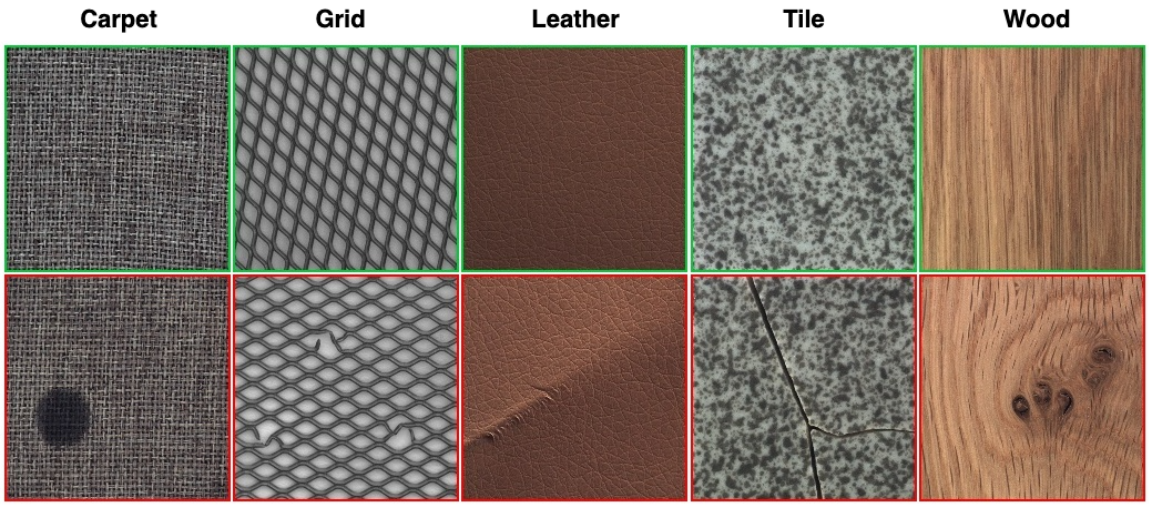

Figure 3: MVTec example images of all five textures.

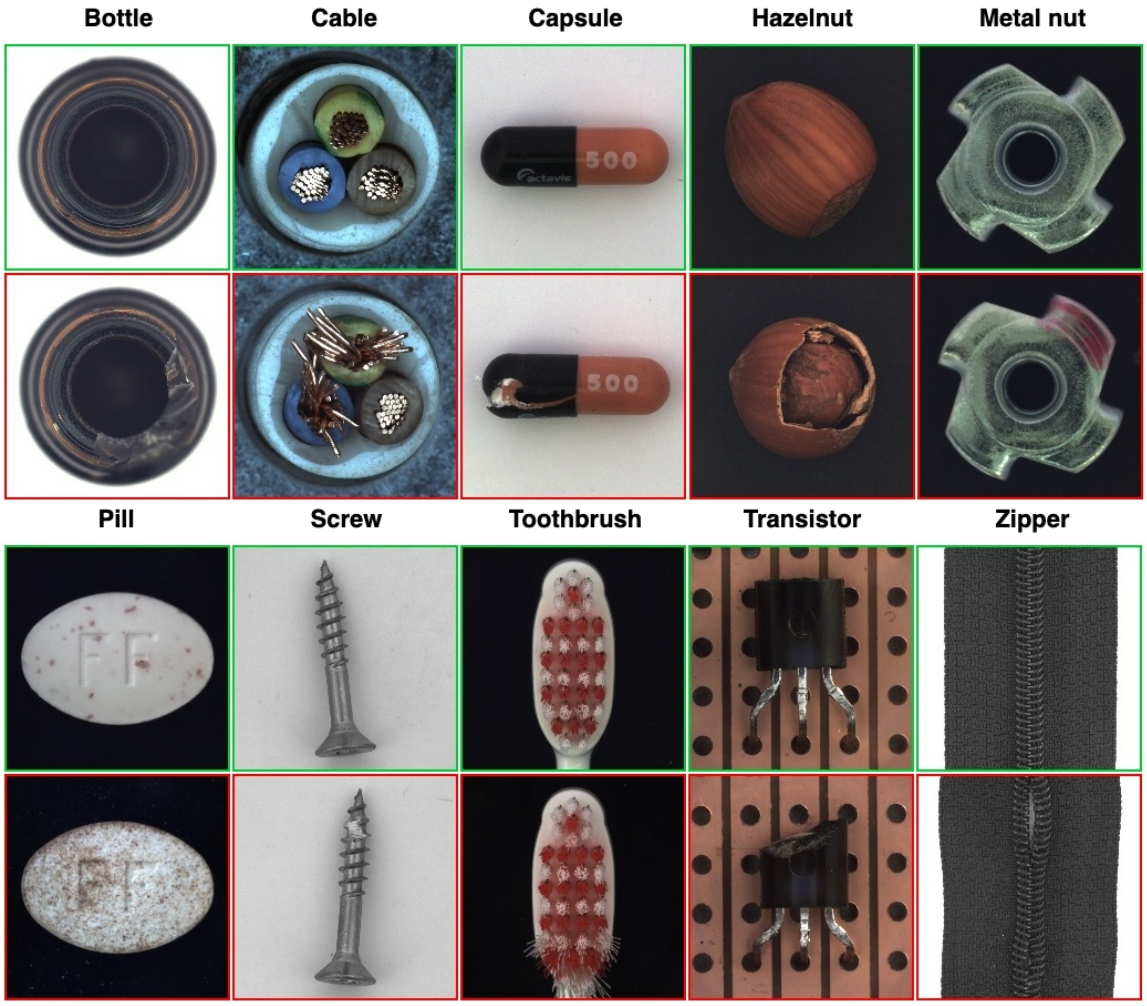

Figure 4: MVTec example images of all ten objects.

[1] includes baseline results for six state-of-the-art methods: Convolutional Autoencoder (L2), Convolutional Autoencoder (SSIM), AnoGAN, CNN Feature Dictionary, Texture Inspection, and Variational Model. Tables 2 and 3 present the results obtained. For each dataset category, the first row represents the ratio of correctly classified samples of anomaly-free images, and the second row represents the ratio of correctly classified samples of anomalous images. The method that has the highest mean of these two values is highlighted in bold.

Category	AE (SSIM)	AE (L2)	AnoGAN	CNN Feature Dictionary	Texture Inspection	Variational Model
Bottle	**0.85**	0.70	0.95	1.00	-	1.00
	0.90	0.89	0.43	0.06	-	0.13
Cable	**0.74**	0.93	0.98	0.97	-	-
	0.48	0.18	0.07	0.24	-	-
Capsule	0.78	**1.00**	0.96	0.78	-	1.00
	0.43	**0.24**	0.20	0.03	-	0.03
Hazelnut	1.00	**0.93**	0.83	0.90	-	-
	0.07	**0.84**	0.16	0.07	-	-
Metal Nut	1.00	**0.68**	0.86	0.55	-	0.32
	0.08	**0.77**	0.13	0.74	-	0.83
Pill	0.92	1.00	**1.00**	0.85	-	1.00
	0.28	0.23	**0.24**	0.06	-	0.13
Screw	0.95	**0.98**	0.41	0.73	-	1.00
	0.06	**0.39**	0.28	0.13	-	0.10
Toothbrush	0.75	**1.00**	1.00	1.00	-	1.00
	0.73	**0.97**	0.13	0.03	-	0.60
Transistor	1.00	**0.97**	0.98	1.00	-	-
	0.03	**0.45**	0.35	0.15	-	-
Zipper	**1.00**	**0.97**	0.78	0.78	-	-
	0.60	**0.63**	0.40	0.29	-	-

Table 2: Accuracies obtained for object categories.

Category	AE (SSIM)	AE (L2)	AnoGAN	CNN Feature Dictionary	Texture Inspection	Variational Model
Carpet	**0.43**	0.57	0.82	0.89	0.57	-
	0.90	0.42	0.16	0.36	0.61	-
Grid	0.38	**0.57**	0.90	0.57	1.00	-
	1.00	**0.98**	0.12	0.33	0.05	-
Leather	0.00	0.06	0.91	**0.63**	0.00	-
	0.92	0.82	0.12	**0.71**	0.99	-
Tile	1.00	**1.00**	0.97	0.97	1.00	-
	0.04	**0.54**	0.05	0.44	0.43	-
Wood	0.84	1.00	0.89	**0.79**	0.42	-
	0.82	0.47	0.47	**0.88**	1.00	-

Table 3: Accuracies obtained for texture categories.

In a similar work [18], Bergmann and his collaborators addressed anomaly detection in manufacturing environments, focusing on defect segmentation.

[18] claims that most of the convolutional autoencoders used in this field suffer from serious problems and that this is primarily due to the use of per-pixel reconstruction error based on a lp-distance. According to [18], whenever the reconstruction includes slight localization inaccuracies around edges, this approach leads to large residuals. Additionally, it fails to reveal defective regions that have been visually altered when intensity values stay roughly consistent.

To address these issues, they propose a new approach for calculating the error when reconstructing the image. Instead of using a per-pixel error measurement such as l2-distance, they propose an alternative error measure, using regions (a set of pixels) to calculate the error in place of using one pixel. This approach is commonly known as structural similarity (SSIM) metric [19].

In order to evaluate the newly introduced approach, [18] uses two real-world industrial inspection datasets. A woven fabric textures dataset built by them and the

NanoTWICE dataset of nanofibrous materials [20]. The woven fabric dataset used consists of 100 defect-free images per texture for training and validation and 50 images for testing, while the NanoTWICE dataset consists of 5 defect-free images used for training and validation along with 40 defective images for evaluation.

An alternative approach that is increasingly gaining popularity in unsupervised anomaly detection is using Generative Adversarial Networks (GANs). Even though both autoencoders and GANs have the same number of components, an encoder and decoder for the autoencoder, and a generator and a discriminator for the GAN, the philosophy followed is different. Autoencoders learn more about "how can I memorize this particular set of images with the greatest accuracy/efficiency" which is different from GANs which are more about "how can I make an image look real in general". One more difference is that unlike GANs which use two loss metrics, autoencoders use only one loss metric.

In [21], to detect anomalies such as retinal fluid, Schlegl trained a GAN on optical coherence tomography images of the retina. Specifically, this detection is accomplished by searching for a latent sample that minimizes the per-pixel l2-reconstruction error as well as a discriminator loss. This approach requires a large number of optimization steps to find a good latent sample which makes it very slow. Consequently, it is more suitable for applications that are not time-sensitive. For a faster inference, Zenati [22] proposed to use bidirectional GANs [23] to add the missing encoder network. However, since GANs are prone to run into mode collapse, having all modes of the distribution of defect-free images captured by the model is not guaranteed. Moreover, GAN training is more difficult than training an autoencoder and this is due to the fact that the loss function of the adversarial training typically cannot be trained to convergence [24].

CHAPTER 4: METHODOLOGY

Our work consists of two major steps:

1. Implementing the L2 convolutional autoencoder pipeline as described in [1] - [18].
2. Experimenting new architectures using this pipeline.

4.1 Building the L2 convolutional autoencoder pipeline

Following the approach described in [1] - [18], evaluating the L2 CAE consists of several steps, starting from the raw data (the MVTec dataset), and ending with computing accuracies. These steps are:

1. Making the training and testing datasets by preprocessing the MVTec dataset
2. Training the model using a proposed CAE architecture
3. Reconstruction of the output of the CAE
4. Measuring the distance between the input image and its reconstruction
5. Thresholding the distances in order to calculate the accuracies

Figure 5 illustrates all the steps of the pipeline. In a nutshell, first, the MVTec dataset is preprocessed to make: training, validation, testing datasets. The L2 CAE is then trained using the training and validation datasets. Predictions are then made by the trained L2 CAE using the test dataset. Using the output of the autoencoder, we reconstruct the image as a whole. Next, distances between the input images and their reconstructions are calculated. Finally, thresholds are found, and accuracies are calculated.

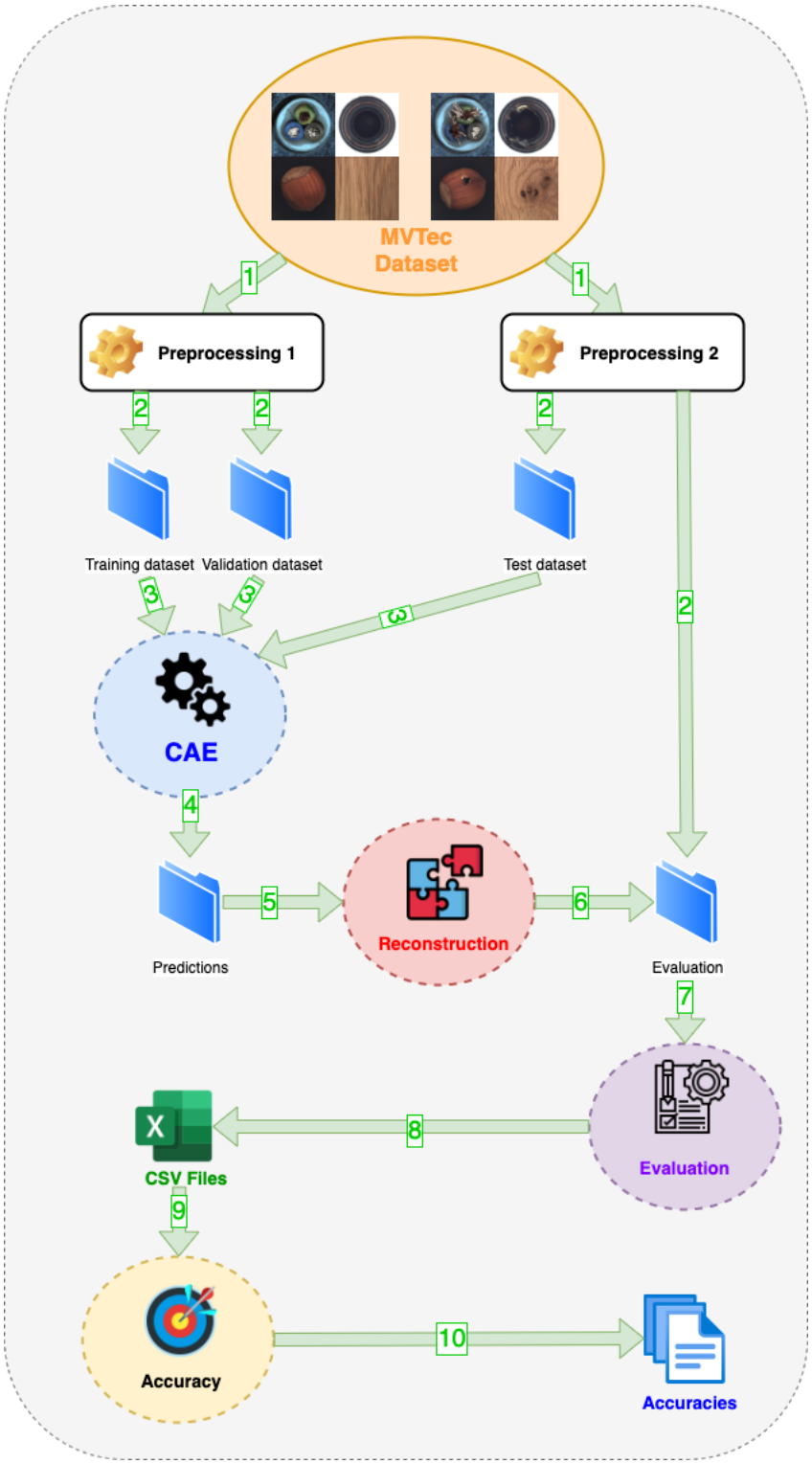

Figure 5: Pipeline of the MVTec L2 CAE.

In the following sections we will explain every step of the pipeline in detail.

4.1.1 Training dataset

Since the input layer of the autoencoder is significantly smaller than the images in the original dataset, the training dataset is formed by sampling patches. Given that the that MVTec dataset images are mostly of size 1024 by 1024, the preprocessing phase consists of:

- Resizing the images to 256 by 256
- Randomly cropping these resized images resulting in patches of 128 by 128

Knowing that for each category of the MVTec dataset (carpet, bottle, etc.) a model is to be trained, 10000 patches are made per category.

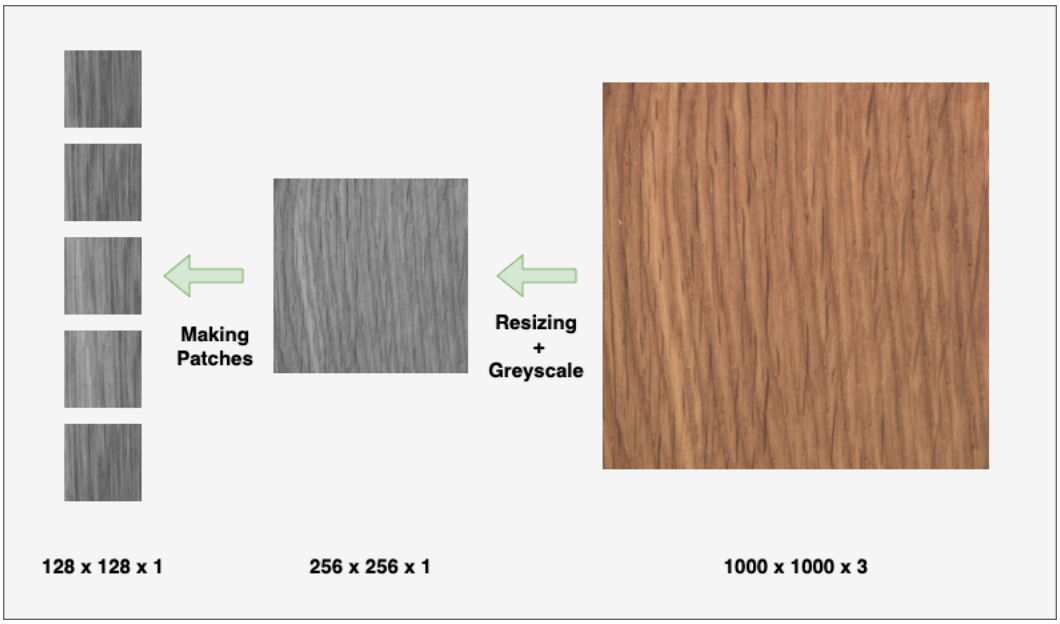

Figure 6: Process of building the training dataset.

4.1.2 Test dataset

The process of making the test dataset is different from the one of making the training dataset. In this step we don't randomly crop an image to make patches, instead, we make patches in a way to keep all the pixels of the input image.

First of all, we take the input image and we resize it to 256 by 256. Then, we stride on it a window of 128 by 128 (the size of the patch) horizontally saving each time the window values as a patch. We iterate this process by moving the window vertically by the same stride until we cover the entire image.

If no overlap is tolerated, we will get 4 patches where each patch will have 2 edges that are from the middle of the image as illustrated by Figure 7.

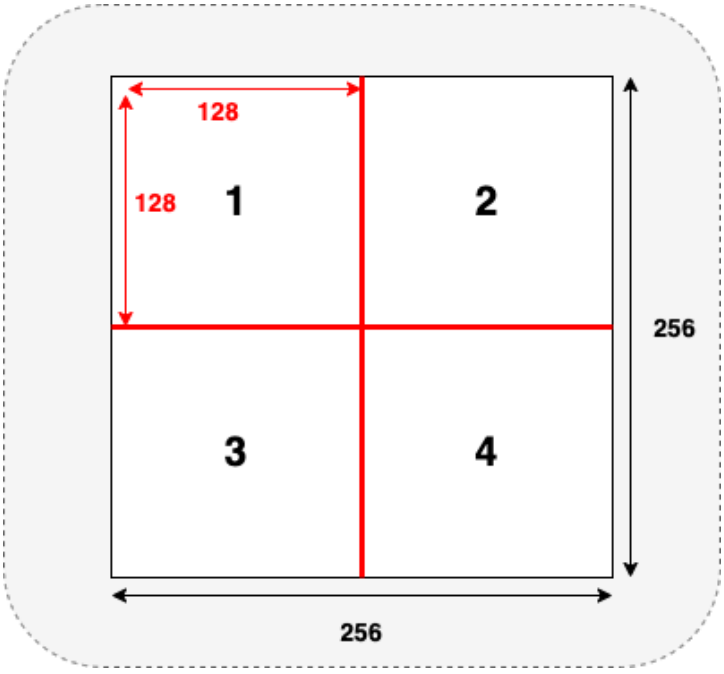

Figure 7: Making patches without overlap.

In computer vision, it is known that edges of images (the edges of the 4 patches in our case) are not considerably taken into account when doing many calculations. Usually, deep neural networks give different results on the edge regions of an image compared to the center of an image. Since the preprocessing step involves breaking larger images into patches, the edges of these patches may be critical for many images, as the edge of a patch may fall in the center of an image where an anomaly is present. For this reason, [1] chose to follow another approach, using a smaller striding step compared to the size of the patch. By doing this we are incorporating the central edges more efficiently in our future calculations, more precisely, the reconstruction step.

Using the resized images to 256 by 256, and aiming for patches of size 128 by 128, [1] chose to use a striding of 30 pixels, figure 8 illustrates this idea. This choice obviously means that padding is needed. To complete a patch when the striding arrives at the edge of the image, we will use the last encountered pixel as a gap-filler, figure 9 illustrates this.

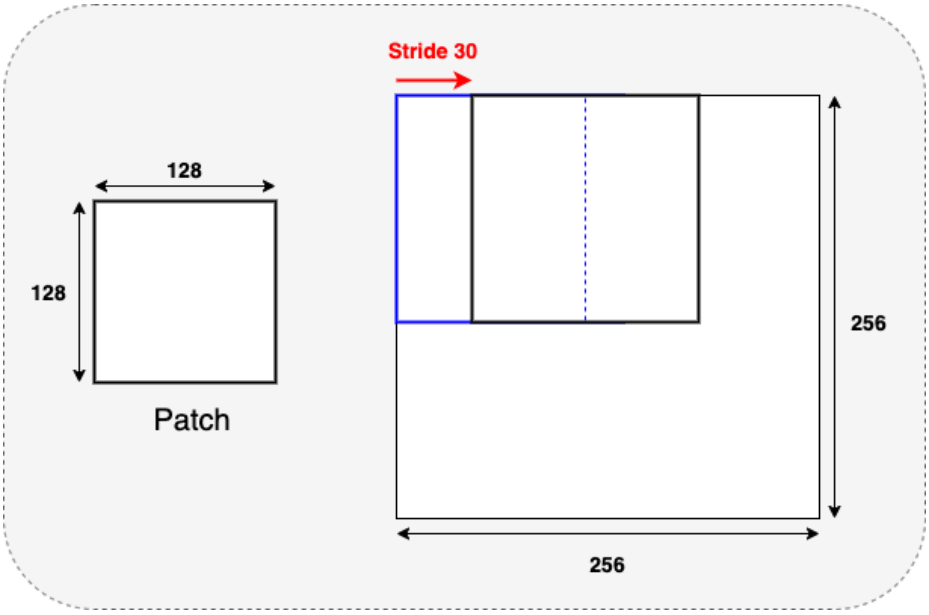

Figure 8: Making patches with overlap.

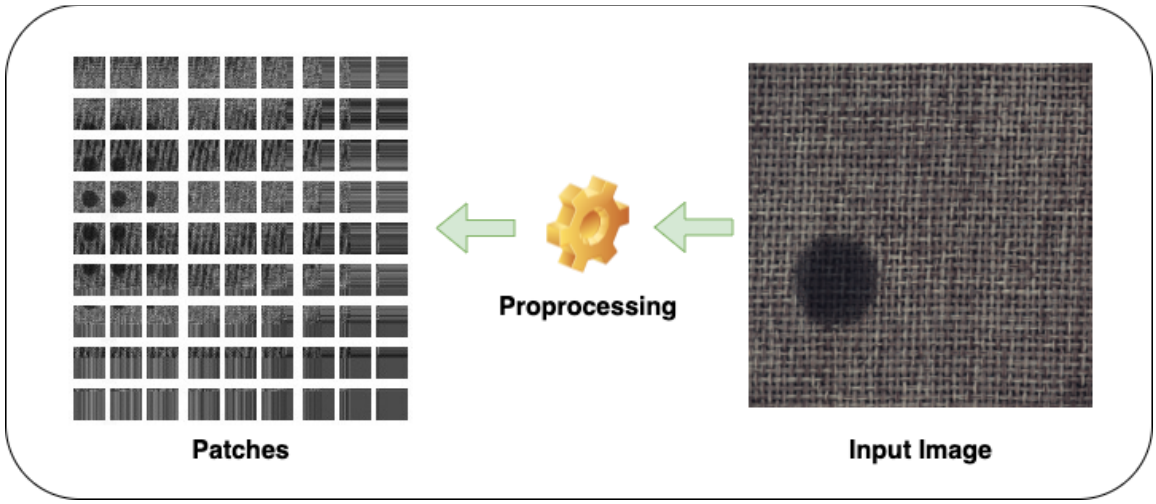

Figure 9: Building the test dataset using striding.

Each category of the MVTec dataset, like: hazelnut wood, etc., has a "raw" test subdataset. The latter comprises several images which are preprocessed to make the test dataset on which we evaluate the trained model. Using the recommended 30 pixels striding, each image gives 81 patches. While building the test dataset, we took into consideration the reconstruction step which requires knowing the order of patches. Each patch has a line number and a column number.

4.1.3 L2 CAE

We built the L2 CAE architecture following the explanation provided in [1] - [18] which consists of two parts: the encoder and the decoder, both have 9 convolutional and deconvolutional layers respectively, the dimension of the latent space is 100. Table 4 gives an overview of this architecture. Each model is trained for 200 epochs using the ADAM [25] optimizer with an initial learning rate of 2×10^{-4} and a weight decay set to 10^{-5}. Leaky rectified linear units (ReLUs) with slope 0.2 are applied as activation functions after each layer except for the output layers of both the encoder and the decoder, in which linear activation functions are used.

	Layer	Output Size	Parameters		
			Kernel	Stride	Padding
Encoder	Input	128x128x1			
	Conv1	64x64x32	4x4	2	1
	Conv2	32x32x32	4x4	2	1
	Conv3	32x32x32	3x3	1	1
	Conv4	16x16x64	4x4	2	1
	Conv5	16x16x64	3x3	1	1
	Conv6	8x8x128	4x4	2	1
	Conv7	8x8x64	3x3	1	1
	Conv8	8x8x32	3x3	1	0
	Conv9	1x1x100	8x8	1	1
Decoder	Deconv1	8x8x32	3x3	8	1
	Deconv2	8x8x64	3x3	1	1
	Deconv3	8x8x128	4x4	1	1
	Deconv4	16x16x64	3x3	2	1
	Deconv5	16x16x64	4x4	1	1
	Deconv6	32x32x32	3x3	2	1
	Deconv7	32x32x32	4x4	1	1
	Deconv8	64x64x32	4x4	2	1
	Deconv9	128x128x1	4x4	2	1

Table 4: L2 CAE architecture [18].

For the training, L2-distance is used as a loss function, hence the name L2 CAE. This choice is made for the simplicity and computational speed of this per-pixel measure.

4.1.4 Reconstruction

Once all the patches of an image, from the built test dataset, have been passed through the trained model, a reconstruction is to be done. The goal is to rebuild the full output image, the size of the latter is 256 by 256.

The test dataset was made with keeping as much information of the image as possible in mind, hence, overlapping of patches was done. As a result of this choice, the reconstruction has to take into account this overlapping.

To reconstruct the entire output image, for each patch, two distinct cases are to be taken into consideration:

1. If the pixel of the patch is not an overlapping pixel, we simply copy its value into the full output image being built.
2. If the pixel is an overlapping pixel, we calculate the average between this pixel and the corresponding overlapping pixels of all the adjacent patches. Then we copy its value to the full output image being built.

Figure 10 illustrates the places of these pixels: non overlapping and overlapping pixels.

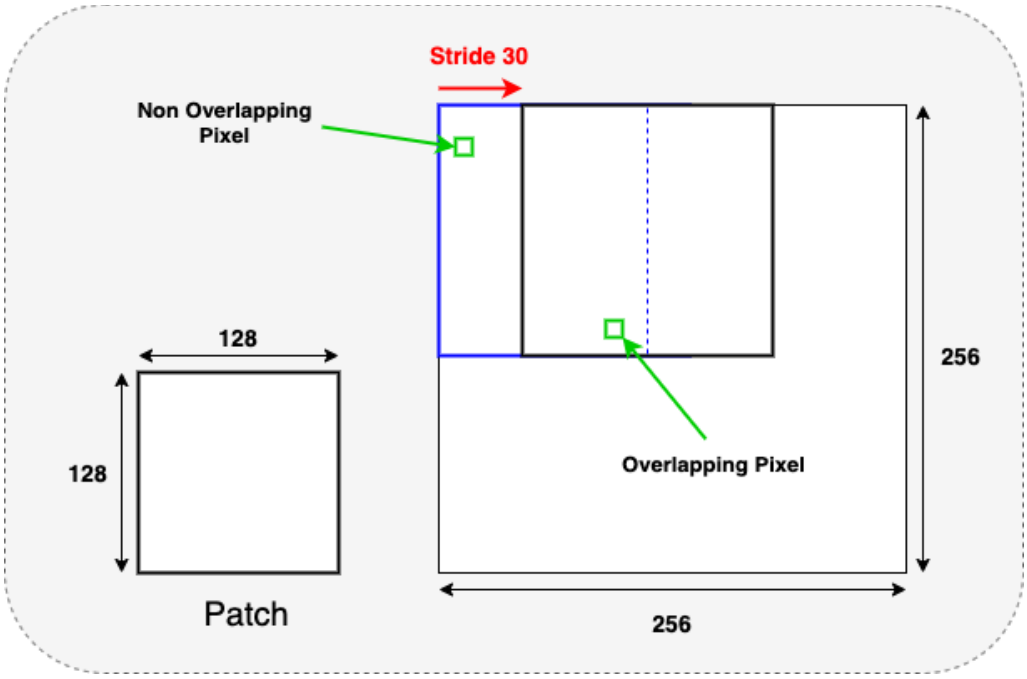

Figure 10: An example of a non-overlapping pixel and overlapping pixel.

The second reconstruction operation seems a little bit tricky because for each overlapping pixel we calculate the average of all the overlapping pixels from all patches that share this specific pixel. This operation happens for every single overlapping pixel. To make this happen, instead of uncomfortably using a set of nested loops, we opted for using another approach. We:

- First, build the full output image by copying values from all patches, summing the values of all the overlapping pixels whenever needed.
- In parallel, build an abstract matrix of the same size as the final output image (256 by 256) that contains the dividing factors by which each overlapping pixel will be divided by. For example, if a pixel of the output image is the sum of two
- pixels of two different patches, then, the corresponding value of the abstract matrix should be 2.

- Finally, calculate the average of the overlapping pixels using both matrices. We divide the built full output image by the abstract matrix of dividing factors.

To illustrate this idea, figure 11 presents a simple example of adding two patches that have two overlapping pixels in common: the most right two pixels of patch 1 and the most left two pixels of patch 2. Matrix 1 represents the merge of patch 1 and patch 2, the values of the non-overlapping pixels are copied without change from patch 1 and patch 2, and the values of the overlapping pixels are the sum of the overlapping pixels. Matrix 2 represents the calculated dividing factors, ones are due to the fact that these are non-overlapping pixels, the value corresponding to the two overlapping pixels is 2, and this is due to the number of patches sharing these pixels. Finally, the final full image output is built by dividing values of matrix 1 by the values of matrix 2.

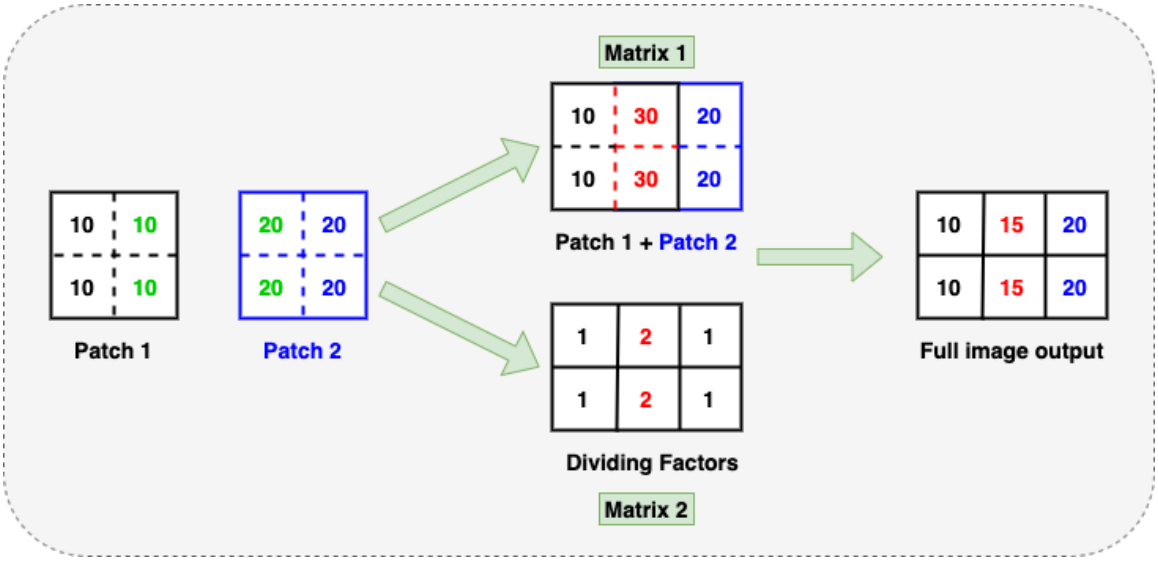

Figure 11: Illustration of the dividing factors idea.

Some of the reconstructions were good and some of them were not that good. Figure 12 shows some of the reconstructions.

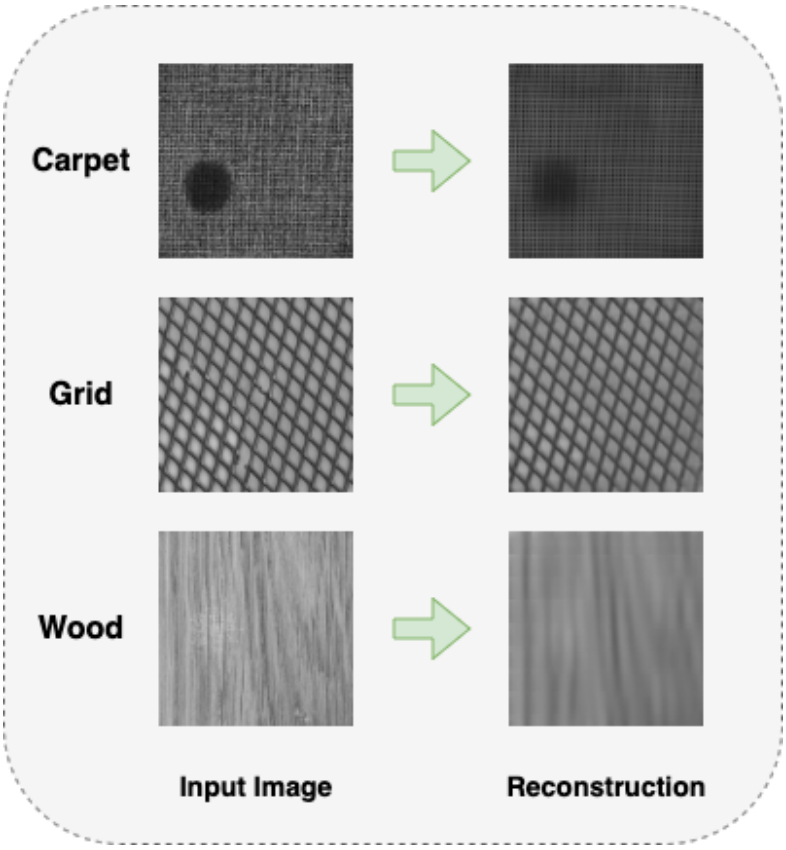

Figure 12: Samples of some reconstructions.

4.1.5 Calculating distances

Now that we have successfully reconstructed the full output image, the next step is to calculate the distance between the input and output images. The shorter the distance between the two images the more similar they are. [1] used a per-pixel measure, the Euclidean distance measure, which is also known as L2-distance. The Euclidean distance is the shortest distance between two points in a space called the Euclidean space. This

metric is commonly used to measure the similarity between two data points and used in various fields such as geometry, data mining, deep learning and others. [26]

In geometry, say we have two points P1 and P2, (x1, y1) and (x2, y2) are their coordinates respectively. The Euclidean distance between P1 and P2 is calculated following a specific formula. Figure 13 presents this formula.

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Figure 13: Euclidean distance formula.

Since we are calculating the L2-distance between two images which are basically matrices, the L2-distance between image X and its reconstruction R is calculated following the formula presented in figure 14. Where X (r, c) denotes the intensity value of image X at the pixel (r, c), and **h** an **w** represents height and width of the image.

$$L2(X, R) = \sum_{r=0}^{h-1} \sum_{c=0}^{w-1} (X(r, c) - R(r, c))^2$$

Figure 14: L2 distance between two images formula.

The L2-distance was calculated for each image of each category of the MVTec test dataset, table 5 represents some of these distances.

[1] calculated two accuracies for each category. For example, Carpet has two accuracies: the accuracy of the trained model successfully predicting good carpets, and the accuracy of the trained model successfully predicting defective models.

Driven by the way accuracies are calculated in [1], the reconstruction step produces two csv files for each category, one for good products and one for defective ones.

Category	Data Kind	L2-Distance
Carpet	Good	275290348.0
	Defective	255828067.0
Grid	Good	756916593.0
	Defective	1107697629.0
Wood	Good	1176286551.0
	Defective	1125998755.0

Table 5: Samples of L2-distances.

In this step, reconstruction, and after making the csv files, we made whisker plots. The goal is to see if we can distinguish good products from defective ones based on the calculated distances for each category. Figures 15 and 16 represent two samples of a successful and an unsuccessful distinction between good and defective products respectively.

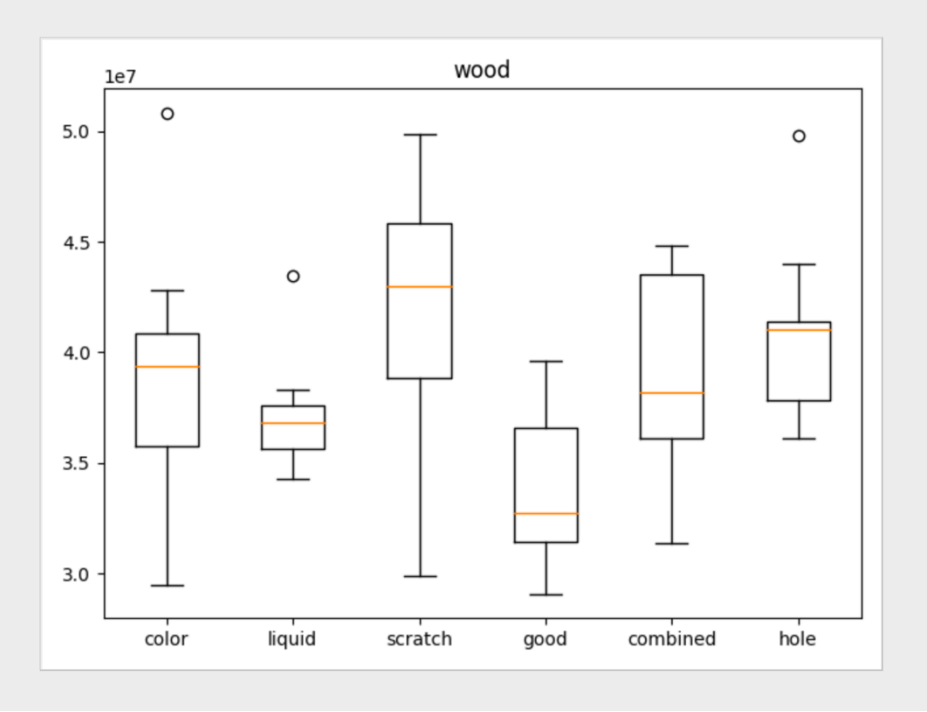

Figure 15: A sample of a successful distinction between good and defective.

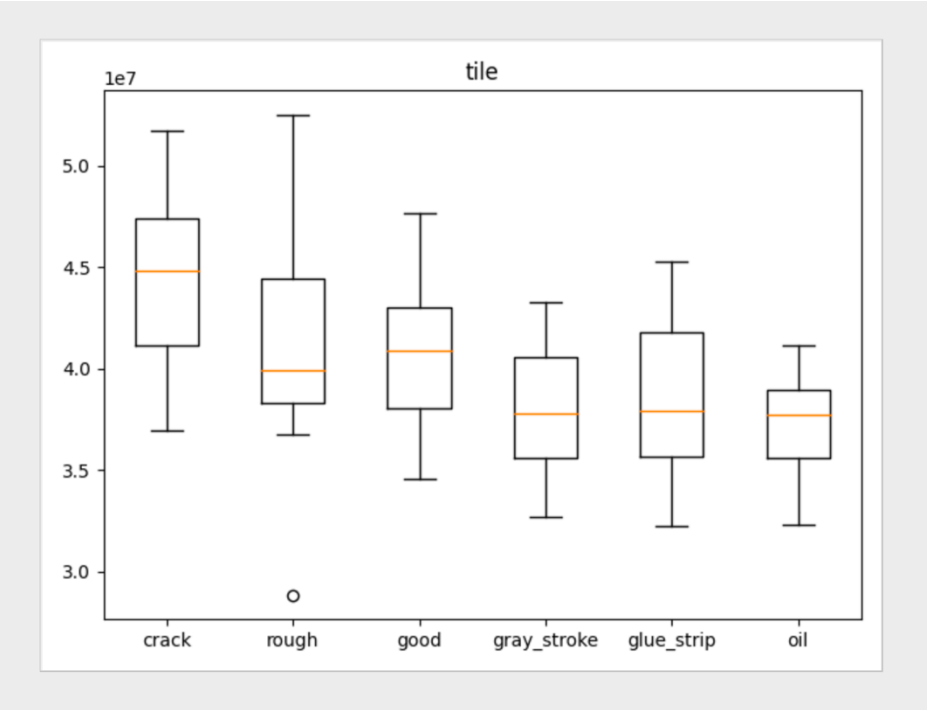

Figure 16: A sample of an unsuccessful distinction between good and defective.

4.1.6 Thresholding

After successfully calculating the distances between the input and output images for each category of the MVTec dataset, the next step consists of calculating the accuracies. To do this, we have to find a threshold for each category based on which we can decide what is successfully predicted from what is not.

Knowing that each category of the MVTec dataset has two sets of data: good and defective, [1] has exclusively used the calculated distances of the good products to find a threshold. The latter is then used on both good and defective sets of data to calculate accuracies.

In order to calculate the thresholds, we opted for the brute force approach. We tried all distances of the good products as thresholds. Then, we picked the best threshold that maximizes the accuracy of the good products. Finally, to calculate the defective products accuracy, we used this threshold on the defective products. Like this we found the two targeted accuracies: good and defective products accuracies.

Picking the 'best' threshold was the subject of some experimenting. After calculating all accuracies based on all tested thresholds, to define the best we tried the following logics, the best threshold was chosen using:

1. 100% of the calculated accuracies.
2. 90% of the calculated accuracies.
3. 80% of the calculated accuracies.
4. 70% of the calculated accuracies.

Table 6 represents results of different logics experimented to choose the best accuracy. By comparing the found accuracies using different logics with the ones of [1],

we found that using 100% of the calculated accuracies to find the best threshold was too biased to good products, we had to lower this value to 70% which we think is the best logic for picking the threshold.

	Grid	
	Good	Defective
100%	1.00	0.04
90%	0.91	0.54
80%	0.81	0.65
70%	0.71	0.74

Table 6: Grid results of experimenting different picking best threshold logics.

4.1.7 Speed of the CAE

In this work, we have also calculated the time needed by all CAEs we experimented to make predictions for each category of the MVTec test dataset. To do this, we calculated the average time taken to make a prediction for one image for each category, this was done in the most accurate way. We avoided including the loading time of images which can be affected by the read/write speed of the SSD, we have just used the prediction time.

4.2 Experimenting other architectures

Now that we have built the whole CAE pipeline, from the raw MVTec dataset to the accuracies, the next step is to experiment with other CAEs. To do so, we have done an academic review on recently published papers about CAEs, some of the papers shared

code, others shared only their CAE architecture. We finished by experimenting with three different architectures of three published papers.

4.2.1 Oh and Yun 2018

[27] focused on detecting anomalies based on abnormal sounds in special cases of manufacturing. More precisely, industrial environments that require constant verifying and monitoring of a machine. They proposed an autoencoder architecture that is based on the quality of the reconstruction to identify the anomaly. They evaluated their model using Surface-Mounted Device (SMD) machine sound. They claim achieving state-of-the-art performance for anomaly detection. Table 6 summarizes the architecture we used to run our experiment which is based on their proposed architecture. We made changes on the input and output layers to make them 128 by 128, some changes had to be done on the striding too.

4.2.2 Chow et al. 2020

[28] presents a research work that was on detecting defects on concrete structures. Chow et al. implemented a convolutional autoencoder using defect-free images to do that. The dataset used to evaluate their model is made by them and remains confidential.

	Layer	Output Size	Parameters		
			Kernel	Stride	Padding
Encoder	Input	128x128x1			
	Conv1	64x64x64	5x5	2	1
	Conv2	32x32x64	5x5	2	1
	Conv3	16x16x96	5x5	2	1
	Conv4	8x8x96	5x5	2	1
	Conv5	4x4x128	5x5	2	1
	Conv6	2x2x128	4x4	2	1
	Conv7	1x1x160	4x4	2	1
	Conv8	1x1x160	4x4	2	1
	Conv9	1x1x192	3x3	2	1
	Conv10	1x1x192	3x3	2	1
Decoder	Deconv1	2x2x192	3x3	2	1
	Deconv2	4x4x160	3x3	2	1
	Deconv3	8x8x160	4x4	2	1
	Deconv4	16x16x128	4x4	2	1
	Deconv5	32x32x128	4x4	2	1
	Deconv6	64x64x96	5x5	2	1
	Deconv7	128x128x64	5x5	2	1
	Deconv8	128x128x32	5x5	1	1
	Deconv9	128x128x32	5x5	1	1
	Deconv10	128x128x1	5x5	1	1

Table 7: [27] CAE architecture.

Tables 8 and 9 represent the architecture used to run our experiment which is based on [28] architecture.

	Layer	Output Size	Parameters		
			Kernel	Stride	Padding
Encoder	Input	128x128x1			
	Conv1	128x128x16	3x3	1	1
	Conv2	128x128x16	3x3	1	1
	MaxPooling	1 64x64x16	2x2	2	0
	Conv3	64x64x32	3x3	1	1
	Conv4	64x64x32	3x3	1	1
	MaxPooling	32x32x32	2x2	2	0
	Conv5	32x32x64	3x3	1	1
	Conv6	32x32x64	3x3	1	1
	MaxPooling	16x16x64	2x2	2	0
	Conv7	16x16x128	3x3	1	1
	Conv8	16x16x128	3x3	1	1
	MaxPooling	8x8x128	2x2	2	0
	Conv9	8x8x256	3x3	1	1
	Conv10	8x8x256	3x3	1	1
	MaxPooling	4x4x256	2x2	2	0
	Conv11	4x4x512	3x3	1	1
	Conv12	4x4x512	3x3	1	1
	Flatten	8192			
	Dense	100			
	Dense	50			

Table 8: [28] CAE architecture - Encoder.

	Layer	Output Size	Parameters		
			Kernel	Stride	Padding
Decoder	Dense	100			
	Reshape	1x1x100			
	Deconv1	2x2x512	3x3	2	1
	Conv13	2x2x512	3x3	2	1
	Conv14	2x2x512	3x3	2	1
	Deconv2	4x4x256	3x3	2	1
	Conv15	4x4x256	3x3	2	1
	Conv16	4x4x256	3x3	2	1
	Deconv3	16x16x128	3x3	2	1
	Conv17	16x16x128	3x3	2	1
	Conv18	16x16x128	3x3	2	1
	Deconv4	32x32x64	3x3	2	1
	Conv19	32x32x64	3x3	2	1
	Conv20	32x32x64	3x3	2	1
	Deconv5	128x128x1	3x3	2	1
	Conv21	128x128x1	3x3	2	1
	Conv22	128x128x1	3x3	2	1

Table 9: [28] CAE architecture - Decoder.

4.2.3 Gong et al. 2019

In [29], Gong and his collaborators claim that sometimes autoencoders "generalize" and reconstruct the anomalies which they are not supposed to do. To mitigate this problem, they propose to improve the quality of the autoencoder by using a memory module. They call their proposed model: memory-augmented autoencoder, or MemAE. This memory module is responsible for retaining a memory of the defect-free images during the training phase. In the test phase, this memory module is fixed, no more changes are to be made to

this module. The reconstruction of the test dataset images is made using this memory module which makes it tend to be closer to the defect-free images. Table 10 represents the autoencoder architecture used to run our experiment which is inspired by [29].

	Layer	Output Size	Parameters		
			Kernel	Stride	Padding
Encoder	Input	128x128x1			
	Conv1	64x64x64	5x5	2	1
	Conv2	32x32x64	5x5	2	1
	Conv3	16x16x96	5x5	2	1
	Conv4	8x8x96	5x5	2	1
Decoder	Deconv1	2x2x192	3x3	2	1
	Deconv2	4x4x160	3x3	2	1
	Deconv3	8x8x160	4x4	2	1
	Deconv4	16x16x128	4x4	2	1

Table 10: [29] CAE architecture - Decoder.

CHAPTER 5: RESULTS & FUTURE WORK

In this chapter we will present the results of our experiments. After experimenting several logics of picking the best threshold, we concluded that picking the best threshold using 70% of the calculated accuracies is the best logic. Table 11 represents the accuracies calculated for each experimented CAE along with the inference time taken to predict a single image. The winning accuracy is highlighted, this is decided after calculating the average of the two accuracies, good and defective accuracies, for each experiment. More results using the remaining logics can be found in the appendices.

5.1 Future work

In this research work, we implemented a pipeline that allowed us to conduct a series of experiments. A future work to our work could be summarized in the following points:

- Experimenting other CAE architectures.
- Trying other distance metrics in place of the L2-distance such as structural similarity (SSIM) metric [19].
- Extend the pipeline to do segmentation and this could be done by calculating residual maps.
- Experiment other approaches in place of CAEs such as AnoGANs.

Category		Bergman et al 2019 – implementation	Oh and Yun 2018	Chow et al. 2020	Gong et al. 2019	Bergman et al 2019 - paper
Carpet	Good Accuracy	0.71	0.71	0.71	0.71	**0.57**
	Defect Accuracy	0.14	0.11	0.09	0.25	**0.42**
	Inference Speed (ms)	2.152	2.718	3.360	2.086	-
Grid	Good Accuracy	0.71	**0.71**	0.71	0.71	0.57
	Defect Accuracy	0.74	**0.98**	0.81	0.70	0.98
	Inference Speed (ms)	2.138	2.717	3.336	2.088	-
Leather	Good Accuracy	0.72	0.72	**0.72**	0.72	**0.06**
	Defect Accuracy	0.15	0.15	**0.26**	0.15	**0.82**
	Inference Speed (ms)	2.132	2.689	3.356	2.197	-
Tile	Good Accuracy	0.70	0.70	0.73	0.70	**1.00**
	Defect Accuracy	0.23	0.15	0.18	0.73	**0.54**
	Inference Speed (ms)	2.119	3.383	3.351	2.077	-
Wood	Good Accuracy	0.68	0.68	0.68	**0.68**	1.00
	Defect Accuracy	0.82	0.85	0.62	**0.95**	0.47
	Inference Speed (ms)	2.128	3.375	3.351	2.059	-
Bottle	Good Accuracy	0.70	0.70	0.70	0.70	**0.70**
	Defect Accuracy	0.51	0.24	0.29	0.59	**0.89**
	Inference Speed (ms)	2.155	3.372	3.536	2.155	-
Cable	Good Accuracy	**0.72**	0.67	0.71	0.71	0.93
	Defect Accuracy	**0.46**	0.35	0.26	0.32	0.18
	Inference Speed (ms)	2.114	3.258	3.522	2.114	-
Capsule	Good Accuracy	**0.70**	0.70	0.70	0.70	1.00
	Defect Accuracy	**0.58**	0.51	0.51	0.55	0.24
	Inference Speed (ms)	2.122	3.329	3.523	2.122	-
Hazelnut	Good Accuracy	0.70	0.70	0.70	0.70	**0.93**
	Defect Accuracy	0.86	0.31	0.69	0.80	**0.84**
	Inference Speed (ms)	2.102	3.218	3.420	2.102	-
Metal Nut	Good Accuracy	0.70	0.73	0.73	0.73	**0.68**
	Defect Accuracy	0.03	0.02	0.03	0.05	**0.77**
	Inference Speed (ms)	2.107	3.193	3.462	2.107	-
Pill	Good Accuracy	0.69	0.69	**0.69**	0.69	1.00
	Defect Accuracy	0.38	0.35	**0.58**	0.51	0.23
	Inference Speed (ms)	2.093	3.421	3.035	2.107	-
Screw	Good Accuracy	0.71	**1.00**	0.71	0.71	0.98
	Defect Accuracy	1.00	**0.72**	0.00	1.00	0.39
	Inference Speed (ms)	2.064	3.393	3.390	2.109	-
Toothbrush	Good Accuracy	0.75	0.75	0.75	0.75	**1.00**
	Defect Accuracy	0.00	0.03	0.17	0.00	**0.97**
	Inference Speed (ms)	2.151	3.427	3.436	2.161	-
Transistor	Good Accuracy	0.70	0.72	0.73	0.72	**0.97**
	Defect Accuracy	0.33	0.55	0.53	0.53	**0.45**
	Inference Speed (ms)	2.077	3.403	3.366	2.093	-
Zipper	Good Accuracy	0.72	0.72	0.72	0.72	**0.97**
	Defect Accuracy	0.16	0.29	0.21	0.24	**0.63**
	Inference Speed (ms)	2.069	3.376	3.287	2.058	-

Table 11: Results of the evaluated CAEs. For each experimented method, and for each category, good products and defective products accuracies are calculated along with the inference time which was measured in milliseconds. The method with the highest mean of the two accuracies is highlighted in boldface for each category.

REFERENCES

[1] P. Bergmann, F. Michael, D. Sattlegger and C. Steger, "MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection," in *CVPR*, Long Beach, 2019.

[2] A. G. Frank, L. S. Dalenogare and N. F. Ayala, "Industry 4.0 technologies: implementation patterns in manufacturing companies," *International Journal of Production Economics,* 2019.

[3] A. Kusiak, "Smart manufacturing," *International Journal of Production Research,* 2018.

[4] K. -D. Thoben, S. Wiesner and T. Wuest, "“Industrie 4.0” and smart manufacturing-a review of research issues and application examples," 2016.

[5] H. Yang, S. Kumara, S. T. Bukkapatnam and F. Tsung, "The internet of things for smart manufacturing: A review," *IISE Transactions,* 2019.

[6] B. Giri and S. Sharma, "Optimizing a closed-loop supply chain with manufacturing defectsand quality dependent return rate," *Journal of Manufacturing Systems,* 2014.

[7] J. Piatt, "industryweek.com," 7 October 2014. [Online]. Available: https://www.industryweek.com/operations/quality/article/22008165/five-steps-to-improved-manufacturing-quality.

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2012.

[9] A. Voulodimos, N. Doulamis, A. Doulamis and E. Protopapadakis, "Deep Learning for Computer Vision: A Brief Review," *Computational Intelligence and Neuroscience,* 2018.

[10] N. Koleva, "blog.dataiku.com," 1 May 2020. [Online]. Available: https://blog.dataiku.com/when-and-when-not-to-use-deep-learning.

[11] T. Mitsa, "towardsdatascience.com," 22 April 2019. [Online]. Available: https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee.

[12] Y. Lecun, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE,* 1998.

[13] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[14] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2012.

[15] J. Emge, "buddyloans.com," 8 April 2018. [Online]. Available: https://www.buddyloans.com/news/business/competitive-nature-business-change-go-bust-113471/.

[16] "guru99.com," [Online]. Available: https://www.guru99.com/unsupervised-machine-learning.html.

[17] "deeplearningbook.org," [Online]. Available: https://www.deeplearningbook.org/contents/convnets.html.

[18] P. Bergmann, S. Löwe, M. Fauser, D. Sattlegger and C. Steger, "Improving Unsupervised Defect Segmentation by Applying Structural Similarity to Autoencoders," in *Computer Vision and Pattern Recognition*, Long Beach, 2019.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE TRANSACTIONS ON IMAGE PROCESSING,* 2004.

[20] D. Carrera, F. Manganini, G. Boracchi and E. Lanzarone, "Defect Detection in SEM Images of Nanofibrous Materials," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* 2017.

[21] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth and G. Langs, "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery," in *Computer Vision and Pattern Recognition*, Honolulu, 2017.

[22] H. Zenati, C.-S. Foo, B. Lecouat, G. Manek and V. R. Chandrasekhar, "EFFICIENT GAN-BASED ANOMALY DETECTION," in *ICDM*, Singapore, 2018.

[23] J. Donahue, P. Krähenbühl and T. Darrell, "Adversarial Feature Learning," in *ICLR*, Toulon, 2017.

[24] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," in *ICLR*, Toulon, 2017.

[25] D. P. Kingma and J. L. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," in *ICLR*, San Diego, 2015.

[26] "iq.opengenus.org," [Online]. Available: https://iq.opengenus.org/euclidean-distance/.

[27] D. Y. Oh and I. D. Yun, "Residual Error Based Anomaly Detection Using Auto-Encoder in SMD Machine Sound," *Sensors,* 2018.

[28] J. Chow, Z. Su, J. Wu, P. Tan, X. Mao and Y. Wang, "Anomaly detection of defects on concrete structures with the convolutional autoencoder," *ELSEVIER,* 2020.

[29] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh and A. v. d. Hengel, "Memorizing Normality to Detect Anomaly: Memory-augmented Deep Autoencoder for Unsupervised Anomaly Detection," in *ICCV*, Seoul, 2019.

APPENDIX A: RESULTS USING 100% OF THE ACCURACIES

Category		Bergman et al 2019 – implementation	Oh and Yun 2018	Chow et al. 2020	Gong et al. 2019	Bergman et al 2019 - paper
Carpet	Good Accuracy	**1.00**	**1.00**	**1.00**	**1.00**	0.57
	Defect Accuracy	**0.00**	**0.00**	**0.00**	**0.00**	0.42
	Inference Speed (ms)	2.152	2.718	3.360	2.086	-
Grid	Good Accuracy	1.00	1.00	1.00	1.00	**0.57**
	Defect Accuracy	0.04	0.33	0.26	0.04	**0.98**
	Inference Speed (ms)	2.138	2.717	3.336	2.088	-
Leather	Good Accuracy	**1.00**	**1.00**	**1.00**	**1.00**	0.06
	Defect Accuracy	**0.02**	**0.02**	**0.07**	**0.03**	0.82
	Inference Speed (ms)	2.132	2.689	3.356	2.197	-
Tile	Good Accuracy	1.00	1.00	1.00	**1.00**	**1.00**
	Defect Accuracy	0.07	0.01	0.01	**0.54**	**0.54**
	Inference Speed (ms)	2.119	3.383	3.351	2.077	-
Wood	Good Accuracy	1.00	1.00	1.00	**1.00**	1.00
	Defect Accuracy	0.48	0.35	0.23	**0.58**	0.47
	Inference Speed (ms)	2.128	3.375	3.351	2.059	-
Bottle	Good Accuracy	1.00	1.00	1.00	1.00	**0.70**
	Defect Accuracy	0.32	0.03	0.03	0.21	**0.89**
	Inference Speed (ms)	2.155	3.372	3.536	2.155	-
Cable	Good Accuracy	1.00	1.00	1.00	1.00	**0.93**
	Defect Accuracy	0.05	0.02	0.05	0.00	**0.18**
	Inference Speed (ms)	2.114	3.258	3.522	2.114	-
Capsule	Good Accuracy	1.00	1.00	1.00	1.00	**1.00**
	Defect Accuracy	0.00	0.00	0.00	0.00	**0.24**
	Inference Speed (ms)	2.122	3.329	3.523	2.122	-
Hazelnut	Good Accuracy	1.00	1.00	1.00	1.00	**0.93**
	Defect Accuracy	0.51	0.03	0.36	0.37	**0.84**
	Inference Speed (ms)	2.102	3.218	3.420	2.102	-
Metal Nut	Good Accuracy	1.00	1.00	1.00	1.00	**0.68**
	Defect Accuracy	0.00	0.00	0.00	0.01	**0.77**
	Inference Speed (ms)	2.107	3.193	3.462	2.107	-
Pill	Good Accuracy	1.00	1.00	1.00	**1.00**	1.00
	Defect Accuracy	0.09	0.05	0.28	**0.43**	0.23
	Inference Speed (ms)	2.093	3.421	3.035	2.107	-
Screw	Good Accuracy	**1.00**	**1.00**	**1.00**	**1.00**	0.98
	Defect Accuracy	**1.00**	**1.00**	**1.00**	**1.00**	0.39
	Inference Speed (ms)	2.064	3.393	3.390	2.109	-
Toothbrush	Good Accuracy	1.00	1.00	1.00	1.00	**1.00**
	Defect Accuracy	0.00	0.00	0.03	0.00	**0.97**
	Inference Speed (ms)	2.151	3.427	3.436	2.161	-
Transistor	Good Accuracy	1.00	1.00	1.00	1.00	**0.97**
	Defect Accuracy	0.15	0.33	0.05	0.05	**0.45**
	Inference Speed (ms)	2.077	3.403	3.366	2.093	-
Zipper	Good Accuracy	1.00	1.00	1.00	1.00	**0.97**
	Defect Accuracy	0.03	0.02	0.02	0.03	**0.63**
	Inference Speed (ms)	2.069	3.376	3.287	2.058	-

Table 12: Results of the evaluated CAEs using 100% of the calculated accuracies to find the best threshold.

APPENDIX B: RESULTS USING 90% OF THE ACCURACIES

Category		Bergman et al 2019 – implementation	Oh and Yun 2018	Chow et al. 2020	Gong et al. 2019	Bergman et al 2019 - paper
Carpet	Good Accuracy	0.93	0.93	**0.91**	0.93	0.57
	Defect Accuracy	0.00	0.00	**0.21**	0.00	0.42
	Inference Speed (ms)	2.152	2.718	3.360	2.086	-
Grid	Good Accuracy	0.91	**0.91**	0.91	0.91	0.57
	Defect Accuracy	0.54	**0.86**	0.54	0.01	0.98
	Inference Speed (ms)	2.138	2.717	3.336	2.088	-
Leather	Good Accuracy	0.91	**0.91**	0.91	**0.91**	0.06
	Defect Accuracy	0.07	**0.10**	0.09	**0.10**	0.82
	Inference Speed (ms)	2.132	2.689	3.356	2.197	-
Tile	Good Accuracy	0.91	0.91	0.91	0.91	**1.00**
	Defect Accuracy	0.16	0.04	0.01	0.62	**0.54**
	Inference Speed (ms)	2.119	3.383	3.351	2.077	-
Wood	Good Accuracy	0.95	0.90	0.90	**0.90**	1.00
	Defect Accuracy	0.50	0.35	0.25	**0.65**	0.47
	Inference Speed (ms)	2.128	3.375	3.351	2.059	-
Bottle	Good Accuracy	0.90	0.90	0.90	0.90	**0.70**
	Defect Accuracy	0.41	0.01	0.11	0.22	**0.89**
	Inference Speed (ms)	2.155	3.372	3.536	2.155	-
Cable	Good Accuracy	**0.91**	0.91	1.00	0.91	0.93
	Defect Accuracy	**0.21**	0.11	0.05	0.16	0.18
	Inference Speed (ms)	2.114	3.258	3.522	2.114	-
Capsule	Good Accuracy	0.91	0.91	0.91	0.91	**1.00**
	Defect Accuracy	0.04	0.01	0.01	0.11	**0.24**
	Inference Speed (ms)	2.122	3.329	3.523	2.122	-
Hazelnut	Good Accuracy	0.90	0.93	0.90	0.93	**0.93**
	Defect Accuracy	0.70	0.19	0.51	0.64	**0.84**
	Inference Speed (ms)	2.102	3.218	3.420	2.102	-
Metal Nut	Good Accuracy	0.91	0.91	0.91	0.91	**0.68**
	Defect Accuracy	0.00	0.01	0.01	0.01	**0.77**
	Inference Speed (ms)	2.107	3.193	3.462	2.107	-
Pill	Good Accuracy	0.92	0.92	0.92	**0.92**	1.00
	Defect Accuracy	0.13	0.21	0.31	**0.47**	0.23
	Inference Speed (ms)	2.093	3.421	3.035	2.107	-
Screw	Good Accuracy	0.90	0.90	0.90	0.90	**0.98**
	Defect Accuracy	1.00	1.00	1.00	1.00	**0.39**
	Inference Speed (ms)	2.064	3.393	3.390	2.109	-
Toothbrush	Good Accuracy	0.92	0.92	0.92	0.92	**1.00**
	Defect Accuracy	0.00	0.03	0.10	0.00	**0.97**
	Inference Speed (ms)	2.151	3.427	3.436	2.161	-
Transistor	Good Accuracy	0.92	0.92	0.92	0.92	**0.97**
	Defect Accuracy	0.23	0.40	0.08	0.10	**0.45**
	Inference Speed (ms)	2.077	3.403	3.366	2.093	-
Zipper	Good Accuracy	0.91	0.91	0.91	0.91	**0.97**
	Defect Accuracy	0.05	0.23	0.05	0.13	**0.63**
	Inference Speed (ms)	2.069	3.376	3.287	2.058	-

Table 13: Results of the evaluated CAEs using 90% of the calculated accuracies to find the best threshold.

APPENDIX C: RESULTS USING 80% OF THE ACCURACIES

Category		Bergman et al 2019 – implementation	Oh and Yun 2018	Chow et al. 2020	Gong et al. 2019	Bergman et al 2019 - paper
Carpet	Good Accuracy	0.82	0.82	0.82	0.82	**0.57**
	Defect Accuracy	0.07	0.00	0.07	0.06	**0.42**
	Inference Speed (ms)	2.152	2.718	3.360	2.086	-
Grid	Good Accuracy	0.81	**0.81**	0.81	0.81	0.57
	Defect Accuracy	0.65	**0.98**	0.66	0.56	0.98
	Inference Speed (ms)	2.138	2.717	3.336	2.088	-
Leather	Good Accuracy	0.81	0.81	**0.81**	0.81	0.06
	Defect Accuracy	0.11	0.11	**0.22**	0.14	0.82
	Inference Speed (ms)	2.132	2.689	3.356	2.197	-
Tile	Good Accuracy	0.82	0.82	0.82	0.82	**1.00**
	Defect Accuracy	0.21	0.08	0.08	0.66	**0.54**
	Inference Speed (ms)	2.119	3.383	3.351	2.077	-
Wood	Good Accuracy	0.84	0.79	0.79	**0.79**	1.00
	Defect Accuracy	0.63	0.52	0.40	**0.70**	0.47
	Inference Speed (ms)	2.128	3.375	3.351	2.059	-
Bottle	Good Accuracy	0.80	0.80	0.80	0.80	**0.70**
	Defect Accuracy	0.44	0.24	0.20	0.56	**0.89**
	Inference Speed (ms)	2.155	3.372	3.536	2.155	-
Cable	Good Accuracy	**0.81**	0.81	0.81	0.81	**0.93**
	Defect Accuracy	**0.30**	0.22	0.25	0.26	**0.18**
	Inference Speed (ms)	2.114	3.258	3.522	2.114	-
Capsule	Good Accuracy	0.83	**0.83**	0.83	0.83	1.00
	Defect Accuracy	0.46	**0.48**	0.43	0.18	0.24
	Inference Speed (ms)	2.122	3.329	3.523	2.122	-
Hazelnut	Good Accuracy	0.80	0.83	0.80	0.80	**0.93**
	Defect Accuracy	0.81	0.26	0.67	0.77	**0.84**
	Inference Speed (ms)	2.102	3.218	3.420	2.102	-
Metal Nut	Good Accuracy	0.82	0.82	0.82	0.82	**0.68**
	Defect Accuracy	0.02	0.01	0.02	0.01	**0.77**
	Inference Speed (ms)	2.107	3.193	3.462	2.107	-
Pill	Good Accuracy	0.81	0.81	0.81	**0.81**	1.00
	Defect Accuracy	0.29	0.27	0.43	**0.48**	0.23
	Inference Speed (ms)	2.093	3.421	3.035	2.107	-
Screw	Good Accuracy	**0.81**	**0.81**	**0.81**	**0.81**	0.98
	Defect Accuracy	**1.00**	**1.00**	**1.00**	**1.00**	0.39
	Inference Speed (ms)	2.064	3.393	3.390	2.109	-
Toothbrush	Good Accuracy	0.83	0.83	0.83	0.83	**1.00**
	Defect Accuracy	0.00	0.03	0.13	0.00	**0.97**
	Inference Speed (ms)	2.151	3.427	3.436	2.161	-
Transistor	Good Accuracy	0.80	0.82	0.82	0.82	**0.97**
	Defect Accuracy	0.30	0.50	0.33	0.30	**0.45**
	Inference Speed (ms)	2.077	3.403	3.366	2.093	-
Zipper	Good Accuracy	0.81	0.81	0.81	0.81	**0.97**
	Defect Accuracy	0.11	0.28	0.19	0.21	**0.63**
	Inference Speed (ms)	2.069	3.376	3.287	2.058	-

Table 14: Results of the evaluated CAEs using 80% of the calculated accuracies to find the best threshold.