Anomaly Detection in Images using Deep Encoder-Decoder Models Probabilistic Models & Machine Learning (Project 2)

Amin Fadaeinejad [1]

Abstract

The detection of anomalous structures in image data is one of the most important applications is the field of machine vision. We are going to use a unsupervised method in order to detect the defected and defect-free pixels in a image. We will also do a comparison between different model parameters to show how they will effect the result. In the manufacturing industry, for example, if there is a product that has a fault in it, it must be recognized or else it will cause future problems for the company, therefore, it is very crucial for such systems to exist in order to prevent things from happening.

1. Introduction & Background

Humans have shown great performance in detecting anomalous and abnormality in images, however, it seems that anomaly detection is a hard task for machine learning systems. There are many relevant applications that rely on unsupervised algorithms that are able to detect anomalous regions in images. In the following, we will give an explanation for the things we are about to discuss. While this classification on an image level is important, it is unclear how current state-of-the-art methods perform on what we call anomaly detection tasks. The problem set is to find novelties in images that are very close to the training data and differ only in subtle deviations in possibly very small, confined regions. Clearly, to develop machine learning models for such and other challenging scenarios we require suitable data. Curiously, there is a lack of comprehensive real-world data-sets available for such scenarios. Large-scale data-sets have led to incredible advances in many areas of computer vision in the last few years.

[1]Department of Electrical Engineering and Computer Science, York University, Toronto, Canada. Correspondence to: Amin Fadaeinejad <afadaei@yorku.ca>.

Project Report for EECS6327 (Fall 2020), Department of Electrical Engineering and Computer Science, York University.

1.1. Anomaly Detection

Anomaly detection is also known as out-layer analysis is a step-in data mining that identifies data points, events, and observations that deviate from a data set's normal behaviour. Anomalies our out-layers are data points within the data sets that appear to deviate markedly from expected outputs. It is the process of finding patterns in data that do not conform to prior expected behaviour. Anomaly detection is being employed more increasingly in the presence of big data that is captured by sensors, social media platforms, huge networks, etc. They also have many applications including energy systems, medical devices, banking network intrusion detection, etc. Machine learning is progressively being used to automate anomaly detection.

1.2. Auto-Encoders

Auto Encoder is a generative unsupervised deep learning algorithm used for reconstructing high-dimensional input data using a neural network with a narrow bottleneck layer in the middle that contains the latent representation of the input data. Auto-Encoders have two main parts called, Encoder and Decoder.

- Encoder: Accepts high-dimensional input data and translates it to latent low-dimensional data. The input size to an Encoder network is larger than its output size.
- Decoder: The Decoder network receives the input from the Encoder coder's output. The decoder's objective is to reconstruct the input data. The output size of a Decoder network is larger than its input size.

In brief works, we can say that the Auto-encoders accepts high-dimensional input data, compresses it down to the latent-space representation in the bottleneck hidden layer; the Decoder takes the latent representation of the data as an input to reconstruct the original input data.
Machine learning is progressively being used to automate anomaly detection. There are several applications for Auto-Encoders such as Anomaly Detection, Dimensionality Reduction, Recommendation Engines, Denoising Images, Image recognition and, Image generation.

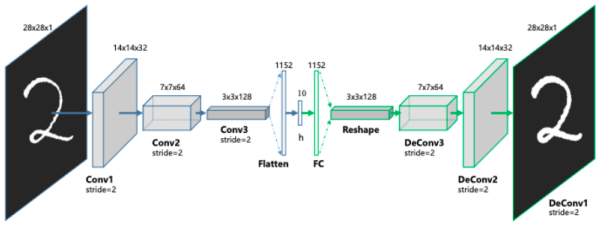

Figure 1. Network Architecture of a Auto-Encoder, from (Dertat, 2017)

1.3. Accuracy methods

In order to be able to compare our result with the results that are already published be need to have a standard accuracy method. First off we need to know the concept of True and Predicted conditions. Two types of correct predictions (True Positives and True Negatives), and there are two types of errors. Error Type I for any observation predicted positive when actually it is negative (False Positive, also called False Alert). Error Type II for any observation predicted negative when actually it is positive (False Negative).

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	True positive	False positive, Type I error
	Predicted condition negative	False negative, Type II error	True negative

Figure 2. Definition of every parameter, from (Maiza, 2019)

According the figure 2 we have an understanding towards True Positive, True Negative, False Positive and False Negative. Based on that, it is common to analyze four metrics which are:

$$Precision : \frac{TP}{TP + FP} \tag{1}$$

$$Recall : \frac{TP}{TP + FN} \tag{2}$$

$$Accuracy : \frac{TP + TN}{TP + FP + FN + TN} \tag{3}$$

But because Precision and Recall usually play against each other, we may rely on their harmonic mean which is the F1-score.

$$F_1 = \frac{2}{Recall^{-1} + Precision^{-1}} = \frac{2RecallPrecision}{Recall + Precision} \tag{4}$$

Now that we have a clear understanding towards the accuracy metrics we can show the result in the next parts of this report.

2. Related Works (literature reviews)

There are several methods including supervised and unsupervised for detecting anomalies. The work of (Paul Bergmann, January 2021) is a great work for this field and it is also our implementation to use their data-set and why using a network is a bit different from theirs. The landscape of methods for unsupervised anomaly detection is diverse and many approaches have been suggested to tackle the problem (An & Cho., 2015) (I. Goodfellow & Courville., 2016). Other than that there are other methods such as:

2.1. Generative Adversarial Networks (GANs)

(T. Schlegl, Springer, 2017) propose to model the manifold of the training data by a generative adversarial network (GAN) (I. Goodfellow & Bengio., 2014) that is trained solely on defect-free images. The generator is able to produce realistic-looking images that fool a simultaneously trained discriminator network in an adversarial way. For anomaly detection, the algorithm searches for a latent sample that reproduces a given input image and manages to fool the discriminator. An anomaly segmentation can be obtained by a per-pixel comparison of the reconstructed image with the original input.

2.2. Deep Convolutional Autoencoders

Convolutional Autoencoders (CAEs)(I. Goodfellow & Courville., 2016) are commonly used as a base architecture in unsupervised anomaly detection settings. They attempt to reconstruct defect-free training samples through a bottleneck (latent space). During testing, they fail to reproduce images that differ from the data that was observed during training. Anomalies are detected by a per-pixel comparison of the input with its reconstruction. Recently, (P. Bergmann, 2019) pointed out the disadvantages of per-pixel loss functions in autoencoding frameworks when used in anomaly segmentation scenarios and proposed to incorporate spatial information of local patch regions using structural similarity (Z. Wang & Simoncelli., 2004) for improved segmentation results.

3. Resources (Software and Data)

3.1. Hardware

For training deep neural network model using CPU is a bad option since in training you are doing many small computations for each node in a parallel way it is best to use GPUs for training. For our project we used a free Google

Colab account which several users have access to a 12GB NVIDIA Tesla K80 GPU that can be used up to 12 hours continuously.

3.2. Data Set

For our project, we used the MVTec data-set that belonged to (Paul Bergmann, January 2021), this data set contained several different groups of images such as Bottles, Cables, Capsule, Carpet, Grid, Hazelnut, Leather, Metal Nut, Pill, Screw, Tile, Toothbrush, Transistor, Wood and Zipper. Which for our project due to the limitation we were able to only train one of the data sets. We used the Hazelnut data-set which contained 390 defect-free images. Deep neural networks are known for their impressive accuracy when they are trained with a very large data set, therefore we used data augmentation to increase the number of data samples that we have. We used methods suck as rotation, flipping and adding noise to increase the number of data samples. In the end, we had approximately 2000 images. There are 70 test images which different types such as crack, cut, hole and print. The data-set also includes ground Truth to calculate the accuracy of the testing.

Figure 3. Hazelnut test images with their ground truth is MVTec data-set, from left to right: crack, cut, hole and paint

4. Network Architect

Our model contained several convolution layers. For the model we also used Batch-Normalization with 0.1 momentum and $\epsilon = 10^{-6}$. For the activation function we used the LeakyReLU with negative slop 0.01.

4.1. Encoder

For the model architect refer to Table 1.

4.2. Decoder

For the model architect refer to Table 2.

Table 1. In the table "IN" refers to the input size, "OUT" refers to the output size, "KERNEL" refers to the size of the kernel and "STRIDE" refers to the value of the stride

LAYER TYPE	IN	OUT	KERNEL	STRIDE
CONV2D	3	3	4	2
BATCHNORM2D	3			
LEAKYRELU				
CONV2D	3	32	4	2
BATCHNORM2D	32			
LEAKYRELU				
CONV2D	32	32	4	2
BATCHNORM2D	32			
LEAKYRELU				
CONV2D	32	32	3	1
BATCHNORM2D	32			
LEAKYRELU				
CONV2D	32	64	4	2
BATCHNORM2D	64			
LEAKYRELU				
CONV2D	64	64	3	1
BATCHNORM2D	64			
LEAKYRELU				
CONV2D	64	128	4	2
BATCHNORM2D	128			
LEAKYRELU				
CONV2D	128	64	3	1
BATCHNORM2D	64			
LEAKYRELU				
CONV2D	64	32	3	1
BATCHNORM2D	32			
LEAKYRELU				
CONV2D	32	500	8	1
BATCHNORM2D	500			

5. Experiments

5.1. Training

Just like any other network, we need to train the model in order to get a good performance from it. We used the $L2$ Loss of training. Where x_i is the input image and $\hat{x}_i$ is the generated image.

$$Loss = \frac{1}{n}\sum_{1}^{n} ||x_i - \hat{x}_i|| \tag{5}$$

F1 metric is one of the metrics that is proper for calculating pixel accuracy or segmentation tasks. Figures 5 6 7 8 represent the F1 accuracy for different test images. (For the accuracy we assumed $\tau = 0.2$).

5.2. Outputs

We used the algorithm to detect the defect in the testing images and for every data-set we represented a result.

Table 2. In the table "IN" refers to the input size, "OUT" refers to the output size, "KERNEL" refers to the size of the kernel and "STRIDE" refers to the value of the stride

LAYER TYPE	IN	OUT	KERNEL	STRIDE
CONVTRANSPOSE2D	500	32	8	1
BATCHNORM2D	32			
LEAKYRELU				
CONVTRANSPOSE2D	32	64	3	1
BATCHNORM2D	64			
LEAKYRELU				
CONVTRANSPOSE2D	64	128	3	1
BATCHNORM2D	128			
LEAKYRELU				
CONVTRANSPOSE2D	128	64	4	2
BATCHNORM2D	64			
LEAKYRELU				
CONVTRANSPOSE2D	64	64	3	1
BATCHNORM2D	64			
LEAKYRELU				
CONVTRANSPOSE2D	64	32	4	2
BATCHNORM2D	32			
LEAKYRELU				
CONVTRANSPOSE2D	64	128	4	2
BATCHNORM2D	128			
LEAKYRELU				
CONVTRANSPOSE2D	128	64	3	1
BATCHNORM2D	64			
LEAKYRELU				
CONVTRANSPOSE2D	64	32	4	2
BATCHNORM2D	32			
LEAKYRELU				
CONVTRANSPOSE2D	32	32	3	1
BATCHNORM2D	32			
LEAKYRELU				
CONVTRANSPOSE2D	32	32	4	2
BATCHNORM2D	32			
LEAKYRELU				
CONVTRANSPOSE2D	32	3	4	2
BATCHNORM2D	3			
LEAKYRELU				
CONVTRANSPOSE2D	3	3	4	2
SIGMOID				

5.3. Print

For print we used $\tau = 0.2$, figures 9 show the result.

5.4. Crack

For crack we used $\tau = 0.2$, figures 10 show the result.

5.5. Cut

For cut we used $\tau = 0.2$, figures 11 show the result.

5.6. Hole

For hole we used $\tau = 0.25$, figures 12 show the result.

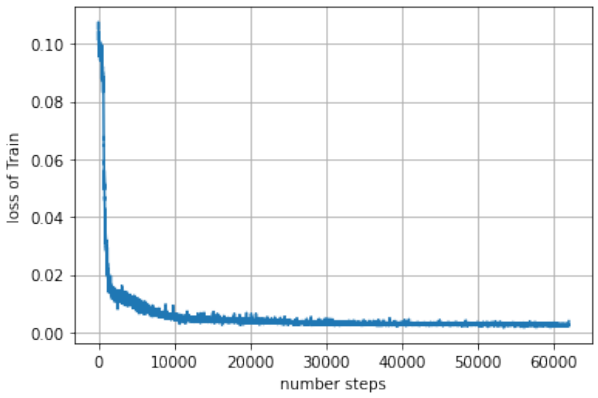

Figure 4. Training loss

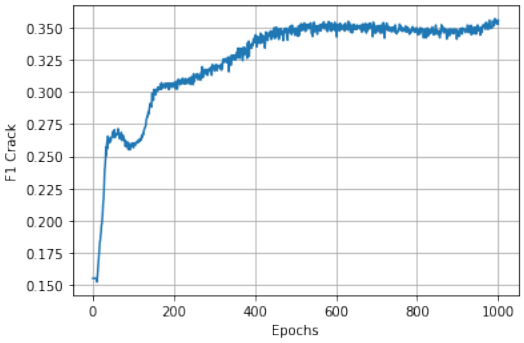

Figure 5. F1 Accuracy for crack data

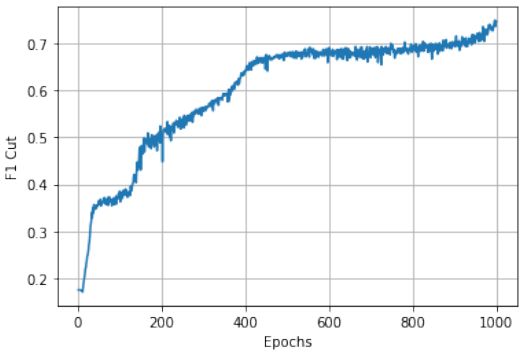

Figure 6. F1 Accuracy for cut data

6. Methodology

6.1. Method

We are going to train our network in order to reconstruct the input image from the latent space the way that we do that is that we feed many normal images to the network so the network will learn the features of normal and defect-free images because the latent space is smaller than the size of the image, the network learns the important part of

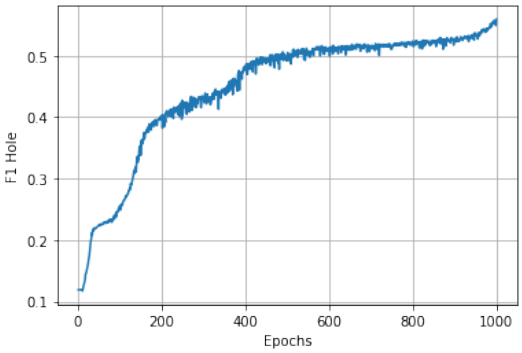

Figure 7. F1 Accuracy for hole data

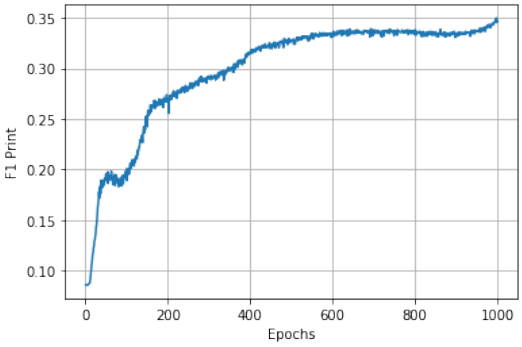

Figure 8. F1 Accuracy for print data

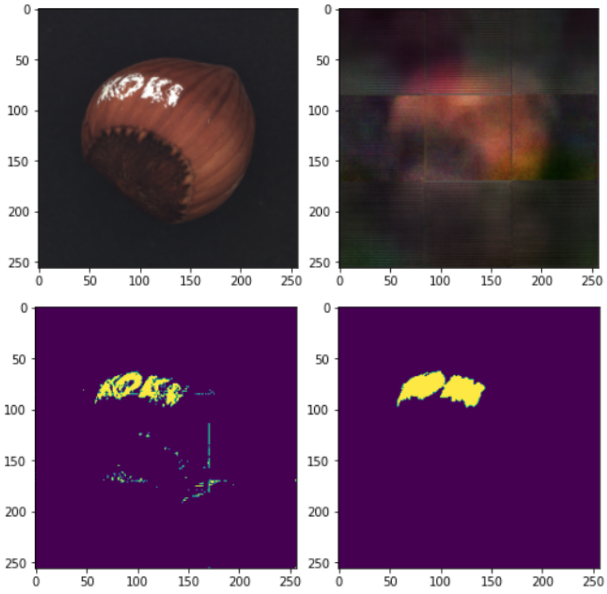

Figure 9. Print Example, Top left: Input, Top right: Output, Bottom left: Predicted defect pixels, Bottom right: Ground Truth

the image, after that since it has learned the important part whenever it sees a new image it will try to use the things it

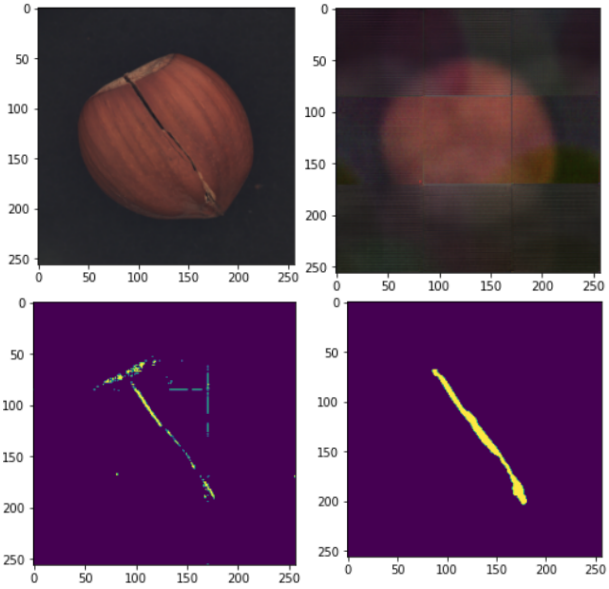

Figure 10. Crack Example, Top left: Input, Top right: Output, Bottom left: Predicted defect pixels, Bottom right: Ground Truth

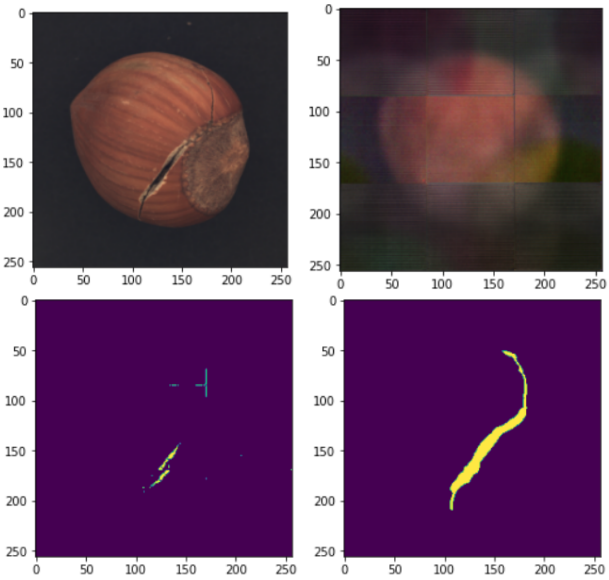

Figure 11. Cut Example, Top left: Input, Top right: Output, Bottom left: Predicted defect pixels, Bottom right: Ground Truth

learned to construct the image and since a defected image has a problem it will have a high constructed error and by that, we find the defected pixels.

- Training:
  Input normal data to learn latent representation (Encoder) Use the latent representation to reconstruct the first image (Decoder)
- Testing:

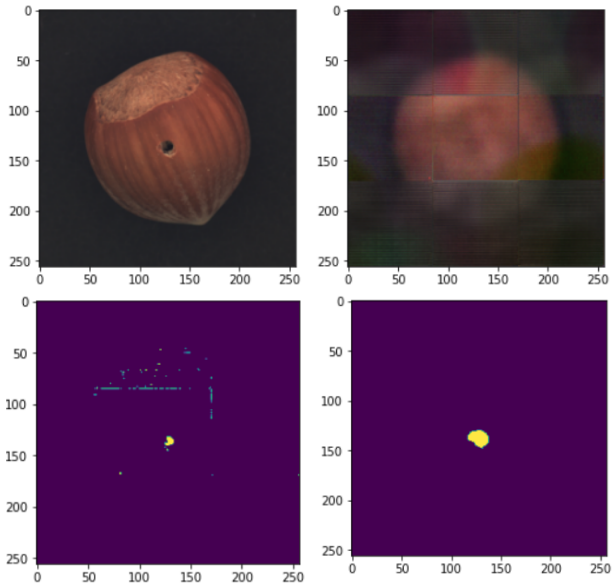

Figure 12. Hole Example, Top left: Input, Top right: Output, Bottom left: Predicted defect pixels, Bottom right: Ground Truth

An image with defect will have a different output compared to normal, hence error will be high Apply a threshold for the reconstruction error to detect the error.

6.2. Algorithms

For the testing part in order to detect the defected pixels we need to use a specifies algorithm.

By using algorithm 1 we can construct an image that shows which pixels have fraud in them. In 5.2 we will see the result of this algorithm.

Algorithm 1 Defect Detection

Input: image(RGB) x_i, number data m, Threshold τ
repeat
  for $i = 1$ **to** m **do**
    $\hat{x}_i \leftarrow f(x_i)$
    $D \leftarrow ||\hat{x}_i - x_i||$
    $E \leftarrow max(D)$ over channel (Converting 3D(RGB) channel to 1D channel)
    for n,k in range of images height and width **do**
      if $E_{nk} > \tau$ **then**
        $E_{nk} \leftarrow 1$
      end if
      if $E_{nk} < \tau$ **then**
        $E_{nk} \leftarrow 0$
      end if
    end for
    show E as the output
  end for
until All images are done

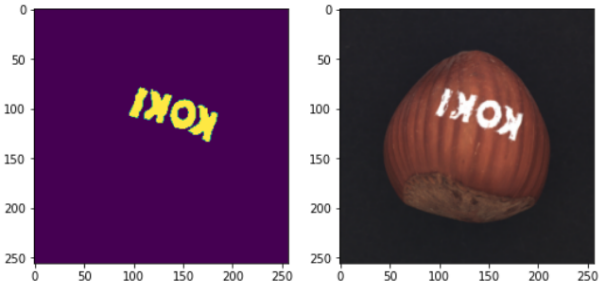

Figure 13. Left: Ground Truth, Right: Test image

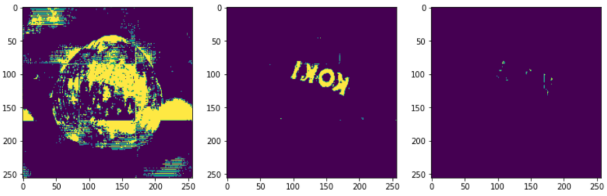

Figure 14. Left: $\tau = 0.7$, Middle: $\tau = 0.2$, Right: $\tau = 0.1$

7. Discussions

7.1. Effect of Threshold

As we mentioned in 5.2, the output depends on parameter τ, which in this part we are going to show the effect of changing this value. In figures 13 and 14 the effect of the threshold can be seen. If the value of the threshold is too high (for example $\tau = 0.7$ in figure 14) the system will miss some pixels that were in fact defected pixels, therefore the F1 accuracy will decrease. If the value of the threshold is too low (for example $\tau = 0.1$ in figure 14) the system will assume that the defect-free pixels are also defected, in this case the F1 will decrease as well since there is a large number of pixels that do not contain defected pixels but the system will predict it anyway.

7.2. Effect of Latent Space size

In our model and result, the dimension of the latent space was 500, however, this is a hyper-parameter that we could change. Changing this parameter will cause effects on the output. As we mentioned in 1 Decoder's objective is to reconstruct the input data from the output of the Encoder. If the size latent space is too small then the model will face difficulties to learn small details since it has less power to learn the model, therefore in some cases, it might predict some pixels have defected while they are completely defect-free. If we assume that the latent space is too small in

that case the model will start to memorize the input image instead of learning the image features, the model might construct an image that has defective parts since it has a lot of parameters it has the capacity to pass the small details as well, this might lead to a situation that the model is not able to detect the defected pixels in an image since the model will construct an image that has those parts due to its capacity.

8. Conclusions

In the last part we are going to compare our result with other methods and approaches that are out there.

Table 3. The Methods accuracy

METHODS	RECALL	PRECISION	F1
OUR METHOD (PRINT)	0.22	0.82	0.35
OUR METHOD (CRACK)	0.26	0.57	0.36
OUR METHOD (CUT)	0.69	0.8	0.57
OUR METHOD (HOLE)	0.55	0.61	0.75
AE(SSIM)	0.08	**1**	0.15
AE(L2)	**0.84**	0.93	**0.88**
ANOGAN	0.16	0.83	0.27
CNN FEATURE DICTIONARY	0.07	0.90	0.13

We compared our approach with some of the models that are already used and it seems (Paul Bergmann, January 2021) has the best performance when they train their model on a $L2$ loss function. But it seems that our approach is good since we outperform other methods such as AnoGANs, Auto Encoders using SSIM and CNN feature Dictionary. However, our approach has a lower accuracy in all 3 metrics (Recall, Precision and F1) compared to the state-of-the-art method. In one special case, the AE approach with the SSIM loss function was able to reach 100 % for precision, however, they were not able to have a good performance of the Recall, therefor according to our F1 metric, our approach outperforms the AE(SSIM) approach.

References

An, J. and Cho., S. Variational autoencoder based anomaly detection using reconstruction probability. *Technical report, SNU Data Mining Center,*, 2015.

Dertat, A. Applied deep learning - part 3: Autoencoders. Technical report, 2017.

I. Goodfellow, J. Pouget-Abadie, M. M. B. X. D. W.-F. S. O. A. C. and Bengio., Y. Generative adversarial nets. *In Advances in Neural Information Processing Systems,*, 2014.

I. Goodfellow, Y. B. and Courville., A. Deep learning. *MIT Press, Cambridge, MA,*, 2016.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Maiza, A. The unknown benefits of using a soft-f1 loss in classification systems. Technical report, 2019.

P. Bergmann, S. Lowe, M. F. D. S. C. S. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. *14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications,*, 2019.

Paul Bergmann, Kilian Batzner, M. F. D. S. C. S. A comprehensive real-world dataset for unsupervised anomaly detection;. *International Journal of Computer Vision*, January 2021.

T. Schlegl, P. Seebock, S. M. W. U. S.-E. G. L. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *International Conference on Information Processing in Medical Imaging,*, Springer, 2017.

Z. Wang, A. C. Bovik, H. R. S. and Simoncelli., E. P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing,*, 2004.