Master thesis
Data Science

Radboud University

Using fast visual inpainting transformers for anomaly detection

Author:
Sébastiaan Versteeg BSc
hello@sebastiaan.app

First supervisor:
prof. dr. E. Marchiori
elenam@cs.ru.nl

Second assessor:
dr. ir. T. Claassen
tomc@cs.ru.nl

September 23, 2022

Abstract

The detection and segmentation of irregularities in images is an anomaly detection problem in computer vision. An approach that is used commonly uses autoencoders trained on good images to fully reconstruct anomalous images. These reconstructions can then be used to perform detection and segmentation by mathematically comparing the original input and the reconstruction. Other solutions introduce patch inpainting as a solution for anomaly detection. One approach uses an attention-based transformer model that achieves promising results. Transformers are generally slower models and thus faster versions have been proposed. We therefore combine the inpainting solution with two types of transformers and see that a linear attention approach performs slightly better than full attention when doing detection and segmentation on the MVTec AD dataset.

Contents

Chapter 1

Introduction

The detection of deviations in data is an important topic in multiple fields. Examples of these include medical imaging [19], surveillance [30] and manufacturing [31]. These types of anomaly detection aim to find irregularities in different types of images. Using this type of processing we can improve outputs and diagnoses of systems either by automating the process or giving support in manual tasks.

Notably self-supervised anomaly detection is an approach that has been explored recently [1, 22]. This type of machine learning means that we use unlabeled data to train our models. We can use examples of normal situations to be able to reconstruct full images where the anomaly has been removed. This allows us to compare a source image to a reconstruction and find any anomalies. This method using image inpainting has been implemented using various methods like generative adversarial networks (GANs) [40] and visual transformers [27].

Transformers [33] are a type of model that is based on self-attention. This attention processes a lot of context, which makes the training of these types of models slow. Therefore alternatives have been explored, like linear transformers [21], fastformers [36] and linformers [34].

Since the linear transformers seem to have some performance improvement over the vanilla transformers we see an opportunity to use this newer type of transformer in the context of image inpainting for anomaly detection. We propose to use a similar method to the one used by Pirnay et al. [27] where we replace the used visual transformer by a visual linear transformer similar to the one introduced by Katharopoulos et al. [21]. Thus we ask:

What is the effect on the performance and efficiency of using linear transformers in an inpainting context for anomaly detection when compared to regular transformers?

To be able to compare the different models we ask subquestions about efficiency and performance:

- What is the difference between softmax and linear attention in *detec-*

tion of anomalies across images with different textures and objects?

- What is the difference between softmax and linear attention in *segmentation* of anomalies across images with difference textures and objects?
- How does the *number of epochs* required to get the best result compare between the softmax and linear attention models across images with different textures and objects?
- How does the *training time* compare between the softmax and linear attention models across images with different textures and objects?
- How do our results compare to previous work?

We will answer these questions by doing an in-depth analysis of using linear transformers compared to regular transformers.

This thesis is structured as follows: first we will discuss machine learning in general (2.1). We will look at transformers and the different types that we need to answer our research questions (2.2). Next we will introduce the concepts of image inpainting and anomaly detection (2.3, 2.4) followed by measures for image similarity (2.5). Consecutively we will discuss related work in the fields of image inpainting and anomaly detection (3.1, 3.2) before explaining how we setup our experiment (4). Then we will show the results from the experimental setup looking at detection, segmentation (5.1) and efficiency (5.2). A discussion of the experimental setup and the results follows (6) and lastly we will draw our conclusions (7).

Chapter 2

Preliminaries

2.1 Types of machine learning

There are different approaches to machine learning. In this section we will introduce a few of these approaches to give context for our research.

2.1.1 Supervised learning

Supervised learning [24] is a type of machine learning approach that always uses a labeled dataset to train a model. This means that for each point in the dataset we already know what class or rank it belongs to. It is the common type of learning used with problems that require classification, regression or ranking. The known samples are used to train a model and to make predictions for data points that the model has not seen yet.

An example would be making predictions about images of animals. Each image would be labeled with the type of animal that is in the picture. Use that data we would be learning our models to be able to predict the type of animal in pictures that the model has never seen before.

2.1.2 Unsupervised learning

This subset of machine learning focuses on training models with unlabeled data. Examples of problems that use this type of learning are clustering and dimensionality reduction. For unsupervised methods it is hard to measure the performance quantitatively because there are no labels to compare the results to.

An example of this type would be clustering people by known information like age, education or income. The goal here is to split the dataset based on information that is similar between the points. So only the points from the dataset are used to create a result, there is no previous knowledge about the data that can be used.

2.1.3 Self-supervised learning

The type of machine learning that we use in this thesis is a form of self-supervised learning [15]. This means that we generate our own labels from a dataset without any manual work required. This is possible by removing or modifying part of the input data and then learn a model to recover the original data. We can measure the performance based on the difference between the original data and the result. This type of learning can be seen as a combination of unsupervised and supervised learning.

An example is a model that tries to predict words in a certain context. We have a sentence: 'The most popular food from France is the croissant'. We can then leave out the word 'croissant' and train a model to predict the word based on the words surrounding it.

2.2 Transformers

The type of model used in our research is a transformer. In this section we will introduce the attention mechanism, explain how this is used in transformers and how this is affecting the kind of transformer we are using.

2.2.1 Vanilla transformers

In 2017 Vaswani et al. [33] introduced the transformer model. A type of sequence to sequence model that uses an attention mechanism as the main building block. This allowed them to increase parallelisation and reach a new state of the art in the context of language translations.

Architecture

The transformer model uses the same encoder-decoder structure used in the models it was compared to in the original paper [3, 11]. This means that the model consists of two parts: an encoder and a decoder.

The encoder takes an input representation $(x_1, ...x_n)$ and transforms this to an intermediary representation $(z_1, ...z_n)$. This intermediary representation is then used as input for the decoder to create an output $(y_1, ..., y_n)$.

Attention The attention mechanism, which takes query, keys and values vector as input, outputs a vector containing a weighted sum of the values. Each of the output values uses a weight that is calculated using a compatibility function of the query and the key in each position.

The attention mechanism [33] is called 'Scaled Dot-Product Attention' or softmax attention and calculates attention using matrices that consist of multiple vectors packed together. These matrices are Q (queries), K (keys)

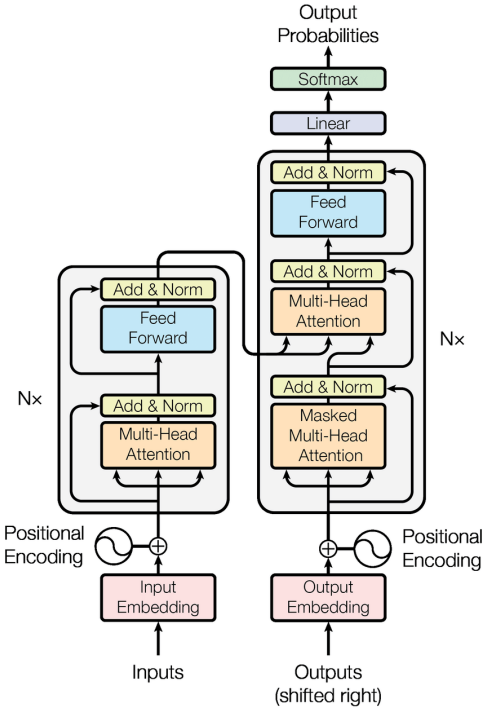


Figure 2.1: The original transformer architecture from [33]

and V (values) of dimension $S \times d_k$, $d_k \times S$ and $S \times d_v$ where S is the sequence length of vectors:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

To be able to apply the attention to more than one representation space we use multi-headed attention. This uses h parallel layers that all attend to a smaller part of the full dimension. In the case of the original papers $h = 8$ and the dimension of each attention head becomes $d_k = d_v = D/h = 64$ when $D = 512$. This makes the multi-headed attention similar to single-head attention with the full dimension when we look at the computational cost.

To combine the results of the attention heads we learn linear projections (W_i^Q, W_i^K, W_i^V) that we project all queries, keys and values to. All the separate outputs of the different attention heads are concatenated and projected into the correct dimension , which gives us the final output.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.2}$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

Encoder The encoder part in the architecture is a stack of N layers. All the layers are the same and consist of two sublayers: the multi-head self-attention and a fully-connected feed-forward network (FFN). After both sublayers we apply layer normalisation [2] and add residual connections [20] around the sublayers. This means that the output of a layer in the encoder is calculated as follows:

$$selfAtt = LayerNorm(input + MultiHead(input_q, input_k, input_v)) \quad (2.4)$$

$$enc = LayerNorm(selfAtt + FFN(selfAtt)) \quad (2.5)$$

where $input_q = input_k = input_v$ are the input of the encoder layer. These arguments are the same, since we are applying self-attention here.

Decoder Similar to the encoder the decoder is a stack of N layers. Compared to the encoder it adds a third sublayer that applies multi-head cross attention to the output from the encoder layers and the first sublayer of the decoder. In the first self-attention sublayer we mask out earlier positions in the output sequence to stop predictions for a certain position i to depend on the outputs at the positions before position i. The residual connections and layer normalisation on each sublayer is identical to the architecture of the encoder. This results in the following calculations for each decoder layer:

$$selfAtt = LayerNorm(input + MaskedSelfAtt(input_q, input_k, input_v)) \quad (2.6)$$

$$crossAtt = LayerNorm(selfAtt + CrossAtt(selfAtt_q, enc_k, enc_v)) \quad (2.7)$$

$$dec = LayerNorm(crossAtt + FFN(crossAtt)) \quad (2.8)$$

where $MaskedSelfAtt$ is the function that applies the masking before $MultiHead$, $input_q = input_k = input_v$ are the input of the decoder layer and enc_k and enc_v the keys and values of the encoder layers. $selfAtt_q$ are the queries from the first attention function.

The final layers consist of a linear fully connected neural network and a softmax layer. This allows us to have a higher number of possible outputs than the size of the input and output embeddings since it projects the output of the decoder part into a probability vector.

2.2.2 Vision transformers

As mentioned in the previous section transformers were originally meant for text problems. The success of the transformer in that context is what made Dosovitskiy et al. [17] start exploring the usage of the attention-based model for computer vision tasks as well.

Their approach tries to use the original transformer implementation as much as possible. This means that they had to introduce an intermediate step to be able to use the images as input for the model.

The original model is designed to receive a 1 dimensional input of token embeddings. The input images for the vision transformers are 2 dimensional and each image has the shape $x \in \mathbb{R}^{H \times W \times C}$ where H and W are the height and width of the image respectively and C is the number of channels. To be able to use the images as input they split the image in square patches with the resolution (P, P). This results in $N = HW/P^2$ patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$.

To obtain the patch embeddings they first flatten the patches and then train a linear projection mapping them to a single dimension. To this sequence of patch embeddings they prepend a learnable embedding. Then to allow for the preservation the positions of the patches, they add positional embeddings to the patch embeddings.

2.2.3 Linear transformers

Since transformers use attention to process so many parts of the images the model has a high memory and time complexity. One of the proposed solutions to reduce the complexity of the models is made by Katharopoulos et al. [21]. They introduce the linear transformer that according to their results can reach similar performance when compared to vanilla transformers with the benefit of being faster.

To see the advantage of their attention function we first need to understand what the bottleneck is in regular transformers. For this we look at equation 2.1 again:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \qquad \text{(2.1 revisited)}$$

Here we apply the softmax function row-wise to $\frac{QK^T}{\sqrt{d_k}}$. This means that the complexity of the multiplication of the three matrices Q, K^T and V becomes $O(S^2 d_k)$.

The approach by Katharopoulus et al. starts by rewriting the attention calculation in equation 2.1 into a generalised equation for any similarity function $sim(Q, K)$.

$$Attention(Q, K, V) = V' \tag{2.9}$$

Given that V_i returns the i-th row of V' as a vector:

$$V'_i = \frac{\sum_{j-1}^{N} sim(Q_i, K_j)V_j}{\sum_{j-1}^{N} sim(Q_i, K_j)} \tag{2.10}$$

This equation is equivalent to equation 2.1 when we say $sim(q, k) = exp(\frac{q^T k}{\sqrt{D}})$.

We need to set a constraint on $sim(q, k)$ for this equation to be able to define an attention function. And that is that it has to be non-negative.

Now given some kernel with a feature representation $\phi(x)$ we can rewrite equation 2.10 as follows:

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)} \tag{2.11}$$

This is possible by using the kernel trick[1]. We can avoid learning a non-linear function using this trick. It allows us to rewrite our similarity function as follows:

$$sim(q, k) = \phi(q)^T \phi(k) \tag{2.12}$$

We can then simplify equation 2.11 using the associative property of matrix multiplication to:

$$V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)} \tag{2.13}$$

In [21] they note that the feature function $\phi(x)$ that corresponds to the exponential kernel exp is infinite dimensional. This makes it infeasible to linearise softmax attention itself. Therefore they choose the following feature map:

$$\phi(x) = elu(x) + 1 \tag{2.14}$$

with $elu(x)$ being the exponential linear unit activation function [12]:

$$elu(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \tag{2.15}$$

where $\alpha = 1.0$. This results in a complexity of $O(Sd_k)$ which is no longer quadratic when compared to softmax attention.

[1] https://en.wikipedia.org/wiki/Kernel$_m ethod Mathematics :_t he_k ernel_t rick$

2.3 Image inpainting

To be able to detect anomalies without labelling a large dataset we are using a semi-supervised method using image inpainting.

Inpainting is the process of filling in missing, damaged or censored parts in paintings or images. It can also be used for object removal or image manipulation. Applying the technique on digital images can be done using different approaches.

Figure 2.2: An example of image restoration using inpainting from Bertalmio et al. [7]

In figure 2.2 we see a photo that is restored by using a context based inpainting method.

Our transformer-based approach, which is modeled after the Inpainting Transformer (InTra) from Pirnay et al. [27] learns to paint regions that are removed from the original images. This allows the models to fully reconstruct an image based on the surrounding patches. This approach is discussed more extensively in section 4.2.

2.4 Anomaly detection

Anomaly detection is the detection of outliers, points that have extreme values compared to the rest, in a dataset. This kind of detection can be useful in different environments. Examples are intrusion detection in security and fault detection in industrial systems.

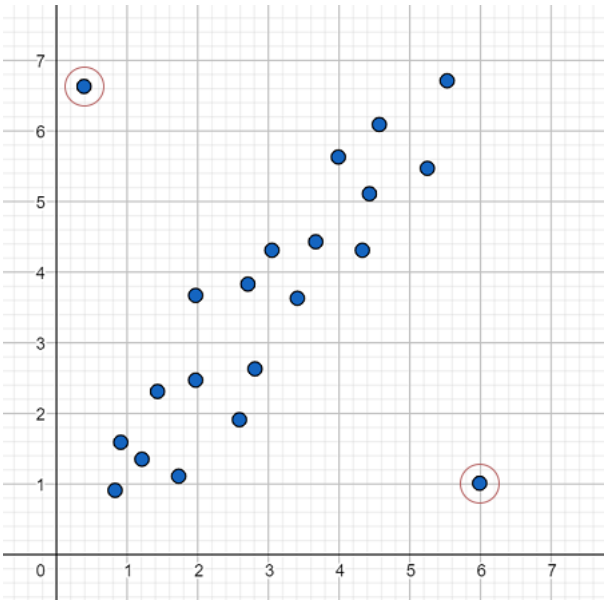


Figure 2.3: An example of outliers in a scatter plot

In our case we are looking at anomalies in manufacturing. We want to find defects in images of objects and textures.

2.5 Image similarity

To allow us to compare inpainted images to original samples from a dataset we want to use objective measures that try to approach a human visual system. For our research we use two measures specifically: Gradient Magnitude Similarity [39, 45] and Structured Similarity Index [35].

2.5.1 Structured Similarity Index

This metric uses three features from an image to be able to make a comparison: luminance, contrast and structure. These features are taken from two images r and d.

The luminance is an estimation of the mean intensity:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.16}$$

Using μ_r and μ_d we can now make a comparison function $l(r, d)$.

$$l(r,d) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.17}$$

C_1 is a constant that avoids instability when $\mu_x^2 + \mu_y^2$ is almost zero. In [35] they choose $C_1 = (K_1 L)^2$ with L being the range of the pixel values and K_1 as a small constant.

Next we make an estimation of the contrast for both images:

$$\sigma_r = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_r)^2 \right) \tag{2.18}$$

Using σ_r and σ_d we can now make a comparison function $c(r, d)$.

$$c(r, d) = \frac{2\sigma_r \sigma_d + C_2}{\sigma_r^2 + \sigma_d^2 + C_2} \tag{2.19}$$

In [35] they choose $C_2 = (K_2 L)^2$ with K_2 as a small constant.

The structure is measured as the covariance divided by the sum of the standard deviation of the images:

$$\sigma_{rd} = \frac{1}{N-1} \sum_{i=1}^{N} (r_i - \mu_r)(d_i - \mu_d) \tag{2.20}$$

$$s(r, d) = \frac{\sigma_{rd} + C_3}{\sigma_r + \sigma_d + C_3} \tag{2.21}$$

When we combine these similarity measures of the images the eventual SSIM function looks like this:

$$SSIM(x, y) = \frac{(2\mu_r \mu_d + C_1)(2\sigma_r d + C_2)}{(\mu_r^2 + \mu_d^2 + C_1)(\sigma_r^2 + \sigma_d^2 + C_2)} \tag{2.22}$$

2.5.2 Gradient Magnitude Similarity

The gradient magnitude similarity (GMS) score uses gradient magnitude maps of a ground truth image and a reconstruction. These local quality maps (LQM) are used to calculate a final score by pooling the map using the standard deviation.

The local quality map is computed using the pixel-wise similarity of gradient magnitude maps. These gradient magnitude maps of the original image r and reconstructed image d are obtained by convolving both images with Prewitt filters in the horizontal (x) and vertical (y) direction. These filters are defined as follows:

$$h_x = \begin{bmatrix} 1/3 & 0 & -1/3 \\ 1/3 & 0 & -1/3 \\ 1/3 & 0 & -1/3 \end{bmatrix} \qquad h_y = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 \\ -1/3 & -1/3 & -1/3 \end{bmatrix} \tag{2.23}$$

The magnitudes at location i for r and d is denoted as $m_r(i)$ and $m_d(i)$:

$$m_r(i) = \sqrt{(r * h_x)^2(i) + (r * h_y)^2(i)} \tag{2.24}$$

$$m_d(i) = \sqrt{(d * h_x)^2(i) + (d * h_y)^2(i)} \tag{2.25}$$

With these gradient magnitude maps we can now compute the GMS:

$$GMS(i) = \frac{2m_r(i)m_d(i) + C}{m_r^2(i)m_d^2(i) + C} \tag{2.26}$$

where C is a positive constant for stability, like the values used for the SSIM.

Chapter 3

Related Work

This chapter introduces previous research into the two main problems we are dealing with: image inpainting and anomaly detection.

3.1 Image inpainting

Inpainting in images is a subject that was already explored by Bertalmio et a. [7] in 2000. They modeled their algorithm after manual inpainting concepts used by conservators. However, their algorithm still needs user input in the form of a mask of the image sections that have to be inpainted. They also encountered problems filling in larger textured regions.

This is why they followed up their initial paper in 2003 [8] that combined texture synthesis with their previously introduced structure inpainting algorithm by decomposing the input image. This method thus depends on three different types of methods: inpainting, texture synthesis and image decomposition.

In 2004 Criminisi et al. [13] proposed a single algorithm that could fill both structures and textures. They use an algorithm that prioritises patches to fill along a user selected area, these patches are then filled by using the pixels from a similar looking source location. However, this does not handle curved structures very well. They also remark that quantifying the performance of their algorithm is a non-trivial task.

Their work is further extended upon by Bugeau and Bertalmio, by introducing a new algorithm for diffusion and texture synthesis [9]. In later work Bugeau et al. [10] identify three main similarity components: texture synthesis, diffusion and coherence. They try to minimise these components in a new algorithm for inpainting inspired by the PatchMatch algorithm [4]. They note that it may be possible that there are no similar patches in the image when the area that needs to be painted is large. Which means that their approach gives poor results for these kinds of situations.

All previously mentioned work still require the section to be inpainted

to be marked prior to applying an algorithm. This non-blind inpainting is addressed by Xie et al. [37] who introduce a deep neural network based approach that continues previous work on denoising [23] and blind inpainting [16]. For this they use stacked denoising autoencoders. However, their method relies on supervised learning and is mostly focused on removing small noise from input images.

An approach that focuses on semantic inpainting of larger regions was introduced by Yeh et al. [40]. Their approach uses a generative adversarial network based model that is trained to give realistic image results. The predictions here are also limited by the network and the training procedure. This means that it shows promising results but may not be applicable to more complex structures. This is also true for an approach using a patch-based GAN [14] which focuses on higher resolution images, which is not the case for the context based approach from [26].

Another GAN-based approach [41] introduces a contextual attention layer into a model that uses both local and global loss for the GAN. The contextual attention layer learns where to borrow or copy information to create reconstructions. This especially improves the inpainting of larger regions.

Building upon this contextual attention approach both Yu [43] and Pirnay [27] use transformers for inpainting. Both use positional embeddings. The first approach focuses on realistic reconstructions of landscapes and faces using texture generation. The approach by Pirnay is mainly focused on usage for anomaly detection and only focuses on reconstructing one single type of image, which would most likely make it unsuitable for the images used by Yu.

3.2 Anomaly detection

The subject of anomaly detection is very large, since anomalies can be found in all kinds of data. In our case we are focusing on anomaly detection in image data. Most notably related work using the MVTec AD dataset [5].

The MVTec AD dataset contains images that were specifically selected for unsupervised anomaly detection. Having a standardised dataset allows for easier evaluation of novel approaches and makes it possible to compare models by quantifying the performance. This is illustrated by applying existing methods on the new dataset.

One of these methods by Bergmann [6] uses convolutional autoencoders to segment anomalous sections in images after training a model only on good samples. They use both a per-pixel L2 loss and the structural similarity index (SSIM) to create two models. They show that using SSIM as metric improves their results. For MVTec AD this seems the best performing model but both types of autoencoders fail to reconstruct small details.

A different approach that has more problems getting good results uses

GANs. In this case a model by Schlegl et al. [29]. Here the results on MVTec AD have trouble with the images including a lot of variations. The categories that perform better are the bottle and pill images that do not contain any rotations or different shapes.

The last approach that is applicable to all the types of images uses a convolutional neural network for feature discovery. This method by Napoletano et al. [25] was designed for binary classification of images to determine if there is an anomaly or not. To be able to create a course anomaly map the model was applied to smaller patches in the image. This achieves satisfactory results but since the model is applied to the different colour layers separately the anomalies in colour are not detected.

More recently Zavrtanik et al. [44] used the MVTec AD dataset for an inpainting approach using a U-net based encoder-decoder network (RIAD). Just like the convolutional autoencoders mentioned previously the loss function uses the SSIM. They combine this with the multi-scale gradient magnitude similarity [39] to focus on more image properties.

The RIAD approach generally outperforms all the previously mentioned models. And since it uses an inpainting approach it is also the most similar model compared to the inpainting transformer by Pirnay et al. [27] that our work is based upon.

The most recent work with the best results for segmentation and detection we could find using the MVTec AD dataset is [42]. Their approach uses a feature extraction approach that they argue is less complex than [28] and [18] which all have a 98% AUROC for detection and segmentation on the dataset. These approaches are what we consider the current state-of-the-art for the dataset.

Chapter 4

Experimental setup

In this chapter we will describe the dataset used in our experiments, the structure of the model, our metrics and evaluation approach to be able to compare different inpainting transformer based models.

4.1 Data

We use the MVTec AD dataset that we previously mentioned in section 3.2. This dataset is also used by other researchers [6, 27, 44] to compare the performance of methods for anomaly detection. The dataset focuses on industrial anomalies and includes information to evaluate both detection and segmentation of anomalies. The dataset contains 15 types of different images, categorised into textures and objects. Some of the images are photos taken in a single place and orientation, while others contain different rotations or positioning. This variation makes it suitable to evaluate the variety of images a model can be used for. The images with defects in the dataset are labelled by anomaly type and segmentation maps are provided for segmentation of the anomalies in those images.

Figure 4.1: An example of a screw, toothbrush and transistor that are defective.

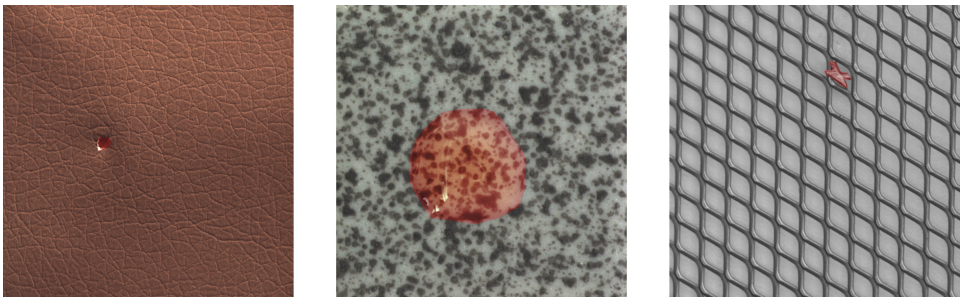


Figure 4.2: An example of a piece of leather, tile and grid that are defective.

In figure 4.1 we see three different object-type images with anamalies. Compared to figure 4.2 where we have examples of textures. These different types of images allow us to test our model to do both detailed texture synthesis and structure inpainting. In the case of anomaly detection the defects on the textured images are smaller and require finer reconstruction of the texture details when compared to the larger objects where the anomalies usually consist of larger deformations or missing elements.

4.2 Model

Our inpainting transformer model is directly based on the inpainting transformer by Pirnay et al. [27]. For completeness we will describe the patch embeddings used for the inpainting problem and give an overview of the model architecture.

4.2.1 Patch embeddings

The model uses a patch based approach to create full reconstructions of images. Each image is split into patches of which one is made blank. The model then learns to inpaint that blank patch based on the surrounding patches. The creation of the patches is similar to the approach discussed in section 2.2.2. An overview of these steps can be found in figure 4.3a.

We start by defining our images as $x \in \mathbb{R}^{H \times W \times C}$ where H is the height, W the width and C the number of channels. We create square patches of the size (P, P). This means that we can split each image into a grid of $M \times N$ patches where $M = \frac{W}{P}$ and $N = \frac{H}{P}$:

$$x_p \in \mathbb{R}^{(M \times N) \times (P^2 \cdot C)} \tag{4.1}$$

$$x_p^{(i,j)} \in \mathbb{R}^{P^2 \cdot C} \tag{4.2}$$

with (i, j) denoting the location of the patch within in the image.

We then want a square window of patches of length L that is smaller than the image. From this window we can the pick any patch that should

be inpainted based on the other patches in the same window. In [27] they formulate this inpainting problem as follows:

> Let $(x_p^{(i,j)})_{(i,j) \in S}$ be a square subgrid of patches defined by some index set $S = r, ..., r + L - 1 \times s, ..., s + L - 1$. Here L is the side length of the window, and (r, s) is the grid position of the upper left patch in the window. If $(t, u) \in S$ is the position of some patch, the formal task to inpaint (t, u) is to approximate the patch $x_p^{(t,u)}$ using only the content and the positions of all other patches $(x_p^{(i,j)})_{(i,j) \in S \setminus (t,u)}$ in the window.

This means that for the transformers to able to determine the position of the patches we need to include information of the position of a patch $x_p^{(i,j)}$. For this we calculate a one dimensional value:

$$f(i, j) = (i - 1) \cdot N + j \tag{4.3}$$

As a last step we need to reshape this patch window into an input sequence for our transformer-based model. We do this by creating a mapping in some latent space of dimension D. Which means that for every patch $(x_p^{(i,j)})_{(i,j) \in S \setminus (t,u)}$ we create and embedding $y^(i, j)$ and for the patch we want to inpaint we add one single embedding $x_{inpaint} \in \mathbb{R}^D$ to the position embedding.

$$y^{(i,j)} = x_p^{(i,j)} E + posemb(f(i, j)) \in \mathbb{R}^D \tag{4.4}$$

$$z = x_{inpaint} + posemb(f(t, u)) \in \mathbb{R}^D \tag{4.5}$$

where $E \in \mathbb{R}^{(K^2 \cdot C) \times D}$ and *posemb* is a standard learnable one-dimensional position embedding.

The final input sequence for the model is then formed by z and $y^{(i,j)}$ for each $(i, j) \in S \setminus (t, u)$.

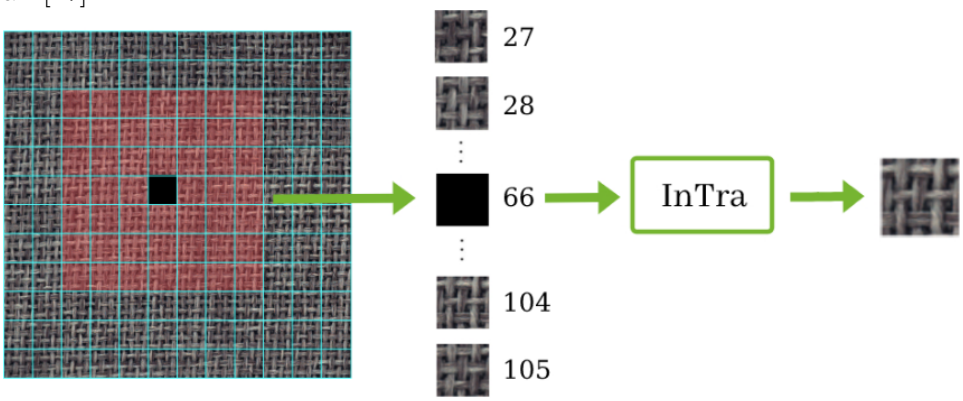
4.2.2 Architecture

The architecture for our inpainting model uses n blocks of the transformer encoder that are stacked. These encoders consist of an attention function and a fully-connected feed-forward network. This follows the approach of the Vision Transformer discussed in section 2.2.2. The structure of this model is visualised in figure 4.4.

We create two different versions of our model.

- A version using the original multi-head self-attention (MSA) from [33] that we discussed in section 2.2.1.
- A version using the multi-head linearised self-attention (MLSA) from [21] that we discussed in section 2.2.3.

Figure 4.3: An overview of the inpainting transformer steps by Pirnay et al. [27]



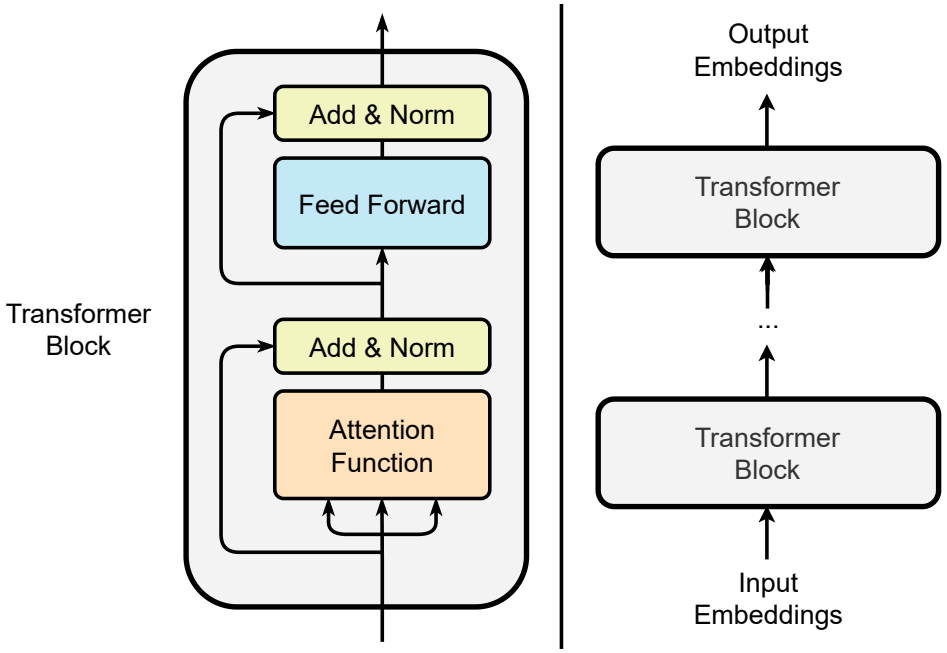


Figure 4.4: The structure of our inpainting model

This differs from the original inpainting transformer (InTra) model in two places: we do not add residual connections between layers and we do not use multi-head feature self-attention (MFSA). Using residual connection would mean adding extra addition computations to the model, which can influence the output of the model and skew the results in the direction of one of the attention functions. With MFSA we would introducing an extra multilayer perceptron in one of the attention functions which would strongly

increase the size of the model in only one case, since the MLSA model would not use this multilayer perceptron. Using MSA makes it simpler to compare the results with MLSA.

4.3 Loss function

To be able to quantify the performance of our models during training we require metrics that are suitable for the inpainting task that we are performing. This means that we need a loss function that takes into account that our dataset contains both textures and objects. For this we use a combination of structural similarity, gradient magnitude similarity and a pixel-wise L2 loss.

Since we try to recreate part of the model from [27] we also use a similar loss function. Which is equal to the one used in [44].

Our loss function L consists of three parts. The first part is a pixel-wise L_2 loss. This does not take into account perceptual differences, since it assumes that all pixels in the images are independent. Therefore we use the structured similarity index (SSIM) [35] and the multi-scale gradient magnitude similarity (MSGMS) [39, 45] that we discussed in section 2.5.

The SSIM is a metric that looks at dependencies between regions of an images by including luminance, contrast and structural information. This means that it assumes that our model is able to find those structures. MSGMS is similar in that it looks at local image quality but does not focus on those structures.

Before we determine the full loss function we first formulate these three base parts given our original patch P and the reconstruction $\hat{P}$ for that patch.

$$L_{SSIM}(P, P) = \frac{1}{N_p} \sum_{x=1}^{W} \sum_{y=1}^{H} 1 - SSIM_{(x,y)}(P_l, \hat{P}_l) \tag{4.6}$$

where N_p is the number of pixels of the patch P. The $SSIM_{(x,y)}$ is the SSIM result for the two pixel in the patch and the reconstruction with (x, y) being the center.

$$L_{MSGMS}(P, P) = \frac{1}{3} \sum_{l=1}^{3} \frac{1}{l} \sum_{x=1}^{W_l} \sum_{y=1}^{H_l} 1 - GMS_{(x,y)}(P_l, \hat{P}_l) \tag{4.7}$$

where P_l and $\hat{P}_l$ are the scaled versions of the patches and $GMS_{(x,y)}(P_l, \hat{P}_l)$ gives the GMS map of those scaled patches at the pixel location (x, y).

We can then define our complete loss function as

$$L(P, \hat{P}) = L_2(P, \hat{P}) + \alpha\, L_{MSGMS}(P, \hat{P}) + \beta\, L_{SSIM}(P, \hat{P}) \tag{4.8}$$

with α and β being loss weights for the MSGMS and SSIM losses.

4.4 Anomaly detection

To be able to use our model for anomaly detection we need to generate a complete reconstruction of some original input image. This reconstruction can then be used to create a anomaly map where we can locate any anomalies.

4.4.1 Reconstruction

The reconstruction $\hat{x}$ of the image $x \in \mathbb{R}^{H \times W \times C}$ requires us to split the image into the same number of $M \times N$ patches as the images we used to train our models. We can then create a reconstruction by selecting a window with size $L \times L$ per patch $x_p^{(t,u)}$ to create a reconstruction of that section of the image.

The ideal window for each patch is the window location where the (t, u) is in the center. Since we want to create a reconstruction of the full image this is not possible for the patches closer to the sides of the image itself. Therefore we want to calculate an appropriate patch window with patch $x_p^{(r,s)}$ in the upper-left corner.

$$pad(x) = max(1, x - \lfloor \frac{L}{2} \rfloor) \tag{4.9}$$

$$r = pad(t) - max(0, pad(t) + L - N - 1) \tag{4.10}$$

$$s = pad(u) - max(0, pad(u) + L - M - 1) \tag{4.11}$$

Reconstructing the image can now be done by using the calculated windows for each patch. This gives us the fully reconstructed image $\hat{x}$.

4.4.2 Anomaly map

With both our original image x and the reconstruction $\hat{x}$ we can create an anomaly map that we can use to detect and locate anomalies in x. For this we use similar multi-scale gradient magnitude similarity calculations that we also used for our loss function in section 4.3.

Instead of a single value we want a complete per-pixel map thus we adapt the calculation slighty to create a per-pixel average of the GMS maps at different scales. Similar to the approach in [27, 44].

$$MSGMS(I, \hat{I}) = \frac{1}{3} \sum_{l=1}^{3} GMS(I_l, \hat{I}_l) \tag{4.12}$$

Since anomalies are usually located in larger connected regions of an image we can aggregate the error of the reconstruction over a larger space by post-processing the MSGMS map using a mean-filter convolution. The filter reduces the intensity variety between pixels and reduces noise. This smoothing prevents a faulty detection when high values are present in small regions of the MSGMS map. These values are more likely to be failed reconstructions or background noise than actual anomalies. This gives us a difference map $diff(I, \hat{I}) \in \mathbb{R}^{H \times W}$.

$$diff(I, \hat{I}) = 1_{H \times W} - (MSGMS(I, \hat{I}) * mean) \tag{4.13}$$

where $mean$ is a mean filter of (21×21) and $*$ the convolution operation. The final anomaly map for the image I is calculated by using the square deviation of the difference map of the image versus the difference map of all the images in the training data T.

$$anomap(I) = \left(diff(I, inp(I)) - \frac{1}{|T|} \sum_{t \in T} diff(t, inp(t)) \right)^2 \tag{4.14}$$

where inp denotes the reconstruction operation done by our transformer model.

4.5 Training

Using the building blocks given in the previous sections we train one MSA and one MLSA model per image in the MVTec AD dataset. This way we can compare the result of the different MSA and MLSA attention functions.

The parameters we used during training are mostly the same as those in [27]. A summary is given in table 4.1.

For training we use the images without anomalies from MVTec AD. Each set of training data contains a different number of images and no predefined validation set. This is why we randomly take 10% or 20 images from the training data to be able to check the patches that our models generate.

The rest of the training data is used to extract 600 random patch windows per epoch per images. This increases the amount of training data and shuffles the input.

Other parameters define the size and number of patches: patch size P, window size L and the size of the images W and H. Since all images in MVTec AD are square this means that $W = H$. These parameters are set to $P = 16$ and $L = 7$. The image sizes differ per image: 256×256, 320×320, 512×512. These were chosen by Pirnay et al. as small as possible without removing details from the patches.

We use a latent dimension of $D = 512$. Both the MSA and MLSA models consist of 13 transformer blocks that have 8 attention heads. This gives us

Table 4.1: Summary of all variables

Parameter	Description	Value
P	Patch size	16
L	Window size	7
W	Image width	Value per image type
H	Image height	Equal to W
α	Weight of the MSGMS loss	0.01
β	Weight of the SSIM loss	0.01
D	Model dimension	512
Batch size		256
Number of layers	Number of transformer blocks	13
Number of heads (h)	Number of heads in the multi-head attention	8
Learning rate	Parameter of the Adam optimiser	0.001
Early stopping		150 epochs
Max epochs		20000

$41,374,976$ trainable parameters. The individual loss weights are set to $\alpha = \beta = 0.01$ and all models are trained with the same Adam optimizer with the learning rate set to 0.001. The batch size is set to 256 and we do not use dropout, since this was not used in the original paper.

To be able to compare differences in the training time and to make sure our models fully converge we train our models for a maximum number of 20000 epochs, which is virtually unlimited since we combine this with early stopping of 150 epochs. This means that we will stop training when do not observe a lower value for the validation loss in 150 consecutive epochs. The best model is then chosen based on the lowest validation loss.

Each model is trained by submitting a job on a SLURM cluster. The jobs are assigned 4 CPU cores, 1 GPU and maximum 16 GB of memory. The nodes in the cluster are outfitted with 2x Intel Xeon Silver 4214, 8x NVIDIA GeForce RTX 2080 Ti and 128GB of memory.

4.6 Evaluation

To be able to compare two different transformer models we will use the test images from the MVTec AD dataset.

The score used is the AUROC, which is standard for visual anomaly tasks [5, 22, 27, 29, 32, 38, 44].

For this we will use the anomaly map $anomap(I)$ we obtained from the images in section 4.4 and compare the value in the anomaly map with the values in the image masks that are in the dataset.

The AUROC is calculated by calculating the receiver operating characteristic (ROC). This is a curve that plots the true positive rate against the

true negative rate. The area under the receiving receiver operating characteristic (AUROC) is the area under the plotted curve.

We want to evaluate both the anomaly detection on the image-level and the anomaly segmentation on the pixel-level. For the last task we can directly use the anomaly map. For the image-level detection we take the maximum pixel value from the anomaly map as a single anomaly score for the whole image.

Chapter 5

Results

In this chapter we will try to answer our subquestions using the results we obtained[1] after training and testing our models.

5.1 Detection & Segmentation

What is the difference between softmax and linear attention in detection of anomalies across images with different textures and objects?

The AUROC % scores of our MSA and MLSA models are given by table 5.1. The best models for each image have been indicated by bold text.

This shows that for detection the MLSA-based models outperform the MSA-based models the majority of the textured images. The wood and leather categories are the exceptions. The difference between the MSA and MLSA models is particularly high in the carpet, grid and leather categories.

The results for detection on the object images are split equally over MSA and MLSA. The values for the cable and pill categories are very low, especially when compared to the other images.

What is the difference between softmax and linear attention in segmentation of anomalies across images with difference textures and objects?

The results for segmentation across the textured images is very similar when compared to the numbers for detection. MSA seems to perform worse for three out of the five textured images. The same is true for the object images. The differences for the capsule, hazelnut and metal nut between MSA and MLSA are especially high. MLSA has a higher score for all these cases with at least a difference of 7%. Interestingly MSA seems to perform better on average when the image size is larger.

Let us analyse results for some specific cases with low and high model performance. Looking at the figure 5.1 we see that the mask and the seg-

[1] https://tensorboard.dev/experiment/1YZvra2iQseNTMWDeBFREw/

mentations of the anomaly for the leather category do not match, only the center of the anomaly is correctly localised. This explains the low AUROC value. Looking at the reconstructions in figure 5.2 we see that both the models using MSA and MLSA are missing details, which means that the average training error used to calculate the anomaly map affects the results. For the better performing grid category we see in figure 5.3 that the anomalies are correctly localised, with the MSA model having more noise in the anomaly map around the sides.

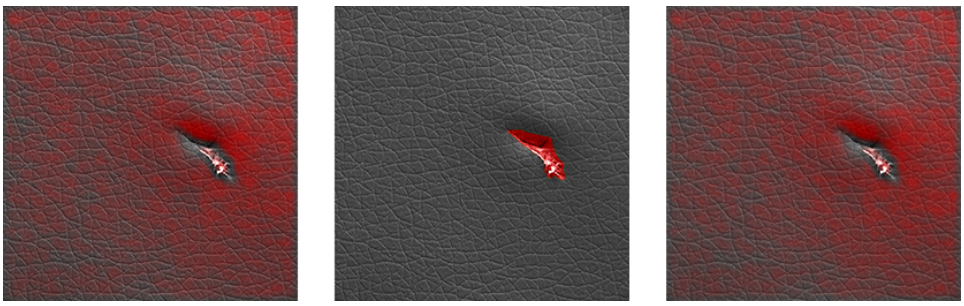


Figure 5.1: Example of an anomalous image from the leather category comparing MSA, the dataset mask and MLSA

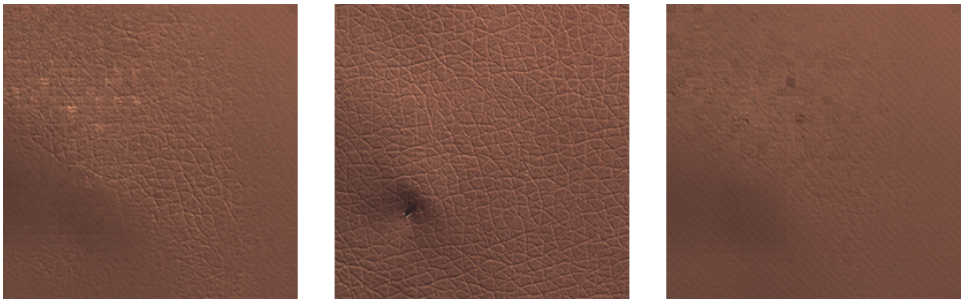


Figure 5.2: Example of reconstructions from the leather category comparing MSA, the original and MLSA

It is interesting to see that the models do not perform the best for the majority of the images for both detection and segmentation, but only for one of these tasks. Since the score for detection depends on the anomaly map that is used to calculate the score for segmentation we would expect the segmentation score to directly relate to the detection score. However, the detection score is only a maximum pixel value which means it only represents one pixel of the anomaly map not taking into account the other values of the anomaly map. The segmentation score does look at all pixels.

Overall our results show that MLSA has better performance than MSA,

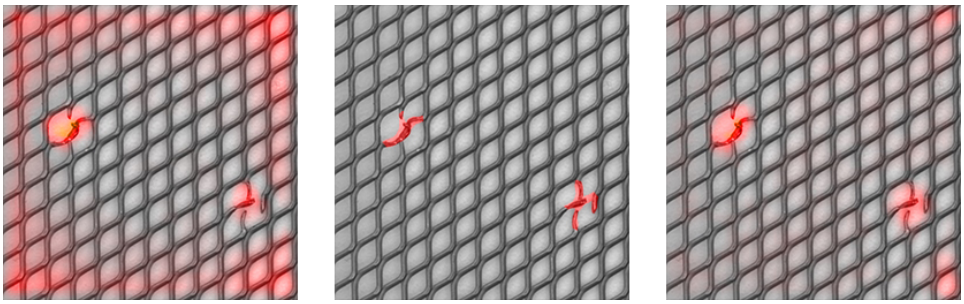


Figure 5.3: Example of an anomalous image from the grid category comparing MSA, the dataset mask and MLSA

both for segmentation and detection on textured and object images. And for detection on texture images.

5.2 Efficiency

How does the number of epochs required to get the best result compare across images with different textures and objects?

The number of epochs required depends on the type of image. Image that have more complex structures require more epochs. We see this especially in the texture category and for the cable, screw and transistor images. Another factor affecting the number of epochs is the amount of colours used. This is shown by the pill and zipper images that both require a very low number of epochs and primarily consist of black, white and grey colours. We give some samples of the image categories in figure 5.4.

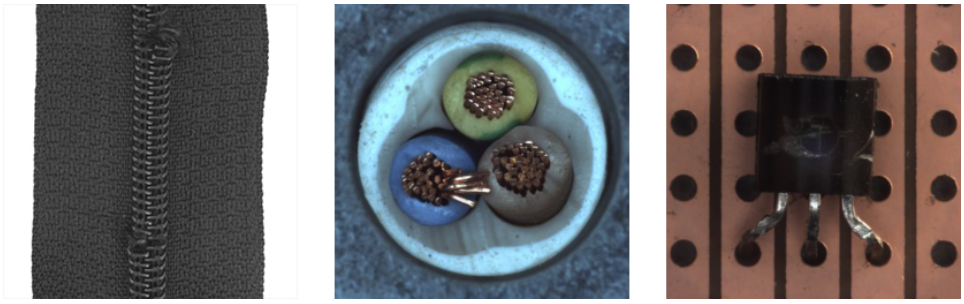


Figure 5.4: Examples of the zipper, cable and transistor image types

How does the training time compare between the softmax and linear attention models across images with different textures and objects?

We see that the time per epoch is lower for MSA in all categories except for the pill and transistor images. With this information we would expect the MSA models to also finish training faster. The opposite is true however.

Category	MSA		MLSA		Image size
	Segmentation	Detection	Segmentation	Detection	
Carpet	90.44	77.09	**91.52**	**86.40**	512
Grid	89.82	76.61	**93.39**	**89.97**	256
Leather	49.99	**86.85**	**56.23**	75.44	512
Tile	**79.31**	71.32	73.33	**73.92**	512
Wood	**70.22**	**77.54**	62.84	72.72	512
Texture average	**75.96**	77.88	75.46	**79.69**	
Bottle	65.82	**91.51**	**68.65**	89.92	256
Cable	88.04	38.17	**90.07**	**43.65**	256
Capsule	81.66	**72.32**	**88.43**	66.33	320
Hazelnut	67.94	**55.89**	**81.59**	53.82	256
Metal nut	54.99	60.95	**67.51**	**78.49**	256
Pill	**88.60**	44.82	88.45	**51.28**	320
Screw	**97.31**	55.69	97.23	**65.32**	320
Toothbrush	80.42	86.67	**81.78**	**87.22**	256
Transistor	77.72	**59.25**	**79.95**	56.71	256
Zipper	**81.22**	**79.07**	79.66	78.07	512
Object average	78.37	64.43	**82.33**	**67.08**	
All average	80.02	66.46	**81.32**	**70.00**	

Table 5.1: Results in AUROC % for segmentation and detection

Category	MSA			MLSA		
	Best epoch	Time/Epoch	Training time	Best epoch	Time/Epoch	Training time
Carpet	**77**	**0:58:05**	**220:42:00**	138	1:07:17	265:47:00
Grid	140	**0:26:48**	130:00:00	**53**	0:33:34	**114:06:00**
Leather	142	0:43:50	214:03:00	142	**0:43:33**	**213:24:00**
Tile	**353**	0:36:40	**308:01:00**	555	**0:39:38**	466:23:00
Wood	149	0:53:56	270:33:00	**124**	**0:56:16**	**258:49:00**
Bottle	37	**0:24:11**	75:47:00	**11**	0:24:33	**66:16:00**
Cable	250	**0:39:59**	267:55:00	**211**	0:48:07	**267:05:00**
Capsule	27	**0:41:38**	123:30:00	**14**	0:41:45	**114:50:00**
Hazelnut	83	**1:22:12**	321:56:00	**19**	1:25:10	**242:43:00**
Metal nut	148	**0:25:31**	127:10:00	**43**	0:28:22	**91:42:00**
Pill	5	0:48:57	128:06:00	**4**	**0:46:54**	**121:57:00**
Screw	245	**0:40:27**	267:01:00	**211**	0:40:37	**245:04:00**
Toothbrush	58	0:12:19	42:55:00	58	**0:12:10**	**42:24:00**
Transistor	271	0:46:40	328:13:00	**196**	**0:44:59**	**260:57:00**
Zipper	5	**0:25:59**	**67:34:00**	5	0:34:52	90:39:00

Table 5.2: Best epochs and time taken to finish training

The number of epochs required for the best result is lower in eleven of the fifteen cases, which also means that the total training time is lower for MLSA.

5.3 Comparison with previous work

How do our results compare to previous work?

When we compare our results those from the original InTra model by Pirnay et al. [27] in table 5.3 we see that the performance of our models is worse than the original model. The only result that is better is the segmentation for the carpet image type. This was to be expected, since we have omitted some parts of the original InTra model that improved their results. We will discuss those choices in chapter 6.

Category	InTra	
	Segmentation	Detection
Carpet	88.2	98
Grid	98.8	100
Leather	99.5	100
Tile	94.4	98
Wood	88.7	97
Texture average	96.1	98
Bottle	97.1	100
Cable	91.0	70
Capsule	97.7	86
Hazelnut	98.3	95
Metal nut	93.3	96
Pill	98.3	90
Screw	99.5	95
Toothbrush	98.9	100
Transistor	96.1	95
Zipper	99.2	99
Object average	96.9	93
All average	96.6	95

Table 5.3: Results from [27]

Chapter 6

Discussion

In our results presented in chapter 5 we see that our model can localise and detect anomalies. However, our models do not perform like those from [27] and some images have particularly low scores. This chapter reflects on those results. We also discuss some of the choices made for the experimental setup introduced in chapter 4.

6.1 Training data

6.1.1 Ground truth masks

In section 4.1 we introduced the MVTec AD dataset for usage in our experiment.

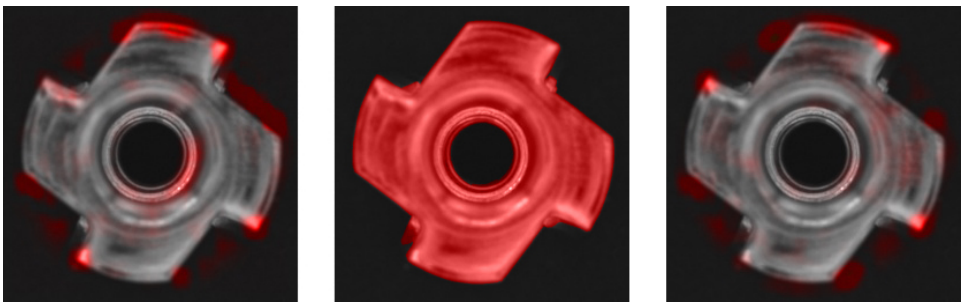


Figure 6.1: Example of an anomalous 'flipped' image from the metal nut category comparing MSA, the dataset mask and MLSA

In figure 6.1 we see an example of the mask compared to the parts of the images indicated as anomalous by our models. Since our models create an anomaly map that is focused on comparing the original anomalous image with a reconstruction we can only mark parts of the image that are different. The centre of these images is not different when compared to non-anomalous images. This means that our models are unable to mark the complete image

as the anomaly. This could explain why the segmentation is relatively low, since the anomaly map is compared to the ground truth that does mark the complete image.

In the case of the transistor category we see a similar result. However, since the category contains multiple situations (rotations, completely missing transistors, non-centred transistors) the effect is less noticeable in the results. In figure 6.2 we can see that the anomaly map only matches the parts that are not present in both the original image and the reconstruction, similar to the anomaly map of the metal nut category.

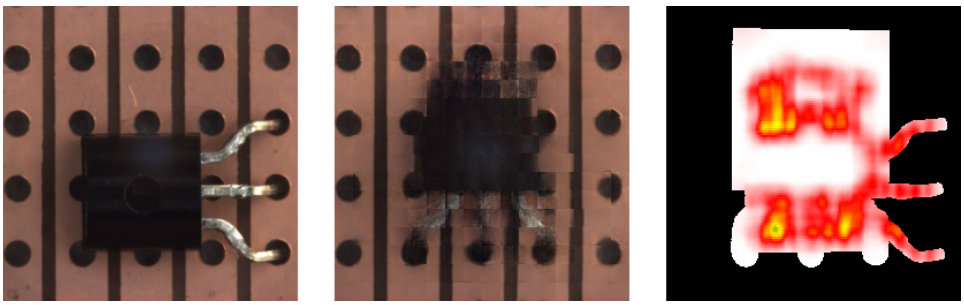


Figure 6.2: Example of an anomalous 'misplaced' image from the transistor category showing the original image, a reconstruction and the anomaly map on top of on the ground truth mask

With this knowledge we would argue that our models would never be able to reach perfect segmentation and these categories should thus be considered as hard cases. The segmentation of these anomalies would require knowledge about the object in the image.

6.1.2 Data augmentations

In [27] the authors mentioned using random rotation and flipping to augment the dataset during training. We have tried to implement both random rotations, vertical and horizontal flipping for our models. However, this would severely impact the results of the models, in most cases blocking them from learning to reconstruct any image. Based on the results from Pirnay et al. we think that the right augmentations could improve training the model. This would require tuning these parameters to find the correct degree of rotations.

6.2 Anomaly map

The evaluation of our models uses the anomaly map introduced in section 4.4.2. Examples of the results produced by equation 4.14 have been shown already in figures 5.1, 5.3, 6.1 and 6.1.

We have seen that some results only show very small parts of our images as anomaly. Since we use equation 4.14 to remove the average reconstruction error we might be losing information. Therefore we compare our anomaly map directly to the MSGMS-map (equation 4.13) in figures 6.3, 6.5 and 6.4.

These images show us that the anomaly map differs from the MSGMS-map. However, the difference is not the same for all the image categories. In the case of the bottle category the anomaly map is missing information. For the carpet category both the MSGMS-map and the anomaly map are similar. Lastly the metal nut shows us a MSGMS-map that is the complete image.

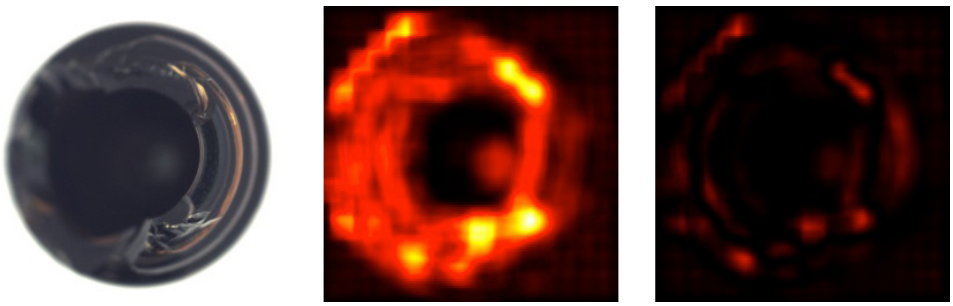


Figure 6.3: Example of an anomalous image from the bottle category showing the original image, the MSGMS-map and the anomaly map

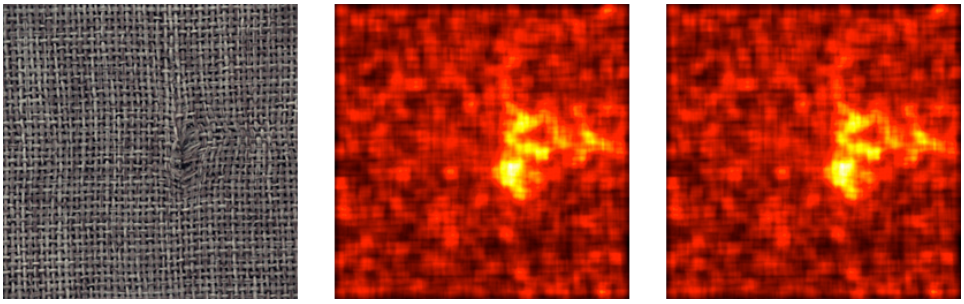


Figure 6.4: Example of an anomalous image from the carpet category showing the original image, the MSGMS-map and the anomaly map

The anomaly map is created by using the average MSGMS-map over the training data. The difference we see could be the result of a high error rate over the good images. Therefore we look at some images in the test data that have no anomalies.

In figure 6.6 and 6.7 we see that for the good images we also have version of a MSGMS-map that could indicate an anomaly that is not present. This means that directly using the MSGMS-map could improve the scores for segmentation. However, since we are using the max pixel value for the

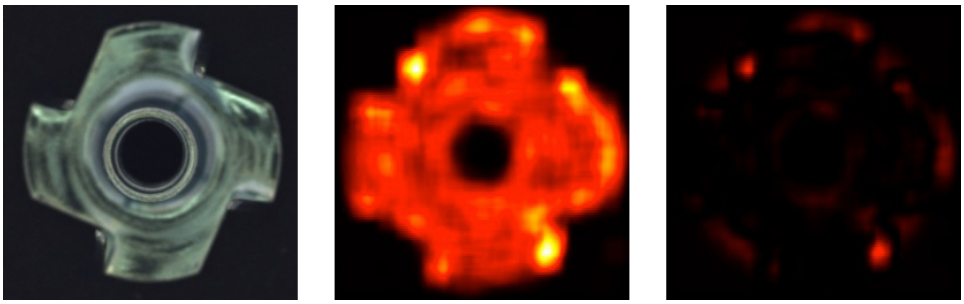


Figure 6.5: Example of an anomalous image from the metal nut category showing the original image, the MSGMS-map and the anomaly map

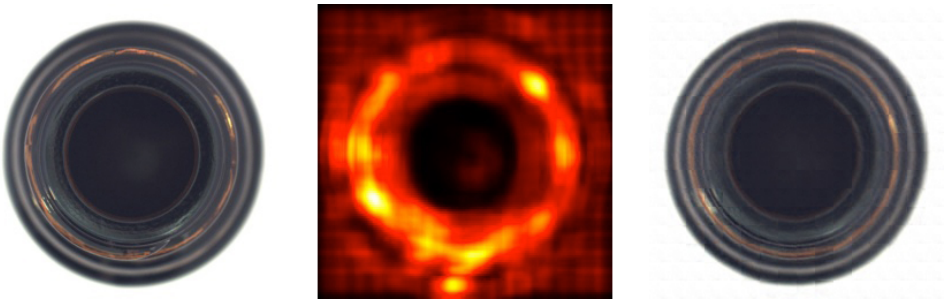


Figure 6.6: Comparison of a good image from the bottle category with the reconstruction and the MSGMS-map

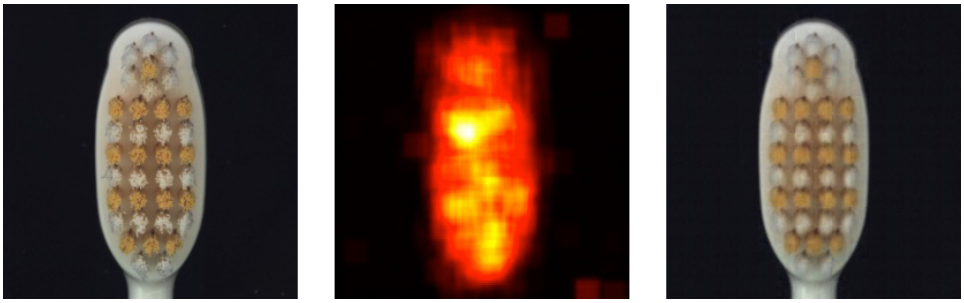


Figure 6.7: Comparison of a good image from the toothbrush category with the reconstruction and the MSGMS-map

detection of anomalies in images this would mean that all images would contain an anomaly. That would severely hurt the performance.

We thus need to use the average error during training to post-process our MSGMS-map that is created by comparing the original images to the reconstructions. The reconstructions that our models make are not sufficiently detailed enough for our MSGMS-map to not spot any differences.

An alternative to using the average training error would be to determine

a threshold. This would allow us to remove values below the threshold. This does require that extra step and would require a different value for each image. In contrast, the current solution can be a compromise for the large range of image types in the dataset.

6.3 Model efficiency

In section 5.2 we answered our research questions regarding the efficiency of the MSA and MLSA models. We saw that the time taken per epoch is lower for MSA when compared to MLSA. However, the paper by Katharopoulos et al. [21] that introduced MLSA reports a significant difference in the time taken for each epoch in favour of MLSA.

Even though the MLSA models require less epochs for training on average, we would still expect the MLSA based models to take less time for each epoch.

The first reason for this difference could be an implementation error of the linear attention function. Our transformers are based on a library[1] by Katharopoulos et al. that contains multiple attention implementations. It allows the user to select a type of transformer, or a part of the transformer structure, and build a model around it. Since we use this library we think it is unlikely that the implementation of the attention function causes the time difference of our model. Moreover, the time difference between running the two attention types is verifiable by running the basic example from the repository. We have done this on the cluster machines, giving us 87.59ms versus 54.43ms for the normal and linear attention functions respectively.

A second cause for the differences could be the code surrounding the attention function. Primarily loading the images from disk and splitting them into patches can be resource intensive. Similarly the loss function can take a longer time to calculate. To be able to figure out the resource usage of each part in our model we would need to run our models with a profiler. This is something that could be a good starting point for future work since a job with a profiler uses a lot more resources, based on the (failed) test runs that we have done.

To make sure that the difference in time per epoch is not something that we only observe for long jobs we have executed a training run with one epoch for the carpet image. This gave us a training time of 1:36:41 for MSA and 1:54:22 for MLSA. Which is a higher difference than the one we see in table 5.2, but still favours MSA as well.

[1] https://github.com/idiap/fast-transformers

Chapter 7

Conclusions

In this thesis we have shown that we can use visual transformers with a linear attention function to tackle an anomaly detection problem using inpainting. Our primary goal was to answer the main research question we asked in the introduction: *What is the effect on the performance and efficiency of using linear transformers in an inpainting context for anomaly detection when compared to regular transformers?*

To answer this question we have successfully implemented two types of models for inpainting: one using regular transformers and one using linear transformers. To evaluate our models they have been trained on the MVTec AD dataset and the results of the inpainting task were used to perform anomaly detection. This required us to calculate anomaly maps using the MSGMS metric that we were able to compare with the masks provided by the dataset.

The results show that linear transformers are able to outperform the regular transformers in the tasks of segmentation and detection. They require less epochs to be able to converge, however in contrast to what we would have expected the time per epoch is longer for these models.

This should answer our primary research question. However, we think that our results should be viewed with caution. It is possible that parts of our implementation affect the resource usage negatively and therefore the time required to train our models.

For future work our models could be revisited to find the resource usage of the different parts like the data loader and metrics functions. An in-depth analysis of the resource usage of these parts could give a better insight to understand our results.

Another change could be adding augmentations to the data loader to see if this would improve the results. As well as hyperparameter optimisation. It might be possible that the different visual transformers can reach better results with other parameters than those that we re-used from other papers. As well as changing the evaluation and the loss function to find better options

that would be better suited for the reconstruction task.

Different approaches using linear transformers could also be explored. For example by using linear transformers for feature extraction in a similar approach like the one introduced in [42].

Apart from using the linear transformers in another context it is also an option to use fastformers [36] or linformers [34] in this context of anomaly detection. This would allow us to evaluate the performance of multiple transformer based approaches to see the effects when compared to regular transformers.

Bibliography

[1] Ali, R., Khan, M. U. K., and Kyung, C. M. Self-Supervised Representation Learning for Visual Anomaly Detection. *arXiv:2006.09654 [cs, eess]* (June 2020). arXiv: 2006.09654.

[2] Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer Normalization. *arXiv:1607.06450 [cs, stat]* (July 2016). arXiv: 1607.06450.

[3] Bahdanau, D., Cho, K., and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]* (May 2016). arXiv: 1409.0473.

[4] Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics 28*, 3 (July 2009), 1–11.

[5] Bergmann, P., Fauser, M., Sattlegger, D., and Steger, C. MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection. *2019* (2019), 9.

[6] Bergmann, P., Löwe, S., Fauser, M., Sattlegger, D., and Steger, C. Improving Unsupervised Defect Segmentation by Applying Structural Similarity to Autoencoders. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* (2019), pp. 372–380. arXiv:1807.02011 [cs].

[7] Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (USA, July 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 417–424.

[8] Bertalmio, M., Vese, L., Sapiro, G., and Osher, S. Simultaneous structure and texture image inpainting. *IEEE Transactions on Image Processing 12*, 8 (Aug. 2003), 882–889. Conference Name: IEEE Transactions on Image Processing.

[9] Bugeau, A., and Bertalmio, M. Combining Texture Synthesis and Diffusion for Image Inpainting. In *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications* (Portugal, 2009), pp. 26–33.

[10] Bugeau, A., Bertalmío, M., Caselles, V., and Sapiro, G. A Comprehensive Framework for Image Inpainting. *IEEE Transactions on Image Processing 19*, 10 (Oct. 2010), 2634–2645. Conference Name: IEEE Transactions on Image Processing.

[11] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]* (Sept. 2014). arXiv: 1406.1078.

[12] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), Nov. 2015. arXiv:1511.07289 [cs] version: 1.

[13] Criminisi, A., Perez, P., and Toyama, K. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing 13*, 9 (Sept. 2004), 1200–1212. Conference Name: IEEE Transactions on Image Processing.

[14] Demir, U., and Unal, G. Patch-Based Image Inpainting with Generative Adversarial Networks. *arXiv:1803.07422 [cs]* (Mar. 2018). arXiv: 1803.07422.

[15] Doersch, C., and Zisserman, A. Multi-task Self-Supervised Visual Learning. *arXiv:1708.07860 [cs]* (Aug. 2017). arXiv: 1708.07860.

[16] Dong, B., Ji, H., Li, J., Shen, Z., and Xu, Y. Wavelet frame based blind image inpainting. *Applied and Computational Harmonic Analysis 32*, 2 (Mar. 2012), 268–279.

[17] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]* (June 2021). arXiv: 2010.11929.

[18] Gudovskiy, D., Ishizaka, S., and Kozuka, K. CFLOW-AD: Real-Time Unsupervised Anomaly Detection with Localization via Conditional Normalizing Flows, July 2021. arXiv:2107.12571 [cs].

[19] Han, C., Rundo, L., Murao, K., Noguchi, T., Shimahara, Y., Milacski, Z. , Koshino, S., Sala, E., Nakayama, H., and Satoh,

S. MADGAN: unsupervised medical anomaly detection GAN using multiple adjacent brain MRI slice reconstruction. *BMC Bioinformatics 22*, 2 (Apr. 2021), 31.

[20] He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385.

[21] Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. *arXiv:2006.16236 [cs, stat]* (Aug. 2020). arXiv: 2006.16236.

[22] Li, C.-L., Sohn, K., Yoon, J., and Pfister, T. CutPaste: Self-Supervised Learning for Anomaly Detection and Localization. pp. 9664–9674.

[23] Mairal, J., Elad, M., and Sapiro, G. Sparse Representation for Color Image Restoration. *IEEE Transactions on Image Processing 17*, 1 (Jan. 2008), 53–69. Conference Name: IEEE Transactions on Image Processing.

[24] Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, Cambridge, MA, USA, Aug. 2012.

[25] Napoletano, P., Piccoli, F., and Schettini, R. Anomaly Detection in Nanofibrous Materials by CNN-Based Self-Similarity. *Sensors 18*, 1 (Jan. 2018), 209. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[26] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. Context Encoders: Feature Learning by Inpainting. Tech. Rep. arXiv:1604.07379, arXiv, Nov. 2016. arXiv:1604.07379 [cs] type: article.

[27] Pirnay, J., and Chai, K. Inpainting Transformer for Anomaly Detection. *arXiv:2104.13897 [cs]* (Sept. 2021). arXiv: 2104.13897.

[28] Roth, K., Pemula, L., Zepeda, J., Schölkopf, B., Brox, T., and Gehler, P. Towards Total Recall in Industrial Anomaly Detection, June 2021. arXiv:2106.08265 [cs] version: 1.

[29] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *Information Processing in Medical Imaging* (Cham, 2017), M. Niethammer, M. Styner, S. Aylward, H. Zhu, I. Oguz, P.-T. Yap, and D. Shen, Eds., Lecture

Notes in Computer Science, Springer International Publishing, pp. 146–157.

[30] Shashikar, S., and Upadhyaya, V. Traffic surveillance and anomaly detection using image processing. In *2017 Fourth International Conference on Image Information Processing (ICIIP)* (Dec. 2017), pp. 1–6.

[31] Susto, G. A., Terzi, M., and Beghi, A. Anomaly Detection Approaches for Semiconductor Manufacturing. *Procedia Manufacturing 11* (Jan. 2017), 2018–2024.

[32] Tsai, D.-M., and Jen, P.-H. Autoencoder-based anomaly detection for surface defect inspection. *Advanced Engineering Informatics 48* (Apr. 2021), 101272.

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention Is All You Need. *arXiv:1706.03762 [cs]* (Dec. 2017). arXiv: 1706.03762.

[34] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-Attention with Linear Complexity. *arXiv:2006.04768 [cs, stat]* (June 2020). arXiv: 2006.04768 version: 3.

[35] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing 13*, 4 (Apr. 2004), 600–612.

[36] Wu, C., Wu, F., Qi, T., Huang, Y., and Xie, X. Fastformer: Additive Attention Can Be All You Need. *arXiv:2108.09084 [cs]* (Sept. 2021). arXiv: 2108.09084.

[37] Xie, J., Xu, L., and Chen, E. Image denoising and inpainting with deep neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (Red Hook, NY, USA, Dec. 2012), NIPS'12, Curran Associates Inc., pp. 341–349.

[38] Xie, Q., Zhang, P., Yu, B., and Choi, J. Semisupervised Training of Deep Generative Models for High-Dimensional Anomaly Detection. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–10. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[39] Xue, W., Zhang, L., Mou, X., and Bovik, A. C. Gradient Magnitude Similarity Deviation: A Highly Efficient Perceptual Image Quality Index. *IEEE Transactions on Image Processing 23*, 2 (Feb. 2014), 684–695. Conference Name: IEEE Transactions on Image Processing.

[40] Yeh, R. A., Chen, C., Lim, T. Y., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. Semantic Image Inpainting with Deep Generative Models. *arXiv:1607.07539 [cs]* (July 2017). arXiv: 1607.07539.

[41] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., and Huang, T. S. Generative Image Inpainting with Contextual Attention. *arXiv:1801.07892 [cs]* (Mar. 2018). arXiv: 1801.07892 version: 2.

[42] Yu, J., Zheng, Y., Wang, X., Li, W., Wu, Y., Zhao, R., and Wu, L. FastFlow: Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows, Nov. 2021. arXiv:2111.07677 [cs].

[43] Yu, Y., Zhan, F., Wu, R., Pan, J., Cui, K., Lu, S., Ma, F., Xie, X., and Miao, C. Diverse Image Inpainting with Bidirectional and Autoregressive Transformers. In *Proceedings of the 29th ACM International Conference on Multimedia* (Virtual Event China, Oct. 2021), ACM, pp. 69–78.

[44] Zavrtanik, V., Kristan, M., and Skočaj, D. Reconstruction by inpainting for visual anomaly detection. *Pattern Recognition 112* (Apr. 2021), 107706.

[45] Zhang, B., Sander, P. V., and Bermak, A. Gradient magnitude similarity deviation on multiple scales for color image quality assessment. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Mar. 2017), pp. 1253–1257. ISSN: 2379-190X.