CHALMERS
UNIVERSITY OF TECHNOLOGY

Unsupervised machine learning used for anomaly detection in x-ray images

Bachelor's thesis in Computer Science and Engineering

Daniel Stribrand

2020-06-16

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

BACHELOR'S THESIS

Unsupervised machine learning used for anomaly detection in x-ray images

Daniel Stribrand

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Unsupervised machine learning used for anomaly detection in x-ray images
Daniel Stribrand

Examiner: Peter Lundin, Department of Computer Science and Engineering
Supervisor: Ulf Norell, Department of Computer Science and Engineering
Advisor: Alexander Grima, GKN Aerospace Sweden AB

Department of Computer Science and Engineering
Chalmers University of Technology / University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 (0)21-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden 2020

Unsupervised machine learning used for anomaly detection in x-ray images

Daniel Stribrand
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

To have computers understand images has long been a hard task due to the largely variable composition of data, but advancements in the field of computer vision has given capable technologies of managing it. The primary technology is the usage of machine learning, where convolutional neural networks has shown to be really potent. This project has tackled the task of detecting minuscule cavities in massive x-ray images of welded metal. The challenge lies both within processing and providing data so that it favors the machine learning model, but also developing models that are complex enough so that it yields a general understanding of the underlying data structure. This project utilizes the benefits of unsupervised machine learning by using a convolutional autoencoder that reconstructs inputs through encoding and decoding. The reconstruction error of the autoencoder is here the primary measurement for anomaly detection. By having the autoencoder trained on millions of small, non-faulty fragments of the x-ray images, the reconstruction error should be larger for fragments with faulty anomalies than those without. Various statistical methodologies have been evaluated in order to find the most suitable alternative to use on the reconstruction error. While there still is work to be done, worthwhile achievements has been made.

Keywords: machine learning, computer vision, anomaly detection, autoencoders, convolutional neural networks, image analysis, unsupervised learning.

Acknowledgments

I want to show all my thankfulness to my supervisors Alexander Grima and Ulf Norell for helping me through with this task. Without their guidance this project would never be what it became. Working in cooperation with GKN and having them allow me do something as interesting as this with their manufacturing has been sensational. I also want to thank Peter Fridolf who helped me with gathering and understanding the source data.

Also seeing that this was my first experience with using machine learning, I must mention that I am very satisfied with the final results.

Daniel Stribrand, June 2020

Contents

1. Introduction

This section will be dedicated to giving a brief introduction to computer vision, the importance of data science and provide the scope, the objectives and the delimitations of the project.

1.1 Background and motivation

For us humans, understanding visual data comes naturally. It takes us a few seconds to look at an image before we understand what it contains and means. However, this visual understanding is one of our skills that does not translate well to computers. The following comic from xkcd explains this humorously[1].

Though, the idea that computers can not understand images is becoming a bit dated. During the last decade, massive improvements have been made in the field of computer vision. A very popular example is that of self-driving cars which are becoming better at detecting visual objects in their field of view [1]. Even one of the first things I did when I began this project was a tutorial with TensorFlow where you got to classify clothes[2].

What this project has explored is one way of using this type of data science in the industry, and everything tells us that projects like these will become all the more common for solving tasks [2]. Thereby, doing projects, prototypes and experiments with actual data from the

[1]Link to the comic: `https://xkcd.com/1425/`

[2]Link to the tutorial: `https://www.tensorflow.org/tutorials/keras/classification`

industries are a vital part for advancements in the field. This project has primarily been a stepping stone for something much larger.

This project was done in cooperation with GKN Aerospace Sweden AB based in Trollhättan. They are a large manufacturer of components for aviation engines that are used for both aircrafts and space rockets [3].

Over the course of the years they have performed x-ray scanning after certain parts of their engines have been welded. These x-ray scans have then been used by the operators who examines the results to see if there is any cavities in the welding, in order to ensure the quality of the products. Afterwards however, this has lead to the amassing of loads of unused image data, which will be exploited for this project. Since the examination of these x-ray scans can be quite tedious (the images are large and the cavities are minuscule), there is desire to ease this process. The idea is to build a tool using machine learning that can help the operators to more quickly identify errors.

1.2 Scope

Developed a machine learning model that can identify anomalies in the aforementioned x-ray scans. This was achieved by building a convolutional autoencoder that learnt how to reconstruct non-faulty images by being trained on only images without problematic anomalies. By using the reconstruction error of the convolutional autoencoder some basic statistical analysis was done for the anomaly detection.

In order to train the model, there was a need to preprocess the source data.

It was also acceptable that the model detected non-faulty anomalies in the images, since the desire was to provide assistance rather than taking exact decisions.

1.3 Objectives

The main goal was to create and evaluate a produced prototype with the following objectives.

- Build a data pipeline for the x-ray images that reads them, crops them, prepares them and stores the prepared data in a file.
- Develop a convolutional autoencoder that will learn how to reconstruct non-faulty images.
- Analyze the predictions of the autoencoder in order to develop functionality for detection of anomalies.
- Mark in the x-ray images where the anomalies were identified.

The project essentially consisted of three parts: data preparation, development of the machine learning model and a statistical analysis for the anomaly detection.

1.4 Questions

Since the focus was on delivering a prototype, most questions was turned into objectives. However one interesting question remains.

- Is machine learning necessary for solving a task like this? The cornerstone for this project was to use machine learning for detection of anomalies, but is that a must?

1.5 Delimitations

The work will only constitute the development of the prototype, which includes the machine learning model and elementary image processing. Due to time limitations, only the convolutional autoencoder with the decided hyperparameters and a handful of statistical methods for the anomaly detection were tested. Neither did this project deliver a final product, but rather gave the components to create one.

Since the material provided from GKN Aerospace AB (primarily the x-ray images) and the prototype is under protection, this report will not show the source data nor the source code.

2. Theory

The theory needed to fully understand and complete this project is seemingly sparse, therefore one can get into the action relatively fast. This section will thereby attempt to reflect that by keeping the contents relatively streamlined and not be too advanced, so that it does not take away the sincerity of the methodology.

The following three categories will be explained with the necessary depth: using images as a source of data, a very small bit of statistics, and the machine learning. These align with the objectives established earlier in section 1.3.

2.1 Images as data

Data is the most valuable component of machine learning. Without it, the model can not properly train. Just like anything else, images can be used as this source of data. The following sections will explain that more thoroughly.

In this project, x-ray images will be the source of data. However, the only thing specifically needed to be known about x-ray images is that they behave according to the same constraints as every other image. In other words; they use the same data structure with a comparable pixel representation. Most of the time they are also monochrome (i.e. uses only a single color channel).

For the purpose of simplicity, images will from here on refer to two-dimensional, singular, static photographs.

2.1.1 The Digital Imaging and Communications in Medicine standard

Since x-ray images uses the same structure as all other images, they can of course be stored in any of the popular image file formats (e.g. Portable Network Graphics, PNG). However, commonly used for x-ray images is the DICOM-format.

DICOM (Digital Imaging and Communications in Medicine) is a standard that establishes "communication and management of medical imaging information and related data" [4], where this project will specifically be using its file format for communication of media [5]. The standard for the file format very is exhaustive and in no use to discuss extensively, since its usage in this project is negligible.

Practically speaking, a DICOM-file simply encapsulates an information object with its corresponding data elements. Meaning that each and every media object will always coexist together with its relevant information [6]. A simple example could be an x-ray scan of a bone fracture, where this image is then encapsulated and stored in the file together with the patient's name, the date of the scan and the name of the doctor [7].

For this project we are simply interested in the smallest and largest pixel value, the photometric interpretation and the pixel data [7]. The photometric interpretation states the pixel encoding of the color channels, which for this project is monochromatic.

2.1.2 The Hierarchical Data Format

As the name suggest, the Hierarchical Data Format is a file format used for storing large amounts of data hierarchically internally in a file. It is best compared to a Unix-like file system that resides inside of each HDF-file [8].

Following list explains the higher-level components of the HDF-format that needs to be known for this project [9]:

- Root: The start of the internal data model. Indexed by the path '/'.
- Path/link/key: A string of identifiers that leads to an object. (Example: '/data/image4').
- Group: A collection of objects (can include other groups). Similar to a directory in a standard file system. (Example: '/data').
- Dataset: A multidimensional array of data elements used as the main storage object.

The main benefit of the HDF-format for this project is how it allows us to store all the preprocessed data in a structured way in a singular file, where we can choose to only index the needed data objects. This allows us to avoid keeping all the processed data in the computer's memory, as well as decreasing the I/O times since we only need to read the exact amount of data objects needed from one file [8].

There exists two sets of file formats developed according to this: the older HDF4-format and the newer HDF5-format. For this project, HDF5 will be used and the corresponding files are referred to as h5-files.

2.2 Basic mathematical statistics

The most powerful tool for analysis of large populations of data is mathematical statistics. It allows us to transform parts of the populations into key values that can be used for very simple comparison. Obviously, the field of mathematical statistics is enormous, so only the concepts used for this project will be considered.

2.2.1 Mean

The average value in a population of numerical values. Found by taking the sum of all numbers in the population, and dividing it by the total amount of numbers in the population. Following formula is used for calculating the mean over a population of numbers [10].

$$x = \{x_0, x_1, ..., x_{n-1}, x_n\}$$

$$\bar{x} = \frac{\sum_{i=0}^{n} x_i}{n}$$

2.2.2 Standard deviation

A measure used to determine how much on average an element in a population of numbers deviates from the mean value. Following formula is used for calculating the standard

deviation over a population of numbers [10].

$$x = \{x_0, x_1, ..., x_{n-1}, x_n\}$$

$$\sigma_x = \pm\sqrt{\frac{\sum_{i=0}^{n}(x_i - \bar{x})^2}{n}}$$

Figure 2.1 shows an example of how the mean value and the standard deviations relates to a population of numbers.

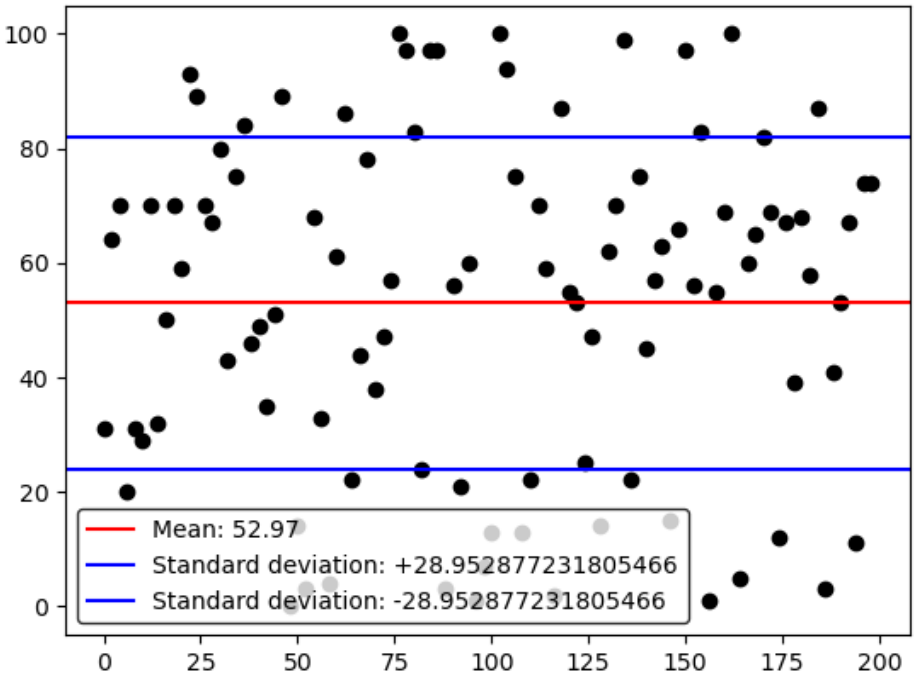


Figure 2.1: Example of a mean value and the standard deviation in a population of numbers.

The standard deviation is useful for understanding the spread of a population and will be handy for the anomaly detection in our x-ray images.

2.3 Machine learning

In computer science the focal point is how to solve advanced tasks by programmatically telling the computer how to do it. Machine learning is the inverse of this, where you instead allow the computers to program themselves in order to solve the task (provided with an initial structure and observational data). This is achieved through the combination of the field of computer science and statistics [11][12].

While machine learning may not solve each and every task, it allows the developers to challenge problems that requires algorithms to advanced for humans to develop [11]. For this project, the most important application of machine learning is computer vision. Computer vision is the field of how to provide computers with a more conscious understanding of images [13].

The following sections will go through each of the important concepts for this project from a practical perspective of how they can be used. Machine learning is a very broad subject, but this project will focus only on neural networks, which is specific type of algorithms used for solving machine learning tasks.

Similar to how there exists many different tools for machine learning, there also exists many different neural networks. However, only deep feedforward networks will be considered here since they are currently the most commonly used neural network [14], and are also what is used in this project.

2.3.1 Artificial neural networks

At its core, a neural network is just one large function f that attempts to approximate an answer y given the input x [14]. This can be written as the following formula:

$$y = f(x)$$

In more practical terms, y is the prediction of the neural network, x is whatever is being predicted on, and f is the mathematical equation system that performs the prediction.

Furthermore, by defining the function f as a composite function of other functions, we get a network of functions [14]. Under the assumption that the functions have no feedback loops, then this is a feedforward network since the the input travels sequentially through the functions. The alternative to this is a recurrent neural network that has internal feedback loops [14].

$$f(x) = f_k(f_j(f_i(x)))$$

Each of these functions represents a layer in the neural network, and their position in the sequence is detrimental. The first layer is referred to as the input layer, the final layer is referred to as the output layer. These are the only layers that we directly interact with, since they are responsible for taking the input data and returning a prediction. All other layers are referred to as the hidden layers, since only the network itself looks at their outputs and modifies their performance [14].

By having multiple hidden layers we get what is called a deep network [12][14]. This is valuable because the approximation of one function is then enhanced by the following function, which gives the network much more complexity for solving its task [12].

In an artificial neural network, a layer contains a set of artificial neurons. Each neuron is simply a node that can hold a numerical value. The neurons inside of a layer are not connected to each other, but instead (or at least most commonly) are connected to all other neurons in the directly previous and succeeding layers [15]. This is referred to as a

fully-connected neural network. Following figure shows an example of this.

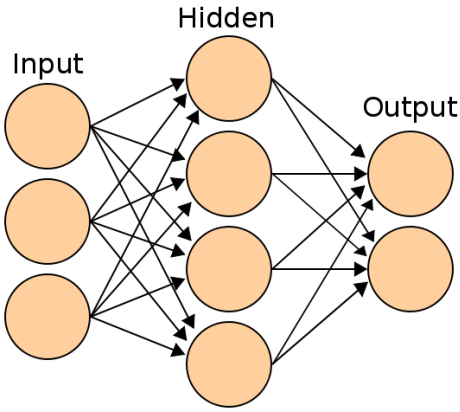


Figure 2.2: A fully-connected artificial neural network with three input neurons, four hidden neurons and two output neurons.

The neurons are arguably the most important part of the network since they are responsible for finding the features and making the predictions. The underlying technique is not that advanced, but it can be tricky to grasp. Remember that we only consider feedforward neural networks here, and that it thus is directed and acyclic. Now, consider the following.

Given the network $y = f_k(f_j(f_i(x)))$, where $y_i = f_i(x)$, $y_j = f_j(y_i)$ and $y = f_k(y_j)$. Let's study an arbitrary neuron a in the layer f_j.

The input to the neuron will be the output of all the other neurons in the layer i, since that is the previous layer and those neurons are all connected to our studied neuron. Let's define those as, $y_i = \{y_{i,0}, y_{i,1}, ..., y_{i,n-1}, y_{i,n}\}$. Each input will also have its own weight connected to our neuron. Let's define those as, $w_a = \{w_{a,0}, w_{a,1}, ..., w_{a,n-1}, w_{a,n}\}$. For both of these, n is the total amount of neurons in the previous layer [16].

In order to get the output of our studied neuron, we have to calculate the weighted sum of the inputs. With what we just defined above, we can now formulate the following output for our arbitrarily picked neuron a in f_j as:

$$y_{j,a} = \sum_{x=0}^{n} (y_{i,x} w_{a,x})$$

On top of that a bias is usually added to the weighted sum in order to allow the neural network to steer the prediction better [16].

$$y_{j,a} = \sum_{x=0}^{n} (y_{i,x} w_{a,x}) + b_a$$

To finalize the weighted sum, an activation function is also applied. This is a nonlinear function that is defined by the layer, and for our example will be referred to as ϕ_j.

$$y_{j,a} = \phi_j(\sum_{x=0}^{n}(y_{i,x}w_{a,x}) + b_a)$$

The activation function is applied because the weighted sum is limited to only understanding linear functions otherwise. By applying a nonlinear transformation, the model can then also be extended to those types of functions as well [14].

There exists many different activation functions. The one used in this project is the rectified linear unit defined as $g(z) = max\{0, z\}$ [14]. Another popular alternative is the sigmoid function defined as $g(z) = \frac{1}{1+e^{-z}}$ [12]. Both of these simply maps their input value in between an interval, $[0, +\infty)$ for the rectified linear unit and $(0, 1)$ for the sigmoid function [15]. All of them have their benefits and drawbacks to be considered.

This operation is then done neuron for neuron, layer after layer in a sequential order, where the output of the previous neurons is the input to the succeeding. The only exception to this is the input data provided to the input layer.

Finally, what is needed to be done in order to achieve an acceptable prediction is to modify the weights and the biases so that the calculation provides the correct output for the given input. This is were the power of the neural networks reside, since through sophisticated algorithms the network programs itself into an acceptable state by observing examples. This will be discussed further in section 2.3.3.

2.3.2 Using a neural network

The first thing that has to be decided when working with neural networks is to determine what task it should solve, often referred to as *feature engineering*. This practically means deciding what to predict and what will be necessary for making such a prediction, or more specifically, what input values can give the desired output values. This is done by defining how many neurons the input layer should have and what their values should represent, as well as how many output neurons the output layer should have and what their values should represent [14][15].

Let's study the network $y = f_k(f_j(f_i(x)))$, and let's say that it should predict the price for a specific product given the supply and the demand.

The input layer f_i will be responsible for taking the input values for the supply and the demand, meaning that it will need two neurons to hold these values. The first neuron will always be provided with the supply and the second neuron will always be provided with the demand.

The output layer f_k will be responsible for providing the output value, which is the predicted price of this specific product, thereby needing only a single neuron.

The hidden layers simply need an arbitrary number of neurons in order to provide more complexity to the network. There is no concrete rule for how many hidden layers there should be or how many neurons they should have, but a simple rule of thumb could be that it should not be much more or much less than the input layer or output layer. Too many neurons in the hidden layers can cause problems, just as well as having too few [14].

In order to predict a new price, a new set of input values are provided. If there is a desire to remodel the network (e.g. changing amount of neurons in a layer), then the network have to be retrained, which will be discussed further in the next section.

2.3.3 Training a neural network

Arguably the most important part of an artificial neural network is that it needs to be trained, otherwise it can not make accurate predictions.

Previously it was discussed that the neural networks make their predictions by taking the input values given to the input layer, where a special weighted sum is calculated neuron for neuron, layer for layer, until it reaches the final layer. In order to get a better prediction for a given input, the only thing needed to be done is to modify the weights and the biases. This is done empirically by the network by iteratively testing it over and over again, and let it modify itself [15].

Training is simply done by providing observable examples for the network to study. Given the network $y = f(x)$, the desire is simply to provide a known x as input and a correspondingly known y as output, (x_{known}, y_{known}), such that:

$$f(x_{known}) = y_{prediction} \approx y_{known}$$

The more $y_{prediction}$ differs from y_{known}, the greater was loss of the prediction. This loss is important because it dictates how much the weights and biases in the network needs to be changed [12].

To calculate this difference a loss function is used [12]. Similar to the activation functions mentioned earlier in section 2.3.1, all of them have their reasons to be and not to be used. A common one is the mean squared error (which is used in this project) with the following formula:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y_{known,i} - y_{prediction,i})^2$$

With i representing one of the n output neurons.

In order to change the weights and biases properly according to the loss, an optimization technique is used. One of the most common is called backpropagation. Practically speaking it studies the output of the loss function, and for each layer identifies which weights and biases that needs to be changed. The reason why it is called the backpropagation is because it moves backwards in the layers [12].

At the end of it, the most important factor is that the observational data needs to be able to cover most of the future cases of input data, so that the network can generalize its predictions accordingly. This requires large amounts of data, but also that the network gets to study it over many iterations.

2.3.4 Hyperparameters

In previous sections certain parameters such as the weights and biases have been discussed. These are important, but its mostly up to the neural network to modify these during training. The hyperparameters however are the parameters that instead are decided during the development of the network by the developer.

These include but are not limited to: the number of hidden layers, the number of neurons in each layer, how many examples to study at a given time during training (batch size) and over how many iterations the data should be studied during training (epochs).

2.3.5 Overfitting and underfitting

The two biggest consequences of a badly built or trained network is overfitting and underfitting [12].

- Overfitting: The neural network has adapted to precisely to the training data and can therefore not generalize to new data.
- Underfitting: The neural network was unable to learn the underlying structure of the data and can therefore not generalize to new data.

2.3.6 Convolutional neural networks

Convolutional neural networks functions similarly to the artificial neural networks as described in section 2.3.1. The major difference is that convolutional kernels is used instead of artificial neurons.

A convolution is a special mathematical linear operation. The operation is not the same as the matrix multiplication which often is used in ordinary artificial neural networks [14], however it slightly reminds of them. Simply what it does is that it maps a smaller matrix over each viable area in the input matrix and takes the sum of the multiplications of the correspondingly mapped elements [13]. This allows the smaller matrix to modify each element in the input matrix according to its neighbouring elements. The smaller matrix is called the convolutional kernel or filter, the input matrix is called the input, and the output matrix is often called the feature map [13][14]. Following is an example of a 3x3 convolutional kernel applied to one area in the 4x4 input matrix, with the complete 2x2 feature map.

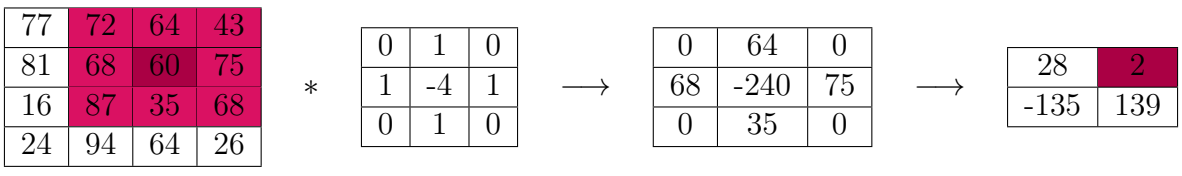


Figure 2.3: Example of the convolutional operation.

Note that the feature maps are also themselves matrices, so the same operation could be performed again. This assumes though that the feature maps produced are not smaller than the convolutional kernels. There are however ways to pad the input so that the feature map remains at the same size after the operation [13].

Using convolutional kernels has massive benefits since the operation can be used in order to detect things such as edges [13][17]. Furthermore, it requires less weights for large inputs, provides parameter sharing (single weight per kernel) and reduces risk of overfitting. Also, since the input does not need to match a specific amount of neurons, it can handle inputs of all sizes (which is handy for images) [14].

Another massive benefit is that the convolutional kernels generally are much smaller than the inputs, which allows them to detect very small details even in larger inputs [14].

Furthermore, in order to provide a more precise approximation and reduce the impact of noise in the training data, a pooling layer is used. This pooling layer decreases the size of the feature maps [13]. A popular pooling technique is max pooling, which simply takes the highest value in a kernel.

In summary, convolutional neural networks make their predictions by creating many feature maps of the input, pooling the feature maps, creating feature maps of the feature maps, and so on until the desired output is produced. Each of these actions will be made into their own layers.

Convolutional neural networks are trained using similar algorithms as described in section 2.3.1.

2.3.7 Difference between supervised and unsupervised

Our previous examples mentioned in section 2.3.1 and 2.3.6 works with a technique called supervised learning. Supervised learning is when the machine learning model gets to train on predetermined inputs and outputs (often called labels). In other words, during training, the model gets provided with an input x_{known} that should produce the output y_{known}. The resulting output of the prediction on x_{known} will be compared to y_{known} [15].

Unsupervised learning is when the machine learning model gets to train on provided inputs without corresponding outputs. This means that system itself must decide what features are of importance and not [15].

This difference is important because this project will work with a specific neural network that learns in an unsupervised manner.

2.3.8 Autoencoders

The autoencoder is a neural network that learns how to reconstruct a given input [14]. This can simply be shown as the following, with the neural network f, the input x and the output y:

$$y = f(x), \text{ where } y \approx x$$

The autoencoder achieves this reconstruction by using an encoder and a decoder [18]. Following the example above; the neural network f will now be the composite of the encoder e and the decoder d:

$$y = d(e(x))$$

The goal of the encoder is to encode the input data to a compressed and hidden representation [18]. This encoding is considered hidden because it is the output of the final layer in the encoder, which is a hidden layer in the composite network and thereby only interacted with by the network itself. The encoding is important because it represents a more abstract and generalized view of the given input. The decoder will take this hidden representation and attempt to reconstruct the input [18].

Autoencoders can be seen as a special case of the feedforward networks that we have seen before [14], and will from here on be treated as such. This means that they benefit from a lot of the techniques used in ordinary artificial neural networks, and can as well use the techniques of the convolutional neural networks. For the case of convolutional networks, in order to decode the encoding of the pooled feature maps, there are special convolutional transpose operations.

An autoencoder trains in an unsupervised fashion. Since its goal is to reconstruct inputs the only thing needed during training is just that. It will observe inputs and compare the reconstructions to them. When the autoencoder trains, it learns which features are important to encode in order to achieve a successful recreation. Importantly, if the autoencoder simply learns to successfully generalize the reconstruction to all future input data, then it is not particularly useful. By training the model in such a way that the reconstruction differs from the input allows for certain approximations to be taken [14].

By using the reconstruction error between the input and the reconstruction, we can determine how much the input differed from the training data [18]. This will be the basis for the anomaly detection in this project.

3. Method

The methodology for this project will be proposed in the order that it was performed. It is primarily an iterative process where each section takes the next step from the previous one. If the work were to be done in any other order (e.g. beginning with the data preparation first), then different conclusions could potentially be drawn due to the different insights gained from that experience. Meaning that this exact order is not a must nor necessarily the optimal.

Initially the idea was to build a convolutional neural network that classifies the images. The problem with that is that it requires the training data to be labeled (i.e. have inputs with predetermined outputs), and while some of it were labeled, most of it was not labeled consistently enough. Since manual labeling would take a lot of time, we began investigating the usage of unsupervised training.

The methodology will proceed in the following order: first experiments with unsupervised training was performed, this was then followed by preprocessing of the real data, then the model got trained, and once the model could perform decent predictions we began experimenting with anomaly detection.

3.1 Material and Software

Following subsections will establish the technological dependencies for this project. None of them are in themselves a necessity for the project to succeed, so any other alternative could be used. However they are recommended.

3.1.1 Programming language

The whole project is developed using Python 3.

3.1.2 Primary libraries and frameworks

- TensorFlow [19]: A machine learning platform that can be used with Python.
- Keras [20]: A high-level machine learning API used together with TensorFlow.
- Pydicom [21]: A Python library for working with DICOM-files.
- h5py [22]: A Python library for working with HDF5-files.
- OpenCV [23]: A Python library used for image processing.
- Matplotlib [24]: A Python library used for plotting and showing images.
- NumPy [25]: A Python library used for scientific computing.
- Pillow [26]: A Python library used for image generation.

3.1.3 Development environment

Most development is done in Visual Studio Code [27] (though any other editor will work equally well). The only exception to the former is that Jupyter Notebook [28] was used when the models predictions were tried for anomaly detection, which was really useful because it allows for saving the state of processed data (i.e. in this case the manual color coding, which will be explained later in section 3.6).

3.2 Implementing the machine learning model

Since the source data lacked consistent labeling, the decision to use supervised learning (such as with a convolutional neural network) became undesirable. Obviously, manual labeling could be done. However that would take an unnecessary large amount of time. Therefore unsupervised learning with a convolutional autoencoder instead became the interesting choice, since they had the potential to be used for anomaly detection in images without the need for labels.

As mentioned in section 2.3.6, the reason to choose convolutional networks instead of ones with artificial neurons is because they are generally better at handling images.

3.2.1 Model and architecture

The autoencoder used in the end of the project is nearly exactly the same as the one used from the start (aside for a few hyperparameters). The developed model is nothing revolutionary since it is a seemingly standard convolutional autoencoder. This is fine, since the main purpose was to apply machine learning to a task.

The reason why it has not been changed that much is because it gave desired results very quickly.

The primary encoder used in this project had three convolutional layers. The convolutional layers used respectively 32, 64 and 128 filters with a 3x3 convolutional kernel that iterated one pixel at the time. Padding was used to maintain the size of the feature maps and the rectified linear unit was used as activation function. Each convolutional layer was followed by a pooling layer that halved the height and the width of the extracted feature maps, by taking out the maximum values in each 2x2 square.

The corresponding decoder did the same but instead reversed it, using transposing convolutional layers and up-sampling layers. The only difference was an additional transposing convolutional layer that extracted a single feature map, i.e. the final reconstruction.

3.2.2 Implementation

The model is built using TensorFlow and Keras in Python. To give a reference, here is some example code for how to create a two-dimensional convolutional autoencoder using the aforementioned:

```
from tensorflow import keras

def build_model():
    model = keras.Sequential()
```

```
model.add(keras.layers.Conv2D('hyperparameters'))
model.add(keras.layers.MaxPool2D('hyperparameters'))
# ... more layers if desired ...
model.add(keras.layers.UpSampling2D('hyperparameters'))
model.add(keras.layers.Conv2DTranspose('hyperparameters'))
return model
```

3.3 Testing the model

With the model implemented in fairly short amount of time. The best way to prove that the model functioned as desired was to simply test its ability to predict and reconstruct provided images. Generally looked for is a low loss from the predictions of similar images as the ones that were trained upon and a higher loss for images that differs more from the training data.

Four major experiments were performed, all of which are documented in section 4.1.1 through 4.1.4.

3.3.1 Generated vertical lines

This test was the first test that was performed due to its sheer simplicity, with the main goal being to see if the model actually learnt something.

The model got to train on 20 autogenerated images of the size 20x20, where each image had a single vertical line going all the way from the upper frame to the lower frame of the image, placed somewhere on the horizontal axis. This was compared to an image that instead had a horizontal line going from the left frame to the right frame, placed somewhere on the vertical axis.

A successful result simply requires a higher reconstruction error with the horizontal lines than the vertical lines.

3.3.2 Handwritten digits

MNIST is a massive database with labeled images of handwritten digits[1].

This was the second test performed because TensorFlow already provides easy and accessible preprocessed data to this dataset[2], so the only thing needed to be done was to feed it to the model.

The data used is 60 000 images of handwritten digits at the size 28x28 pixels. These were trained on over the course of 32 epochs with a batch size of 32. Keep in mind that these images are provided with their corresponding labels (e.g. the handwritten digit two is paired with the label "two"), however since the autoencoder only reconstructs images, these were discarded since they are not meaningful.

The main goal here was to see with precision how much the model learnt when it got to train on a large dataset. In this specific test, it is not compared to a faulty image but

[1]Link to the MNIST database: `http://yann.lecun.com/exdb/mnist/`
[2]Link to TensorFlow's dataset: `https://www.tensorflow.org/datasets/catalog/mnist/`

just the reconstruction of non-faulty ones.

3.3.3 Generated shapes of various sizes

Since the handwritten digits does not relate in any shape to the desired final product of this project, the next step was to replicate the same precision but with circles instead since it is more comparable to the x-ray images. The idea here was to train the model on generated circles that were non-faulty, and then investigate if there was any difference had in the prediction between a faulty and a non-faulty circle. The bigger the difference achieved was, the better.

This test is in itself kind of four different tests, where between each of these tests the layers were played around with to see if there was any benefit to any iteration. For each test, 2000 images were generated, trained over 10 epochs with a batch size of 32. Also to note, the smaller circles are actually painted as squares. This is not an accident, it is just how the library that is used for generating the images does it for really small circles.

The larger circles are faulty, the smaller are non-faulty. So the actual shape is not the defining factor. The size of the anomalies are the same through all tests, just that the frame studied gets larger towards the end.

3.3.4 Generated dirty images

All tests above functions based on one faith, that the images only contain a background and some shape. For the real x-ray images, this will definitely not be the case since the images are naturally irregular. Thereby doing some examples on "dirtier" images would be worthwhile, to see the performance of the model and look for potential solutions.

To be noted: the model in this test used the one with the best performance in the test before, which was a convolutional autoencoder with three convolutional and pooling pair layers for both the encoder and decoder (same as the one described in section 3.2.1). The generated images here are 64x64 in size.

The anomalies were also made larger to make them stand out more from the dirt and also be more representative of the size in the real x-ray scans (since the anomalies there are larger than the ones in the previous test).

3.4 Preparing the data

For a machine learning model to work, the most important part is being able to provide it with correctly formatted data. There needs to be enough different cases available to train on, so that the model can generalize its solution for all future cases.

In this project around 200 x-ray images were used as the source of data. As mentioned before, labeling did exist but not consistently.

3.4.1 Interpreting the data

All the x-ray images were massive. However, large chunks of each image were simply uninteresting since they did not contain any welding. This meant that for the sake of the

models performance, there was a desire to crop out the interesting parts and only train and predict on those.

As mentioned earlier, since the data did not have labels consistently the choice became to use unsupervised learning, meaning that only input data is needed. However, this inconsistency was not only the case for the labels but also for the images themselves.

An x-ray image contained around 1-4 different scans wanted to observed. This means that a single image could contain multiple parts of interest. A guesstimate of the total amount of parts would be around 650 in total, each part having an area of a handful of millions of pixels. The layout of the parts to be observed in the x-ray scans also varied, with some being short and vertical, others being long and vertical, some were horizontal and some were curvy.

This lack of uniformity in the structure of the data meant that any simple automated preprocessing was out of the question.

3.4.2 Sorting the data

The first thing needed to be done was to sort out the raw data between non-faulty and faulty images. As mentioned above, most images contained multiple different parts. This meant that an image could contain both faulty and non-faulty parts. However, for the simplicity of the sorting, if an image contained as much as one faulty part the whole image was considered faulty.

3.4.3 Converting DICOM to HDF5

Since all x-ray images were saved as individual DICOM-files, the first thing done was to write over all the image data to HDF5-files in order to scale the data usage better. This meant that all non-faulty images were written to a singular HDF5-file, where each image became one dataset, and similar for all the faulty images but to another file.

The benefit of this is that all data is stored into only two files, where individual images can be indexed.

Since the raw x-ray images were saved with a fairly high bit depth, all images were converted to 8-bit in order to minimize the size of the data.

3.4.4 Manual cropping

All of the manual cropping were done by using a self-made tool in Python[3]. It simply opened an image with Matplotlib, were the user got the enter which corners the crop should be in between, saved the cropping and marked on the image were the cropping were done. This allowed so that one could easily take out multiple croppings of a single image, and know which ones had already been extracted.

All of the croppings were saved in a new HDF5-file, were each dataset contained a cropping. At the end, approximately 1100 croppings had been extracted.

[3]The reason why there was a need for having a self-made cropping tool was because the software used for viewing the DICOM-images could not crop images. We also wanted to avoid converting images to other file-formats, so we could not use the tools for those either.

Note: Since only the training data needed to be manually cropped at this stage, only the non-faulty images were cropped. When testing with the faulty images later, certain cropping was performed there and then.

3.4.5 Automatically cropping out fragments

Since even the crops were very large and the anomalies to be detected in the images were minuscule in comparison. The next decision taken was to fragment the croppings, which practically meant that each cropping were to be broken down into loads of very small crops of the original cropping.

There were a lot of benefits to doing this:

1. Since each anomaly is minuscule compared to the crops, it would be more challenging to identify them. By fragmenting the croppings, this contrast decreased and the anomalies could be studied more locally.
2. Since loads of fragments could be extracted from a cropping, there were a lot more cases that the model could study, rather than a few larger croppings.
3. If an anomaly were to be found, it would be easy to mark its location in the image since that would just be the position of the fragment.

It was decided that each fragment would be 64x64 pixels, since this was more than enough to fit an anomaly. In order to not split an anomaly between fragments, each fragment would need to overlap with their previous and thus iterate with at most 50 % of the width and the height of the fragment. There was also conditions made so that the borders of the crops were guaranteed to be fragmented too.

Ultimately, this generated approximately two million fragments. Each fragment was put as a dataset into a group corresponding to which original cropping it was, all this in a new HDF5-file. The fragments needed to be seperated into different groups, because otherwise the HDF5-file would bottleneck if so many keys were to be put on the same level.

3.5 Training the model

Since two million fragments could not be loaded into memory at once, a data sequence was built using TensorFlow. Instead of reading all data at once, only all the keys were read from the HDF5-file. These were shuffled at the start of each epoch. For each batch only the required amount of fragments are read simply by only indexing the needed keys and reading those.

The model was trained on a Linux server via SSH (Secure Shell) by running the following command:

```
nohup python3 train.py > /dev/null &
```

Closing the SSH forced the training to stop since the terminal that took the output were removed (even if it were run in the background), but by redirecting the output to anywhere else deceived it into keeping the training script running. Having the need to have the SSH open for a long time is risky since it can lose connection to the network and then halt the training. But this was a decent workaround.

The following training sessions got to train on the same set of data (i.e. all of the non-faulty fragments) using the Adam optimizer and the mean squared error loss function.

3.5.1 First training

The model got to train on the fragments for five epochs with a batch size of 512. This took around 25 hours to complete.

Most results will be documented by using this model.

3.5.2 Second training

The model got to train on the fragments for 20 epochs with a batch size of 256. This took around 5 days to complete.

The model here is exactly the same as in the first training session.

3.6 Using the model for anomaly detection

The final step of this project was to compare methodologies for identifying anomalies. This was be done by comparing the reconstruction errors between faulty and non-faulty fragments using various statistical techniques.

Following subsections will explain each alternative. The goal was that the techniques should be able to differentiate between the reconstruction errors so that a threshold could be established with which fragments without anomalies could be separated from fragments with.

Each of these alternatives was analysed by the results of two different croppings in a single x-ray image. Each of these croppings had their original fragments, reconstructions and reconstruction errors (differences) compared. The reason why two different croppings were compared was because certain techniques differed in quality depending on where the model studied the image. Similarly, a comparison between the techniques used on the original fragments, the reconstructions and the differences were necessary to prove the usefulness of the model's predictions.

The results of these alternatives are documented in section 4.2.1.

Following notations will be used when explaining each alternative:

$$p^f = \{p^f_{0,0}, p^f_{1,0}, ..., p^f_{n-1,m}, p^f_{n,m}\}$$
$$p^r = \{p^r_{0,0}, p^r_{1,0}, ..., p^r_{n-1,m}, p^r_{n,m}\}$$
$$p^d = \{p^f_{0,0} - p^r_{0,0}, p^f_{1,0} - p^r_{1,0}, ..., p^f_{n-1,m} - p^r_{n-1,m}, p^f_{n,m} - p^r_{n,m}\}$$

Where p^f is the flat set of pixels in one of the original fragments studied, p^r is the flat set of pixels in the corresponding reconstruction, and p^d is the elementwise difference between these. For this specific project, n and m are both 64, since those are the dimensions of the fragments.

3.6.1 The sum of the reconstruction errors

Simply calculated the total value in the fragments, the reconstructions and the differences.

Following formulas where used:

$$\text{For a fragment: } SUM = \sum_{i=0}^{n} \sum_{j=0}^{m} p_{i,j}^{f}$$

$$\text{For a reconstruction: } SUM = \sum_{i=0}^{n} \sum_{j=0}^{m} p_{i,j}^{r}$$

For the difference between a fragment and a corresponding reconstruction:

$$SUM = \sum_{i=0}^{n} \sum_{j=0}^{m} |p_{i,j}^{f} - p_{i,j}^{r}|$$

3.6.2 The extreme values of the reconstruction errors

Took out the highest and lowest value in the fragments, the reconstructions and the differences.

Following formulas where used:

$$\text{For a fragment: } MAX = max(p^{f}),\ MIN = min(p^{f})$$

$$\text{For a reconstruction: } MAX = max(p^{r}),\ MIN = min(p^{r})$$

For the difference between a fragment and a corresponding reconstruction:

$$MAX = max(p^{d}),\ MIN = min(p^{d})$$

3.6.3 The standard deviation of the reconstruction errors

Calculated the spread of the of the data in the fragments, the reconstructions and the differences.

Following formulas where used:

$$\text{For a fragment: } \sigma_{p^f} = +\sqrt{\frac{\sum_{i=0}^{n} \sum_{j=0}^{m} (p_{i,j}^{f} - \bar{p^f})^2}{n \times m}}$$

$$\text{For a reconstruction: } \sigma_{p^r} = +\sqrt{\frac{\sum_{i=0}^{n} \sum_{j=0}^{m} (p_{i,j}^{r} - \bar{p^r})^2}{n \times m}}$$

For the difference between a fragment and a corresponding reconstruction:

$$\sigma_{p^d} = +\sqrt{\frac{\sum_{i=0}^{n} \sum_{j=0}^{m} (p_{i,j}^{d} - \bar{p^d})^2}{n \times m}}$$

3.6.4 The mean squared error of the reconstruction errors

Since the mean squared error is defined on the difference between a real value and the predicted value, this one can only be studied for the differences. The idea behind this alternative is similar to taking the sum or the mean, just that it exaggerates the differences more (since it is squared).

Following formula where used:

$$\text{For a difference: } MSE = \frac{1}{n \times m} \sum_{i=0}^{n} \sum_{j=0}^{m} (p_{i,j}^{d})^2$$

4. Results

This section will cover the results that was achieved in chronological order.

Developing the machine learning model was not very challenging, so the testing could begin fairly quickly. Across the tests only minor modifications was done, so the model remained virtually the same across all the tests.

All plotting is done with Matplotlib.

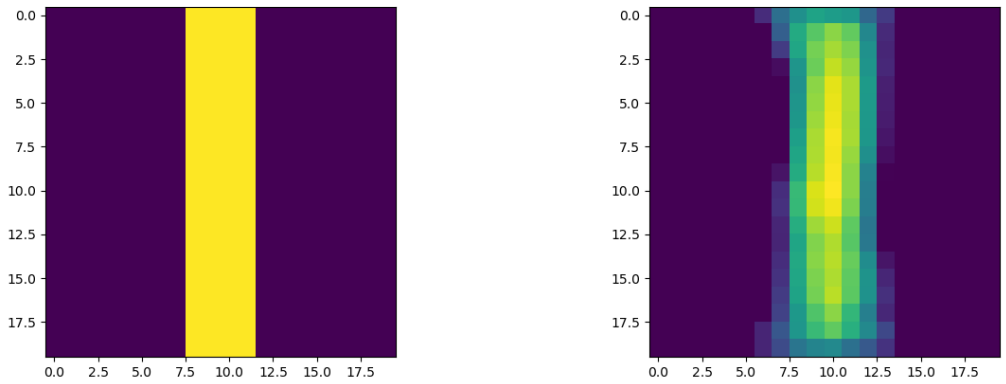
The uttermost desired result is that an anomaly should be identified anywhere in the image, regardless of position and size.

4.1 Testing the model

Following subsections deal with the first tests performed on the model, as explained in section 3.3.

Note: All images are monochromatic and plotted with the viridis color scheme!

4.1.1 Generated vertical lines

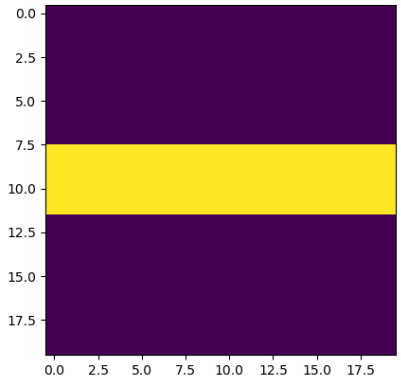


(a) Input image

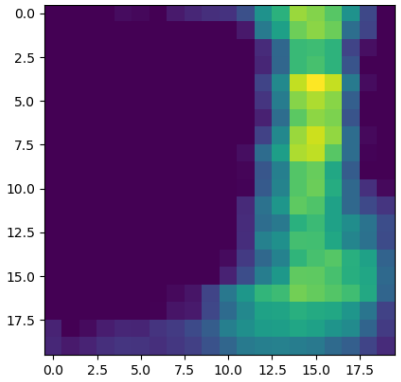
(b) Predicted image

Figure 4.1: Example with a vertical line.

As can be seen in the figure above, if the autoencoder is provided with an image of a vertical line, it can kind of identify it. However, if it is provided with an image of a horizontal line, as can be seen in the figure below, it does not really understand what it is looking at and assumes it to be some kind of vertical line. This simple test proved that the model learnt from the provided data, and can make seemingly decent predictions.



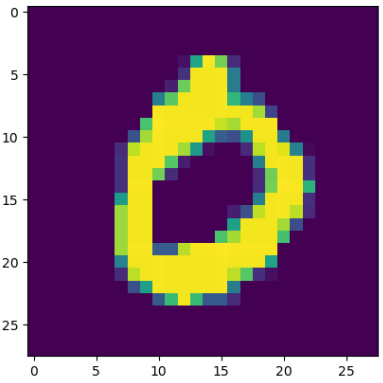
(a) Input image



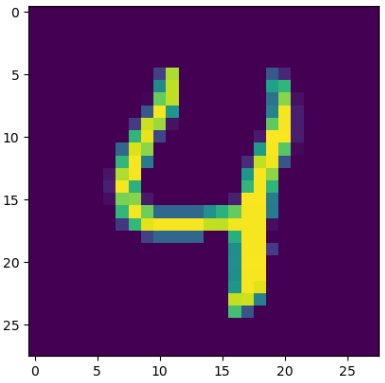
(b) Predicted image

Figure 4.2: Example with a horizontal line.

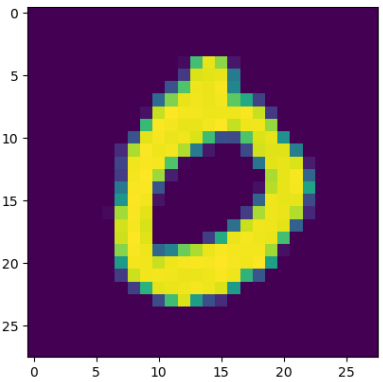
4.1.2 Handwritten digits



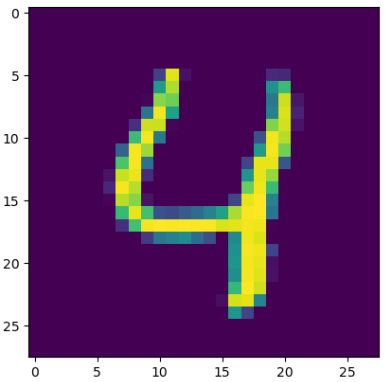
(a) Input digit "zero"



(b) Input digit "four"



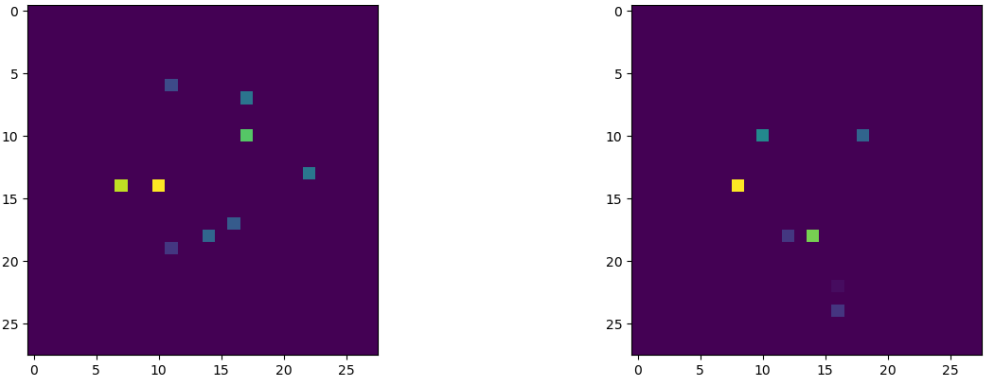
(c) Predicted digit



(d) Predicted digit

Figure 4.3: Example with MNIST handwritten digits.

Comparatively with the example in figure 4.1, these predictions are far better. However, it is important to note that there is a loss in both of these predictions, although it is minuscule. The best way to identify this loss is by doing a simple subtraction from the predicted image on the original image. This is important to remember because it becomes meaningful when attempting to predict whether or not an x-ray image contains an anomaly.



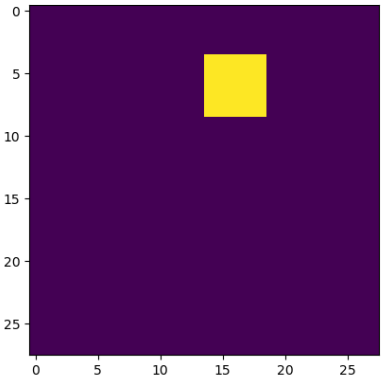
(a) The loss from input digit "zero"

(b) The loss from input digit "four"

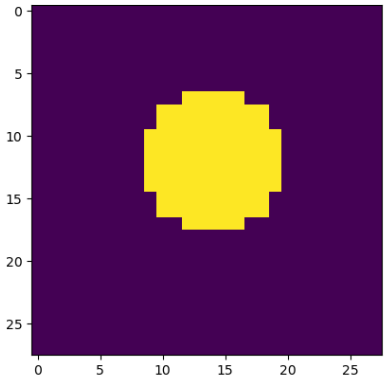
Figure 4.4: The difference between the original image and the predicted image for both digits. (Note: the image shown is the inverted difference).

4.1.3 Generated shapes of various sizes

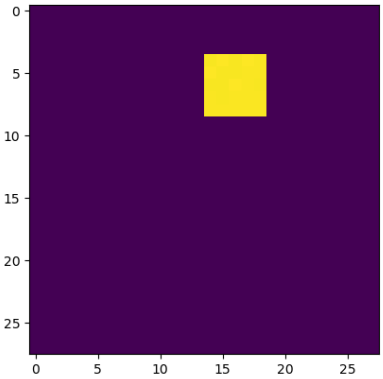
For the first test, as can be seen in figure 4.5 below, the encoder and decoder only had a single convolutional layer and a single pooling layer to its disposal. Despite that, the results are quite good since it is really certain on how the non-faulty circle should look but less so for the faulty counterpart.



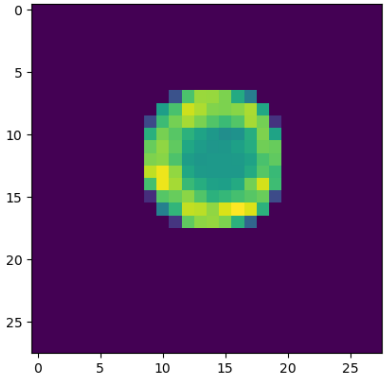
(a) Non-faulty input image



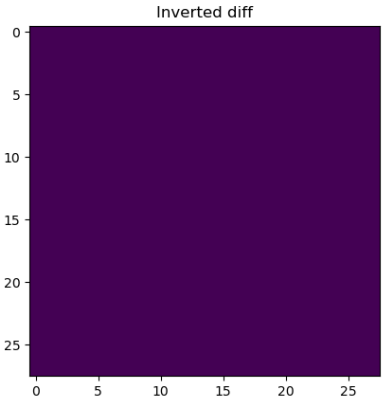
(b) Faulty input image



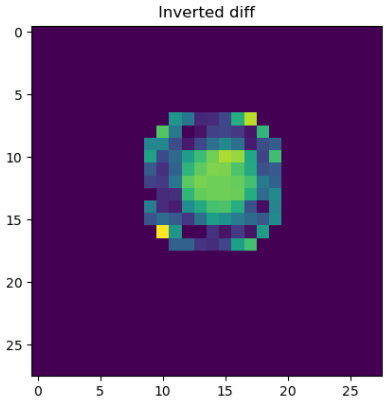
(c) Non-faulty prediction



(d) Faulty prediction



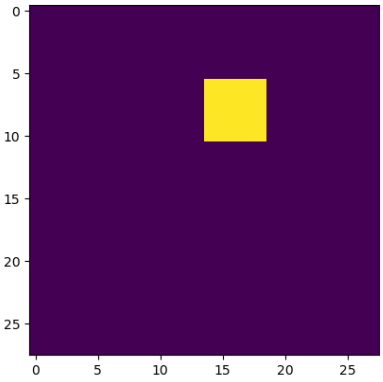
(e) Reconstruction error



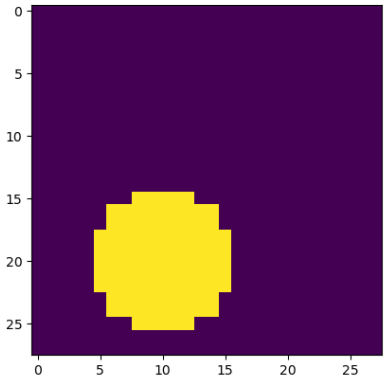
(f) Reconstruction error

Figure 4.5: Example with a single convolutional layer.

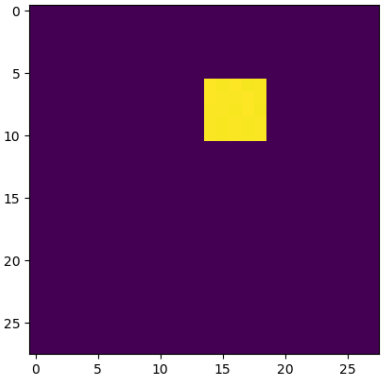
Following this, similar was attempted but with two convolutional layers and two pooling layers.



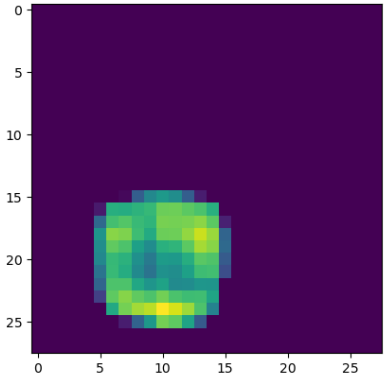
(a) Non-faulty input image



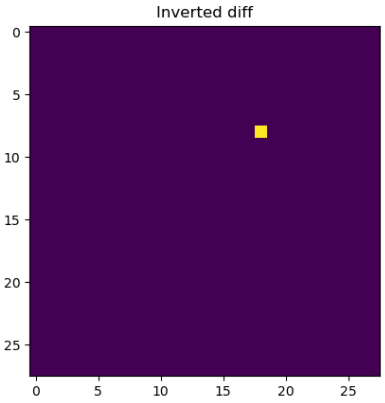
(b) Faulty input image



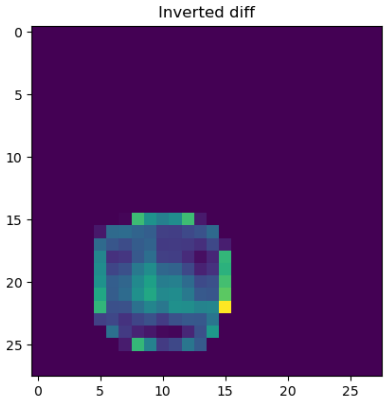
(c) Non-faulty prediction



(d) Faulty prediction



(e) Reconstruction error

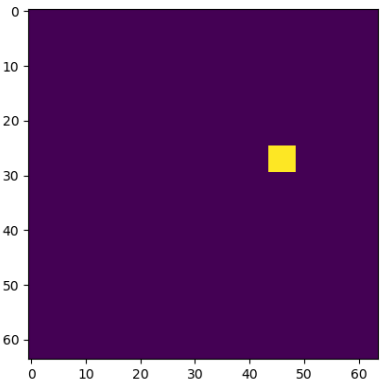


(f) Reconstruction error

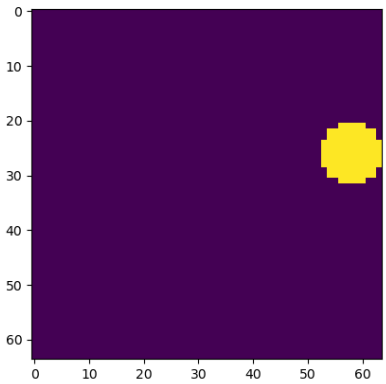
Figure 4.6: Example with a double convolutional layer.

It may be difficult to spot any difference in the performance, but notice how with the faulty circle it tries to reproduce the square shape of the non-faulty circle. This is important, because it means that it learnt a slightly bit more about the features of the non-faulty images than before due to the added layers.

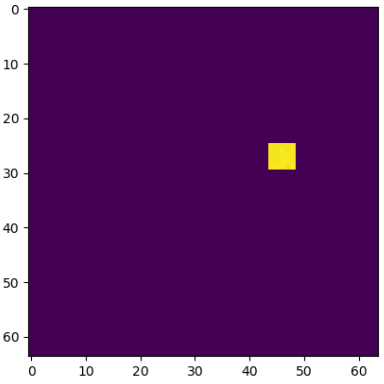
Following this, the similar was attempted but with three convolutional layers and three pooling layers for the encoder and decoder. Note how the size in the images increases from 28x28 pixels to 64x64 pixels. The reason for this is the added pooling layer. 28 is not divisible by eight, and 32 gave no predictions because the image got downsampled too much.



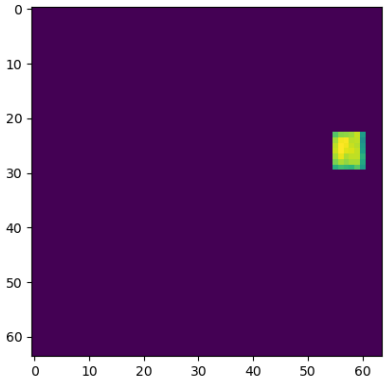
(a) Non-faulty input image



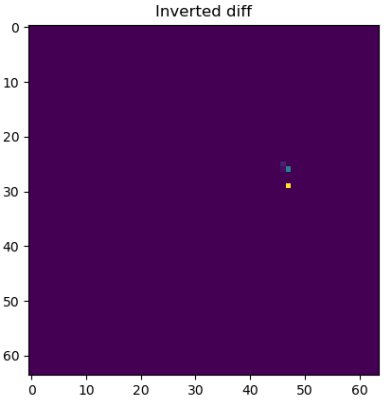
(b) Faulty input image



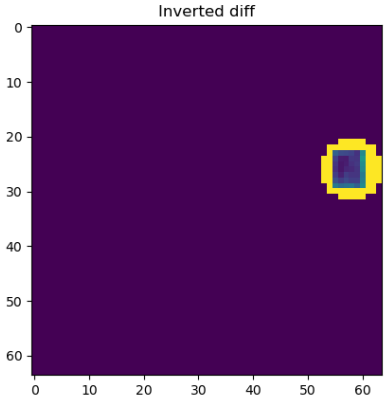
(c) Non-faulty prediction



(d) Faulty prediction



(e) Reconstruction error

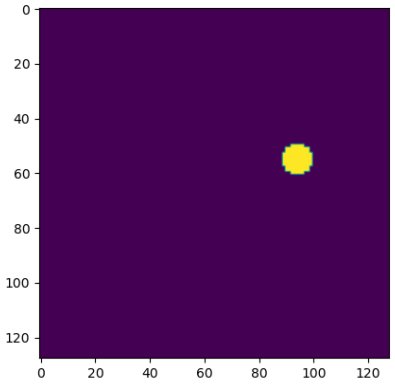


(f) Reconstruction error

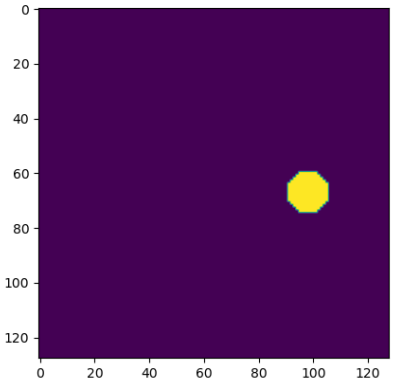
Figure 4.7: Example with a triple convolutional layer.

The figure above shows the best result thus far, since it is certain about how a non-faulty circle should look like and even manages to turn the faulty circle into a non-faulty one. Ideally, this is exactly the result desired from the final product. Thereby, this iteration of the model with the size 64x64 of the images is what will be used.

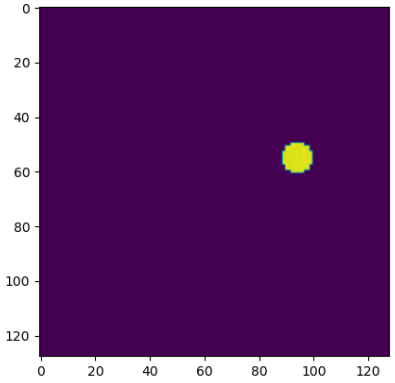
Lastly for this test, the same model is used but the size of the images are further increased by a factor of four, giving the images a size of 128x128. The only reason this is included is to prove that larger images complicates it for the model.



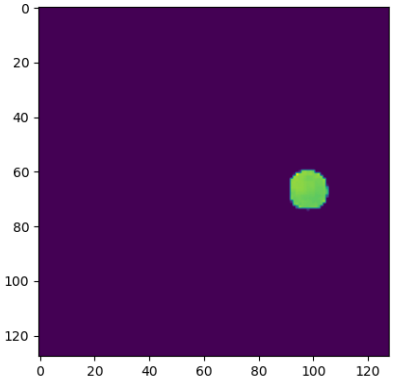
(a) Non-faulty input image



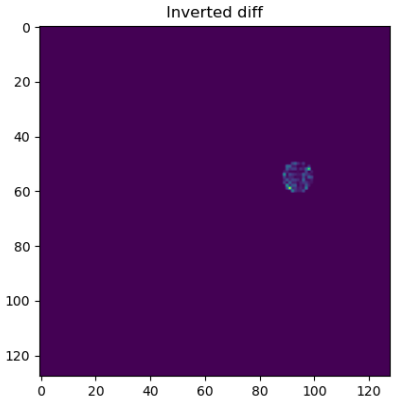
(b) Faulty input image



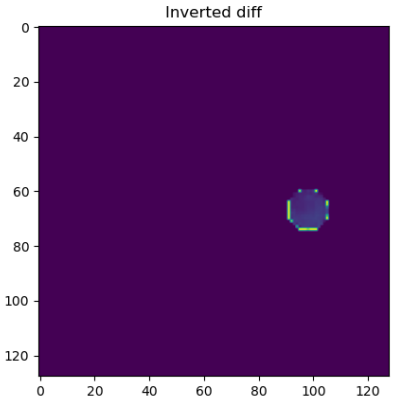
(c) Non-faulty prediction



(d) Faulty prediction



(e) Reconstruction error



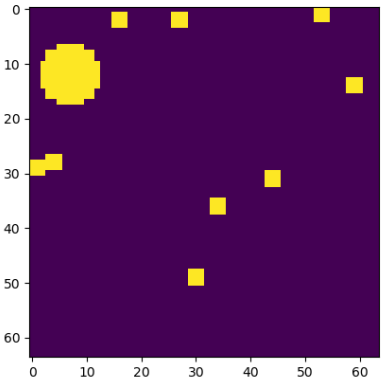
(f) Reconstruction error

Figure 4.8: Example with a triple convolutional layer and larger images.

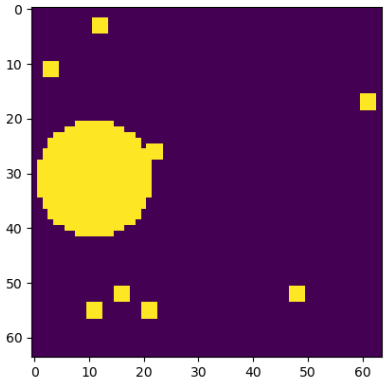
As can be seen in the figure above, the model handles both cases fairly well. Even though it still could function for anomaly detection, it may still complicate the matter.

4.1.4 Generated dirty images

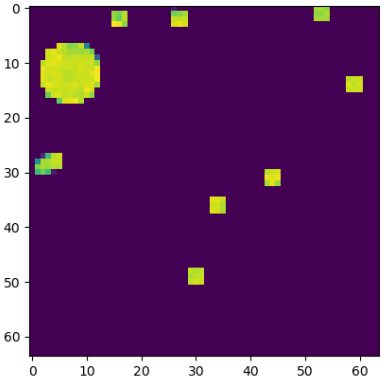
Similar to the second last example given in section 4, these images will be 64x64 pixels and the encoder and decoder will have three convolutional layers.



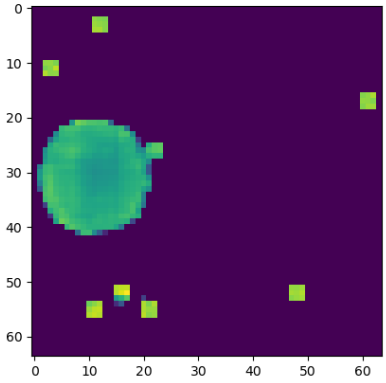
(a) Non-faulty input image



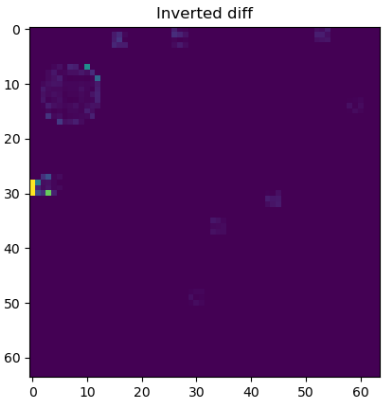
(b) Faulty input image



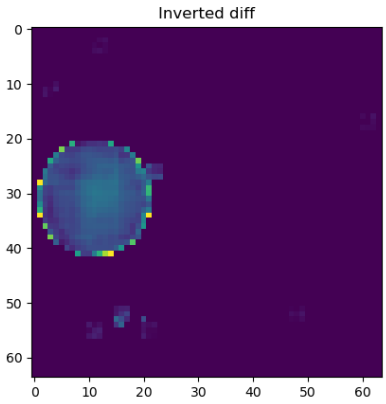
(c) Non-faulty prediction



(d) Faulty prediction



(e) Reconstruction error



(f) Reconstruction error

Figure 4.9: Example with dirty images.

As can be seen in the image above, the same fantastic performance seen in figure 4.7 is not achieved when the images are dirty. This is a bit problematic, since it means that its harder to distinguish between a faulty and a non-faulty anomaly.

However, by simply using the autoencoder for its ability to generalize an image, some simple anomaly detection can still be achieved. Essentially all that needs to be done is to take the difference of the original image and the predicted image, and sum together the differencing pixels. The reason for that is obvious, the more faulty the image is, the more difference will it have from the average image (which is what the model is trained on).

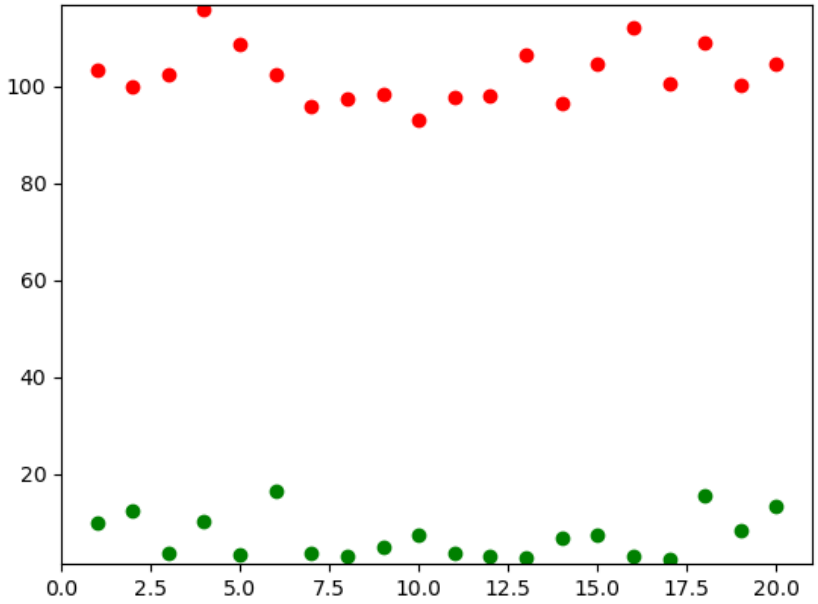


Figure 4.10: The sum of the pixels in the diffed images.
Red: faulty, green: non-faulty.

By putting a threshold between the green values and the red values, some simple anomaly detection can be achieved.

While it can be interesting to look at images, the actual prediction and input does not really matter. Just the reconstruction error between the two of them. Thereby, following tests will primarily be shown through plots.

4.2 Using the model for anomaly detection

The model has now been trained and is ready for some real analysis. Following subsections will go through each of the different alternatives used for anomaly detection.

Each section will show a handful of plots comparing the outcome of the studied alternatives. The comparison will be between two different croppings where it is tested for the fragments in both the croppings, the corresponding recreations and the corresponding differences.

The color coding in the scatter plots are manually labeled, thereby the size of the crops could not be too big. The color coding is also very simple, each scatter point belongs to a fragment with the following definition:

Green	No anomaly in this fragment
Yellow	Some anomaly that could be problematic
Red	Some anomaly that most likely is problematic

Furthermore, the general design of the scatter plots is to put the value analysed on the y-axis and which index of fragment that is studied on the x-axis. In other words, each scatter maps to an index on the x-axis and a comparable measurement on the y-axis.

What will be seen in the following subsections is that there were many alright candidates for a solution to the problem. None of them were perfect though.

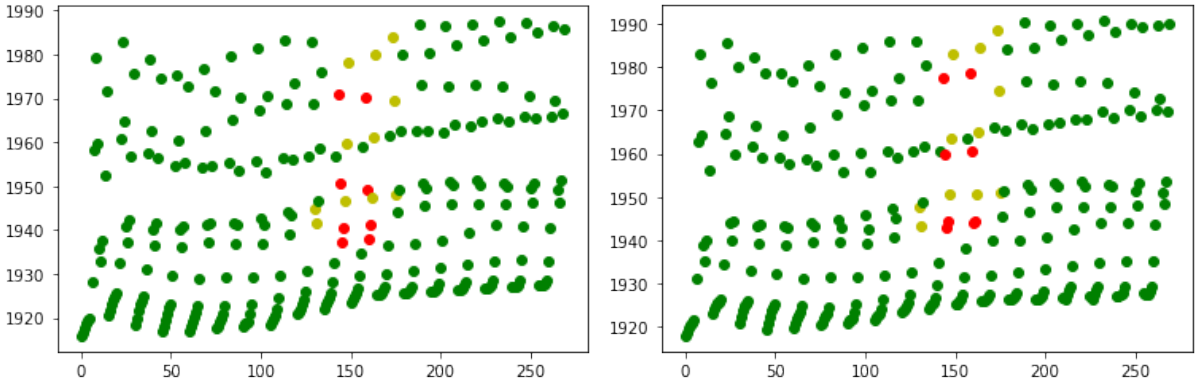
To understand the plots better: A successful alternative is one that can differentiate between the fragments with anomalies (red/yellow) and the ones without (green) using the values on the y-axis. Preferably, the recreations and/or differences should show something that the fragments are incapable of telling themselves, since that would mean that the model could do something that the raw data could not.

4.2.1 Results after the first training

Following results are from the usage of the first trained model. These will make up the bulk of the analysis.

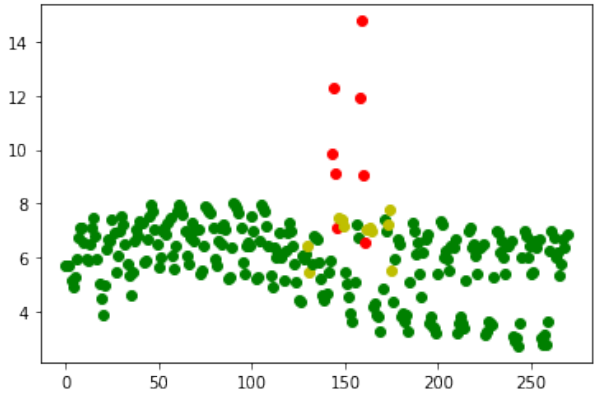
The sum of the reconstruction errors

The first alternative tested was simply taking the sum of all the pixels.



(a) Sum of the fragments.

(b) Sum of the recreations.

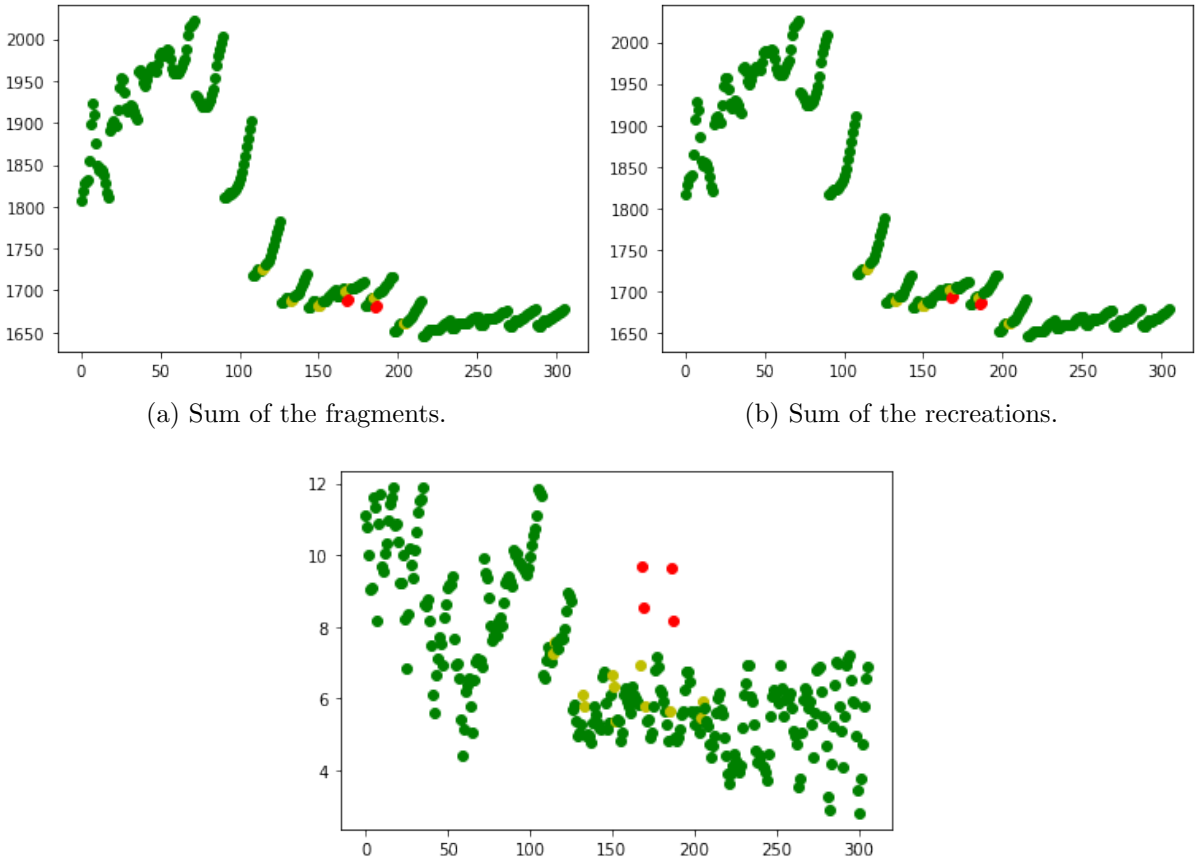


(c) Sum of the differences.

Figure 4.11: The plots of the sums for the first cropping.

By looking at the figures in 4.11, the first takeaway was that simply using the sum of the fragments (a) or the sum of the recreations (b) for any detection is impossible, since one could barely distinguish between the ones with and without anomalies. The sum of the differences (c) however showed a much brighter possibility since there is a decent distinction between some of the likely anomalies and the rest. The problem here however is that most of the uncertain anomalies and even some of the likely anomalies are indistinguishable from the ones without. Possible solutions for this will be discussed later.

The problem with this alternative though is that it fared much worse when studying the second cropping, shown in the figures in 4.12.



(a) Sum of the fragments.

(b) Sum of the recreations.

(c) Sum of the differences.

Figure 4.12: The plots of the sums for the second cropping.

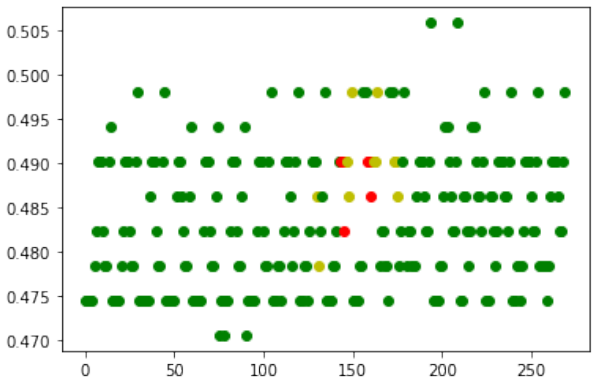
The important conclusion here is that all anomalies are completely indistinguishable from the fragments without. Meaning that using the sum as an alternative is not translatable to all croppings with the current iteration of this model.

The extreme values of the reconstruction errors

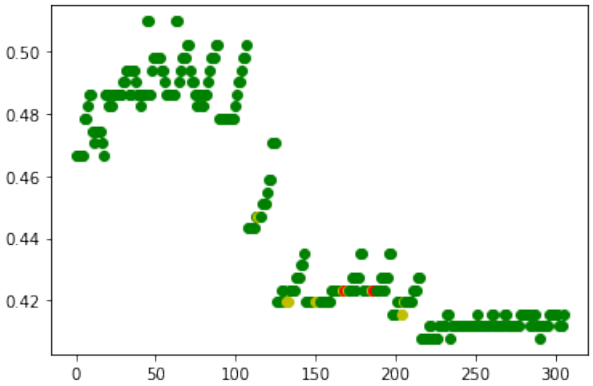
Another alternative is to try and identify anomalies using the extreme values of the fragments. Using extreme values is itself a bit illogical since it completely disregards measurements such as the surface size of the anomaly, which were a big factor in the task of identifying these anomalies. The thing however is that if the model has learnt the data properly, then smaller areas that are non-faulty would be reduced or even removed when using the difference between the fragments and their recreations, meaning that the extreme value would be removed.

The maximums

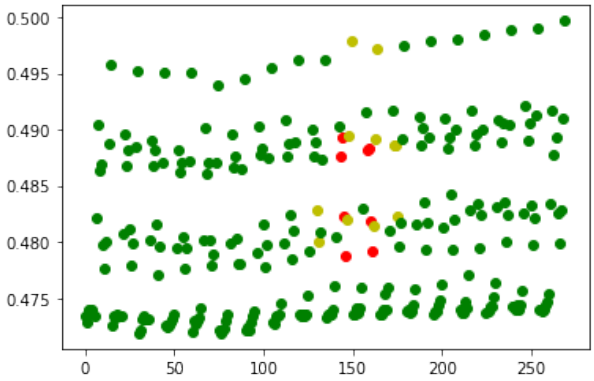
Using the maximums is completely meaningless alternative since the anomalies studied has a smaller values than their non-anomalous counterparts. So it is mainly included for context. The figures in 4.13 tells the same story where the anomalies are all indistinguishable.



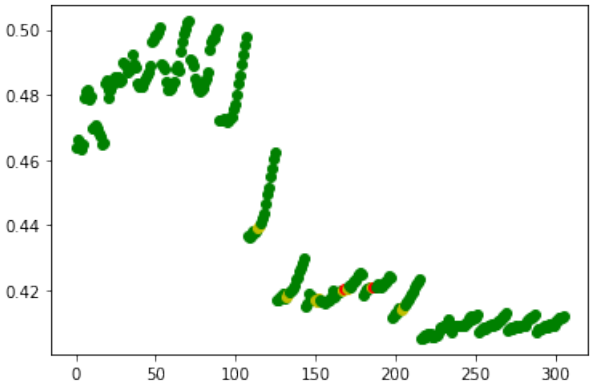
(a) The fragments for the first cropping.



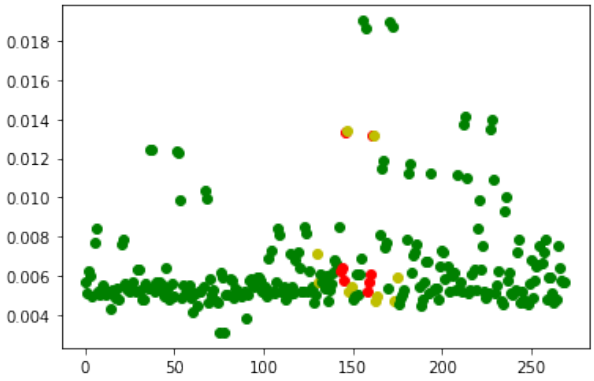
(b) The fragments for the second cropping.



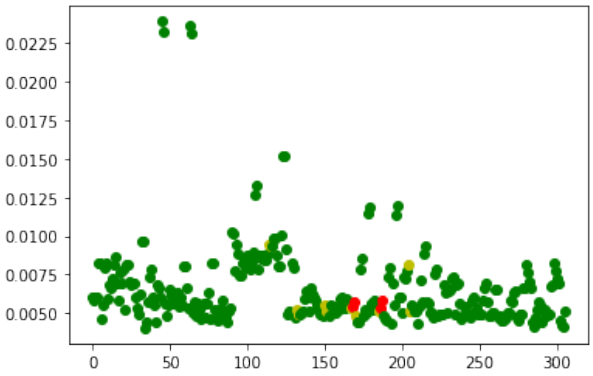
(c) The recreations for the first cropping.



(d) The recreations for the second cropping.



(e) The differences for the first cropping.

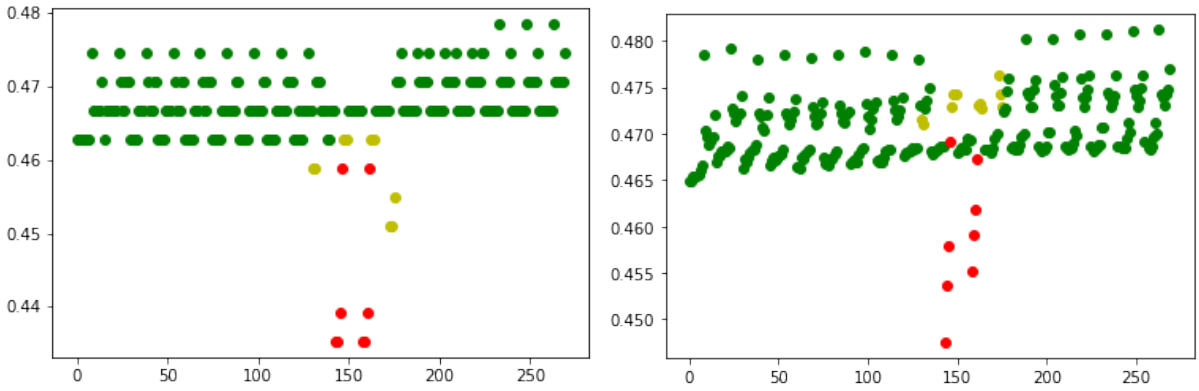


(f) The differences for the second cropping.

Figure 4.13: The plots of the maximums for the first and second cropping.

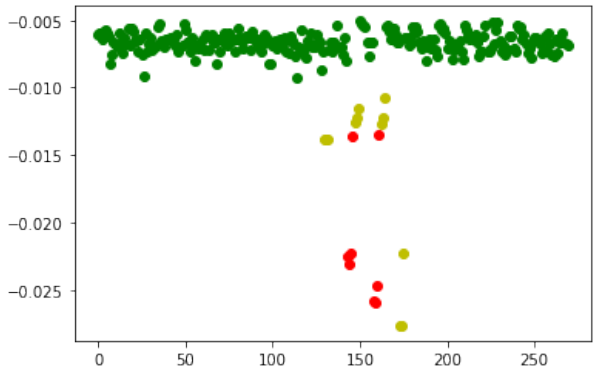
The minimums

The minimal values on the other hand turned out to be much more successful.



(a) Minimums of the fragments.

(b) Minimums of the recreations.



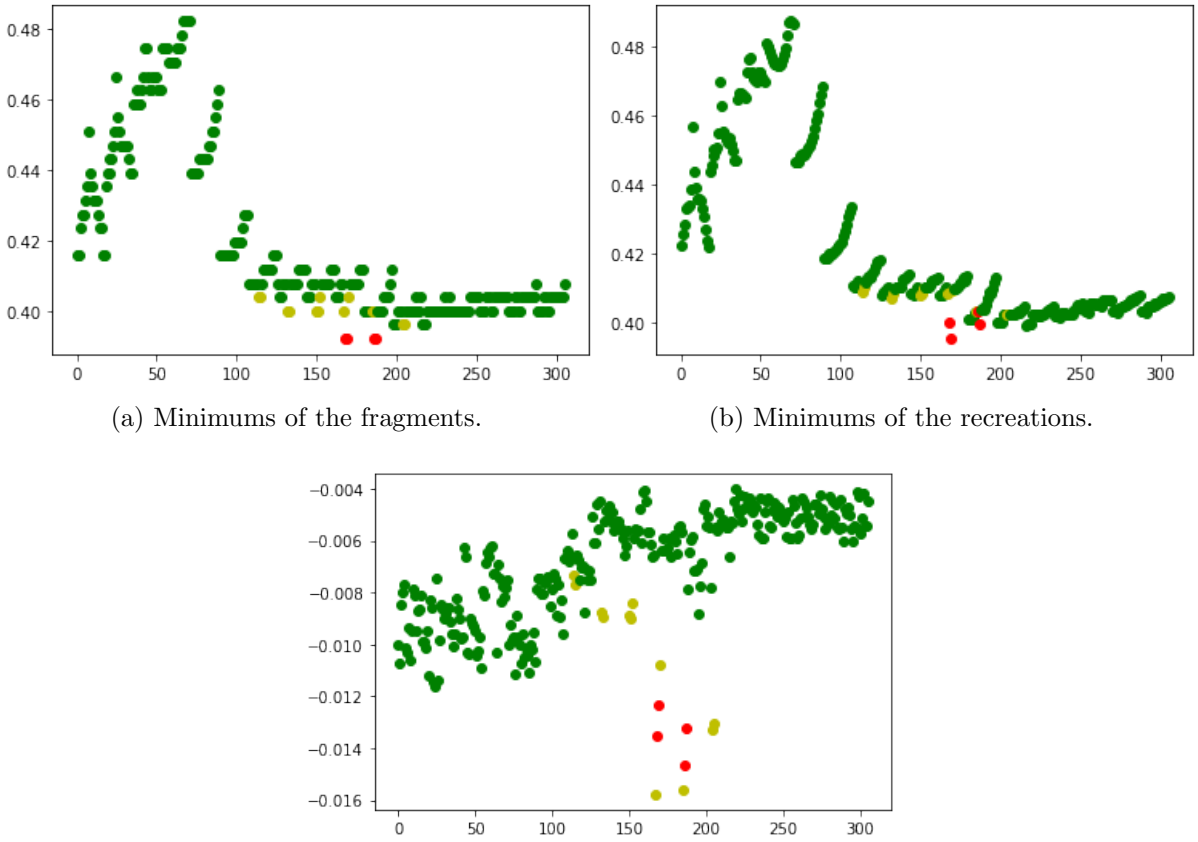
(c) Minimums of the differences.

Figure 4.14: The plots of the minimums for the first cropping.

By looking at the differences for the first crop in figure 4.14, the minimal values of the fragments appear to be the best ones thus far. All anomalies could easily be distinguished from their counterparts. Even the minimums in the raw fragments were decently successful. By looking at figure 4.15 below, the second crop complicated this a bit, however most of the anomalous fragments could still be distinguished with the differences.

An alternative thus could be to establish a threshold of -0.0125 and say that any difference with a smaller minimum value should be considered as an anomaly and anything above as non-anomalous. This would however not catch all the anomalies, but most of the red ones.

Another "good" thing is that the raw fragments fared less well in the second crop, meaning that it were not a reliable alternative and that the model provided something that the raw data could not.



(a) Minimums of the fragments.

(b) Minimums of the recreations.

(c) Minimums of the differences.

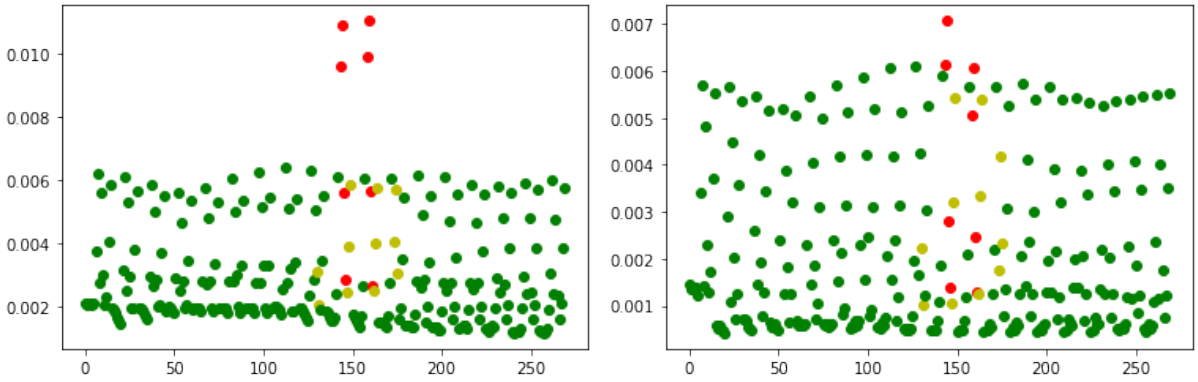
Figure 4.15: The plots of the minimums for the second cropping.

The standard deviation of the reconstruction errors

The standard deviation might be the theoretically most interesting alternative since it studies the spread of the pixels. This is logically intuitive since if a fragment has a higher spread, it would mean that there are bigger differences in the image which could be a hint for something anomalous.

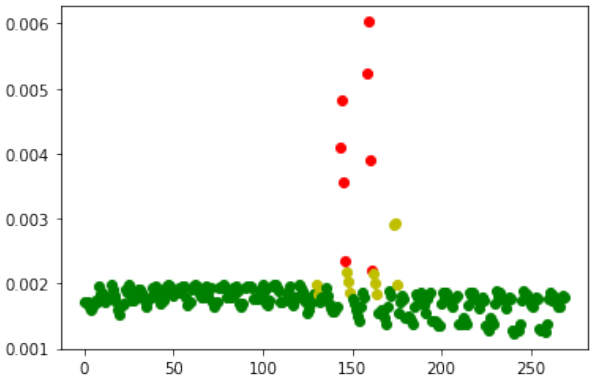
The following results for the first cropping in figure 4.16 also hint on how this could be a possibility. The result with the differences is decently clear since all obvious anomalies are detectable while only some of the uncertain anomalies are not. The raw fragments and recreations fared worse as well.

Note: The standard deviation always defines a positive and a negative value. These will always be the same (aside for the sign), so only the positive is studied here.



(a) Standard deviations of the fragments.

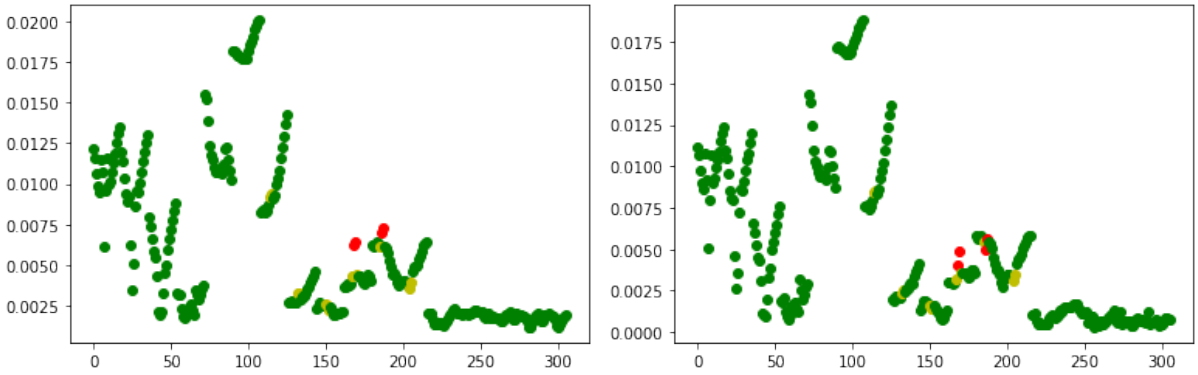
(b) Standard deviations of the recreations.



(c) Standard deviations of the differences.

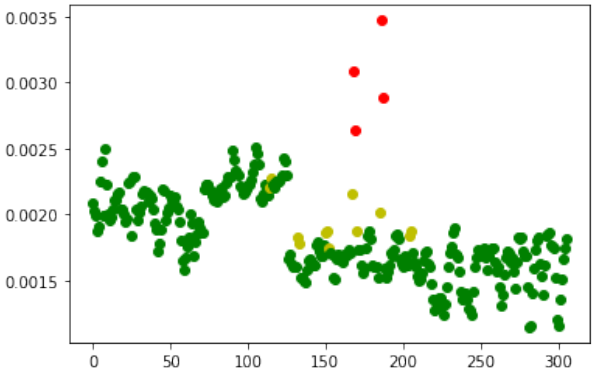
Figure 4.16: The plots of the standard deviations for the first cropping.

This however got more problematic when studying the second cropping in figure 4.17, since only the obvious anomalies are clear.



(a) Standard deviations of the fragments.

(b) Standard deviations of the recreations.



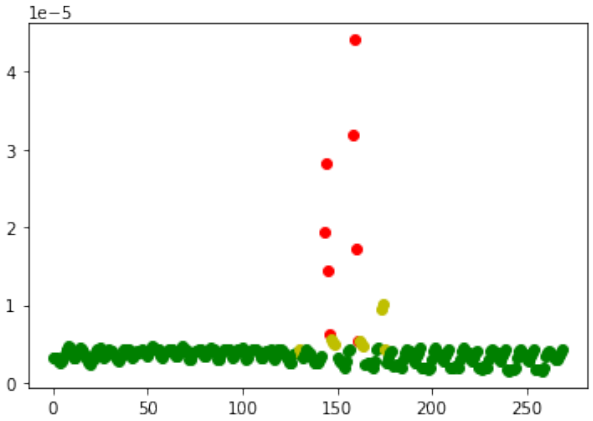
(c) Standard deviations of the differences.

Figure 4.17: The plots of the standard deviations for the second cropping.

The mean squared error of the reconstruction errors

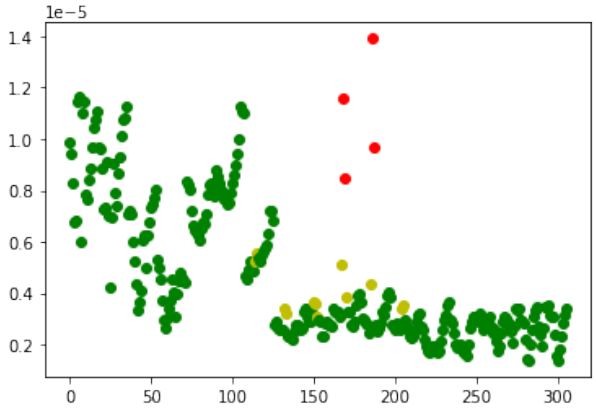
The final one used was the mean squared error. This aggregation is commonly used as the loss function for machine learning models.

By looking at the first cropping it showed good signs of usability, which can be seen in figure 4.18. However similar problem arose as with the other alternatives where the second cropping performed significantly worse, which can be seen in figure 4.19.



(a) Mean squared errors of the differences.

Figure 4.18: The plot of the mean squared errors for the first cropping.



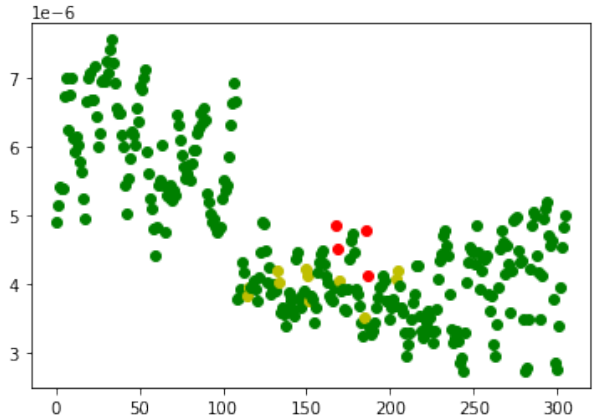
(a) Mean squared errors of the differences.

Figure 4.19: The plot of the mean squared errors for the second cropping.

4.2.2 Results after the second training

In order to get a better understanding of the model's potential, a second training session was conducted. The idea here was that the model should become better at distinguishing between the various types of anomalies. However, that did not turn out to be the case.

What instead happened was that it generally performed worse than what the previous training session did. Following figure shows an example of how it performed using the mean squared error as aggregation on the second cropping. Note that this is not exclusive for the mean squared error, but also true for all other methods.



(a) Mean squared errors of the differences.

Figure 4.20: The plot of the mean squared errors for the second cropping.

Notice how the red anomalies seen in figure 4.19 are much less distinct in figure 4.20.

5. Discussion

The final sections will discuss the results of the project, the flaws and the strengths of this solution, as well as possible actions that could be done in order to further advance this project.

5.1 Interpretation of the results

With the results concluded there is a lot to unpack. While no specific alternative for anomaly detection provided a "single solution to solve them all", there was a lot of potential with a few of them.

However, to truly find which alternative is most suitable there needs to be further analysis done on more croppings and with different test images, as well as reconsidering the work proposed in section 5.3 below. Thereby no alternative will be determined as the solution yet.

5.1.1 Comparing the usage of the raw data to the predictions

Using the raw data as a mean of anomaly detection could maybe be done, but with the given results it fared far less well than with the usage of the model's predictions. In some of the alternatives the raw data could be used for extracting the clear anomalies, however it was less distinct than the predicted counterpart. Since the model worked better with more alternatives, means that it managed to better generalize to the input data and will most likely work for more future cases as well. If further improvements are made on to the model according to section 5.3, then there will most likely not be any competition between the two.

This aligns with the proposed question in section 1.4. Where the question was if machine learning was necessary, and given the studied tools, it most certainly was. While there might exist even better ways of using the raw data, chances are that machine learning will still be the most optimal.

5.1.2 The sums

Initially in the beginning of the project the idea was to simply take the sum of the reconstruction errors and use that for anomaly detection (simply because its intuitive), however it fared quite poorly for the second cropping.

5.1.3 The minimums

The real success story is when looking at the minimum values, which actually turned out quite well for the studied image and two of its croppings. However, the question is how well it adapts for other croppings. Chances are that an anomaly could have different minimum values for reasons such as how the x-ray image was taken and on what material and where in the image it appeared on. Due to which it might be challenging to decide a good threshold.

5.1.4 The standard deviations

The standard deviation also succeeded fairly well. Similar to most alternatives it struggled a bit with identifying the unsure cases, but it has future potential.

5.1.5 The mean squared errors

The results of the mean squared error turned out to be very similar to that of the sums. However, since the mean squared error exaggerates the differences between each corresponding pixel more, this alternative actually ended up being a lot clearer since it elevated the red anomalies further in the second cropping than what the sums were able to do. With a better model, this might actually be a successful alternative.

5.1.6 First training compared to the second training

As can be read in section 4.2.2, the second and longer training performed generally worse than the first one. The reasoning for this is most likely that the model learnt too much about the anomalies. Remember that even non-faulty fragments can still contain non-faulty anomalies, which probably laid the ground for the model's ability to learn how to reconstruct faulty anomalies, which lead to the reconstruction error decreasing.

5.2 Critique

This project is by no means not polarized in its qualities, but actually has a lot of good and bad things to consider.

5.2.1 The good

The biggest strength of the project is that it is a fairly simple and intuitive solution since it is just a standard convolutional autoencoder training on the decided preprocessed data. There are a lot of room for future advancements where a lot of change could be made. The project is by no means in a dead-end, but rather at a crossing where many decisions could be taken.

Since the results turned out fairly decently, there most likely are not too many changes that would be needed to be made either.

5.2.2 The bad

Most of the problematic things for this project persists in the fundamental methodology. Only a single model has truly been trained and tested, where there may be many more things that could be done to improve it. The project might also have benefited from a further, extensive theoretical study where a better model might have been identified from the start.

The model has also only tested one source of data, which is the fragments. It might be worthwhile to try on the larger images and see how it performs then.

Since only a small subset of the data was experimented on, it might be hard to pull too many conclusions about the results. There also exist better ways of studying the results

rather than manual labeling and using scatter plots. (Another method that was used in this project was to mark the predicted anomalies in the image rather than using plots, however since the images can not be shown we had to settle for this alternative instead).

5.3 Future work

The most import thing to consider is how the model is defined. We did not explore all the options for hyperparameters in this project, and there may be better ones out there that could significantly boost the quality of the anomaly detection.

A more advanced idea is to also create another model for object detection that can learn how to automatically crop the x-ray images, so that manual cropping is not needed.

For the anomaly detection it is discussed that a threshold is needed in order to separate non-anomalies from anomalies. For simplicity of this result and the discussion of it, this threshold is treated as a static value (e.g. "anything above 0.01 is bad"), however it might be worth to reconsider a variable threshold that depends on the data studied. The future cases of anomalies might differ from one another and thus could benefit from having a threshold taking that into consideration.

Lastly, the statistical alternatives tested for anomaly detection are very basic. There are likely more sophisticated ways to do the calculations, and probably a few good ways that the proposed alternatives could be combined and used together.

5.4 Sustainability and ethics

An important aspect to consider for all projects is its influence on sustainability and ethics. Often in the context of computer related projects it might be hard to relate to such questions, but machine learning is a clear exception.

5.4.1 Sustainability with usage of machine learning

A problem with machine learning is that the technology requires that a large set of processing units observes data for a reasonably long time in order to train its model and generalize its solution. The more computationally advanced the machine learning task is, the more processing units or time is needed, which requires more energy. This is in contrast to more traditional computations which might not even need a fraction of that energy to simply solve its task. However, most machine learning tasks does not require so much.

Furthermore, machine learning has shown a lot of potential and could definitely be applied to applications of sustainability. Such as climate change predictions that might tell a more clear story of the future or simulations of poverty that can help track the problems. If you put the additional computational power needed for machine learning towards a good cause, that might be a small cost for a large reward [29].

In the case of this project, the cost of training such a machine learning model is minuscule compared to the gains in quality and safety of the manufactured products.

5.4.2 Ethics with usage machine learning

The greatest challenge for machine learning is how it can be used for unethical practices. This includes but is not limited to; face detection, personalized advertisements [30] and predictions of side effects of medicine [11]. The biggest problems with these technologies is how they can be used for malicious activities (such as tracking people using cameras with face detection) [30] and the large amount of private data needed in order to train such a machine learning model [31].

Such ethical problems are however not the case for this project, since it only deals with anonymous data from manufacturing and is only used for increasing the quality of the manufactured products.

6. Conclusion

While this project may not have been concluded with a clear solution, it has put forth a decent prototype to move forward with, that is backed by results showing a clear trend of the possibilities.

The project has been based on ideas with a lot of theoretical backing, and has deployed a fairly intuitive methodology with room for uncomplicated improvements. The only thing needed to be reconsidered is the future work mentioned in section 5.3.

At the end of it all, computer vision is a phenomenal technology with lots of potential.

Bibliography

[1] G. Lewis, "Object Detection for Autonomous Vehicles." [Online]. Available: https://web.stanford.edu/class/cs231a/prev_projects_2016/object-detection-autonomous.pdf, [Accessed: 2020-05-30].

[2] B. Marr, "What is Industry 4.0? Here's A Super Easy Explanation For Anyone," [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/09/02/what-is-industry-4-0-heres-a-super-easy-explanation-for-anyone/#25b880e69788, [Accessed: 2020-05-30].

[3] GKN Aerospace, "Om GKN Aerospace i Sverige," [Online]. Available: https://www.gknaerospace.com/en/about-gkn-aerospace/locations/gkn-aerospace-in-europe/gkn-aerospace-in-sweden/, [Accessed: 2020-04-24].

[4] "NEMA PS3.1.1 / ISO 12052, Digital Imaging and Communications in Medicine (DICOM) Standard," National Electrical Manufacturers Association, Rosslyn, VA, USA, Tech. Rep., [Accessed: 2020-06-10].

[5] "NEMA PS3.10.1 / ISO 12052, Digital Imaging and Communications in Medicine (DICOM) Standard," National Electrical Manufacturers Association, Rosslyn, VA, USA, Tech. Rep., [Accessed: 2020-06-10].

[6] "NEMA PS3.10.7 / ISO 12052, Digital Imaging and Communications in Medicine (DICOM) Standard," National Electrical Manufacturers Association, Rosslyn, VA, USA, Tech. Rep., [Accessed: 2020-06-10].

[7] "NEMA PS3.6.6 / ISO 12052, Digital Imaging and Communications in Medicine (DICOM) Standard," National Electrical Manufacturers Association, Rosslyn, VA, USA, Tech. Rep., [Accessed: 2020-06-10].

[8] The HDF Group. (1997-2020). Hierarchical Data Format, version 5, [Online]. Available: https://www.hdfgroup.org/HDF5/. [Accessed: 2020-06-10].

[9] The HDF Group. (1997-2020). Hierarchical Data Format, version 5: User's guide, [Online]. Available: https://support.hdfgroup.org/HDF5/doc/UG/HDF5_Users_Guide.pdf. [Accessed: 2020-06-10].

[10] U. Dahlblom, *Matematisk statistik*. Gothenburg, Sweden: HB Matematiklitteratur, 2003.

[11] T. M. Mitchell, "The Discipline of Machine Learning." [Online]. Available: http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf, [Accessed: 2020-06-10].

[12] M. A. Nielsen, *Neural Networks and Deep Learning*. [Online]. Available: http://neuralnetworksanddeeplearning.com/, [Accessed: 2020-06-10].

[13] M. Thoma, "Analysis and Optimization of Convolutional Neural Network Architectures." [Online]. Available: https://martin-thoma.com/msthesis/, [Accessed: 2020-06-10].

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org, [Accessed: 2020-06-10].
[15] Dave Anderson and George McNeill, "Artificial Neural Networks Technology." [Online]. Available: http://andrei.clubcisco.ro/cursuri/f/f-sym/5master/aac-nnga/AI_neural_nets.pdf, [Accessed: 2020-06-10].
[16] C. W. Dawson and R. Wilby, "An artificial neural network approach to rainfall-runoff modelling." [Online]. Available: https://doi.org/10.1080/02626669809492102, [Accessed: 2020-06-10].
[17] The GIMP Team. (2002-2015). GIMP, documentation: 8.2. Convolution Matrix, [Online]. Available: https://docs.gimp.org/2.8/en/plug-in-convmatrix.html. [Accessed: 2020-06-10].
[18] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability." [Online]. Available: http://dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-03.pdf, [Accessed: 2020-06-10].
[19] *TensorFlow*. [Online]. Available: https://www.tensorflow.org/, [Accessed: 2020-06-10].
[20] *Keras*. [Online]. Available: https://www.keras.io/, [Accessed: 2020-06-10].
[21] *pydicom*. [Online]. Available: https://github.com/pydicom/pydicom, [Accessed: 2020-06-10].
[22] *h5py*. [Online]. Available: http://www.h5py.org/, [Accessed: 2020-06-10].
[23] *OpenCV*. [Online]. Available: https://pypi.org/project/opencv-python/, [Accessed: 2020-06-10].
[24] *Matplotlib*. [Online]. Available: https://matplotlib.org/, [Accessed: 2020-06-10].
[25] *NumPy*. [Online]. Available: https://numpy.org/, [Accessed: 2020-06-10].
[26] *Pillow*. [Online]. Available: https://python-pillow.org/, [Accessed: 2020-06-10].
[27] *Visual Studio Code*. [Online]. Available: https://code.visualstudio.com/, [Accessed: 2020-06-10].
[28] *Project Jupyter*. [Online]. Available: https://jupyter.org/, [Accessed: 2020-06-10].
[29] R. Vinuesa et al., "The role of artificial intelligence in achieving the Sustainable Development Goals." [Online]. Available: https://www.nature.com/articles/s41467-019-14108-y.pdf, [Accessed: 2020-06-10].
[30] "Tesco face detection sparks needless surveillance panic, Facebook fails with teens, doubts over Google+," 2013-11-11. [Online]. Available: https://www.theguardian.com/technology/2013/nov/11/tesco-face-detection-sparks-needless-surveillance-panic-facebook-fails-with-teens-do, [Accessed: 2020-06-10].
[31] J. Risen and L. Poitras, "N.S.A. Collecting Millions of Faces From Web Images," 2014-05-31. [Online]. Available: https://www.nytimes.com/2014/06/01/us/nsa-collecting-millions-of-faces-from-web-images.html, [Accessed: 2020-06-10].