Faculty of Science and Technology

UNIVERSITY OF LIMOGES

Internship report

Speciality : **(ACSYON)**

Development of deep learning algorithms in computer vision to reduce the need for labelled data in the steel industry

Presented By

Cleque Marlain MBOULOU-MOUTOUBI

Encadrant : **Mr JALARD Benoit**

Presented the : **13/09/2024**

Annee Universitaire : 2023/2024

Faculty of Science and Technology

UNIVERSITY OF LIMOGES

Internship report

for the degree of Master in Applied Mathematics **(ACSYON)**

Development of deep learning algorithms in computer vision to reduce the need for labelled data in the steel industry

Presented By

Cleque Marlain MBOULOU-MOUTOUBI

The Supervisors' Signatures

Host company supervisor **Mr. Benoit JALARD**	Academic supervisor **Mr. Thibault LIARD**
date:	date:

Academic Year : 2023/2024

This document is based on a LaTeX template.[1]

[1]Template source: https://github.com/MarouaneElkamel/RapportPFE/blob/master/rapport.pdf.

ACKNOWLEDGMENTS

This work would not have been possible without the valuable cooperation of a number of people I would like to pay tribute to.

I would like to thank all those who have made this internship a rewarding and enjoyable experience, especially :

I would like to express my deep gratitude to **Mr. Yvon FONTAINE** for the support he provided during our monthly meetings. His advice and ideas greatly contributed to the progress of the project. I also thank him for the motivation he instilled in me, especially through our shared passion for running.

I also wish to thank **Mr. Laurent DOREL** for his contributions throughout my internship, particularly during our biweekly Scrum meetings. These meetings, where each member of the **SIAS team** – whom I also wish to thank – presented their work from the previous two weeks, were key moments of collaboration and exchange.

I am deeply grateful to my host company, **CLECIM**, for the opportunity to work in the field of Artificial Intelligence, a field that I am deeply passionate about.

I wish to extend special thanks to my internship supervisor, **Mr. Benoit JALARD**. From the very beginning, he placed his trust in me, guided me with great pedagogy throughout my stay in Montbrison, and his support was essential to the success of my work. I am also thankful for his support beyond the scope of the internship, whether it was during our cycling challenges, like the 17 km ride up to a mountain pass, or our swimming sessions on Tuesdays at noon.

Finally, I also express my sincere appreciation to the members of the jury for accepting to evaluate my work.

TABLE OF CONTENTS

LIST OF FIGURES

L'inspiration est la solution spontanée d'un problème longuement médité
Napoléon Bonaparte

LIST OF TABLES

Abstract

Image classification methods using deep learning have achieved performance close to human perception. These methods are generally divided into two main steps: automatic feature extraction from images and classification of these extracted features. The advent of neural networks, particularly Convolutional Neural Networks (CNNs), has significantly accelerated these steps. In this report, we present a learning strategy that employs Autoencoder Feature Extraction for Classification (AFEC) . This method was initially detailed in an article I recoved at the end of my intership a article [8] which presente the same strategy for another data and my research was carried out without taking this into account.

This report outlines the implementation schema for the Autoencoder Feature Extraction for Classification model, discusses the challenges encountered in selecting suitable reconstruction metrics for autoencoders, and evaluates their performance. We address the internship's primary research question by assessing the robustness of different models when faced with reduced training data for the classifier. Finally, we present the results of recent research, which may serve as a foundation for future work.

ABBREVIATIONS & ACRONYMS

- **CNN:** Convolutional Neural Networks
- **AE:** Autoencoder
- **SIAS:** Automated Surface Inspection System
- **MSE:** Mean square error
- **NMSE:** Normalised Mean square errors
- **SSIM:** Structural SIMilarity
- **DZL:**Defect Zone Localization
- **AFEC:** Autoencoder Feature extraction for Classification
- **G_Accuracy:** Global Accuracy
- **ResAE:** Residual Autoencoder

CHAPTER 1

PROJECT SCOPE

Plan

Introduction

This chapter is dedicated to thepresentation of the host company (CLECIM) and of the product on which I worked (SIAS) and its limitations

1 Host Company Presentation

In the first section of the report, we will introduce CLECIM, the host company that has made my intership.

1.1 Presentation of Clecim SAS

Clecim, a company established in 1917, employs approximately 210 people. It is a renowned supplier for carbon steel and stainless steel processing lines, rolling mills, as well as mechatronic

products and metallurgical services

With its factory located in France, Clecim is capable of manufacturing mechatronic products and providing maintenance and repair solutions for key components.

Clecim, through its factory, offers a wide range of services such as welding, machining, assembly, piping, and testing. By collaborating with its engineering expertise, Clecim can continuously expand its offerings, including into other sectors such as pneumatics, forging, and the naval industry.

Figure 1.1 – Clecim

Clecim

41, route de Feurs,
42600 Savigneux Cedex
CS 50099,
Tél : 04 77 96 63 00
Fax : 04 77 96 63 63

Société par Actions Simplifiées (SAS)
au capital de 5 000 000 d'euros.

Code APE : 295A
N° SIRET : 324 905 165 RCS Saint-Etienne
N° TVA : FR 46 324 905 165

1.2 Main activities

Clecim designs, manufactures, assembles, and tests various products used in steel processing processes, some examples of which are:

- Process lines and cold rolling mills
  - Pickling lines
  - Galvanizing lines (example shown in the image below)
  - Electrolytic cleaning lines
  - Etc.

Figure 1.2 – Galvanizing lines

- Mechatronic products
  - SIAS (Automatic Surface Inspection System)
  - Laser welders
  - Sheet levelers
  - Etc.

Clecim also offers a variety of metallurgical services:

- Supply of spare parts with a project or individual equipment and also original spare parts, manufactured from drawings.
- Long-term service contracts: maintenance contract, expertise and modernization, training
- Repair of strategic mechanical equipment
- Small projects

1.3 Clients and Markets

Clecim's clients are in the metal processing industry and include all global steelmakers: ArcelorMittal (Luxembourg), Posco (South Korea), Nippon Steel (Japan), Tata Steel, Baosteel (China), etc.

Through its Services activity, Clecim also shares its expertise with other industries: Messier Bugatti, the tire industry with Michelin, plastics, steel, cement, paper, foundry, forging, naval industry...

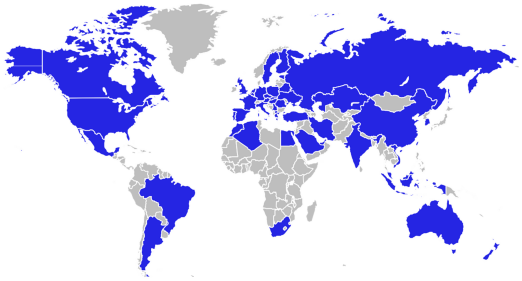


Figure 1.3 – Countries with Clecim references

1.4 Computer Vision

Computer vision is a key technology used by Clecim to enhance the performance of its quality control systems. It enables the processing and analysis of images captured by cameras to detect and characterize defects on steel sheets.

The company is renowned for its automated surface inspection systems **SIAS**, which leverage the latest advancements in deep learning and computer vision. These solutions allow for the detection of surface defects with high precision and optimize manufacturing processes.

2 SIAS's Presentation

2.1 History

The history of SIAS began over 30 years ago with the launch of the first SAVE prototype in 1994, following a decade of collaboration with MATRA MS2I. Several evolutions have since been developed: Xline®, launched in 2000, was later improved and renamed SIAS Xline® NextGen in 2012.

The year 2017 marked a turning point when Clecim decided to embark on research and development of SIAS DeepLearning, integrating Artificial Intelligence into the product. This decision led to the launch of the first prototype in 2019 at the ArcelorMittal facility in Liège, and the global offering of the product to customers starting in 2022.

2.2 Operation of the SIAS

The SIAS (Automatic Surface Inspection System) is a mechatronic product composed of a hardware and software architecture designed to collect and analyze data. This product is entirely conceived, developed, and assembled by Clecim. The system is used for non-destructive quality control by companies specializing in the processing of flat steel. The data collected consists of images captured by cameras installed on steel processing lines, such as galvanization lines, pickling lines, hot rolling mills, etc.

These images allow, through the system's computing component, the visualization of steel coils unwinding under the system's cameras in the form of strips, specifically indicating the presence or absence of defects at each point of the coil. All defects are detected, analyzed, classified into categories, and analysis reports are generated.

The artificial intelligence integrated into SIAS DeepLearning, with its neural network, is capable of detecting defects that were previously undetectable by earlier versions of SIAS, even at a line speed of up to 1400 m/min.

This enables automated control over the quality of the inspected products. The reliability and traceability of the results, as well as productivity and safety, are significantly improved compared to cases where this task is performed by an operator.

2.3 SIAS's Limitations

The SIAS DeepLearning system, like most deep learning models, relies on neural networks. A significant limitation of these models is the large number of images required to effectively train them. During my internship, I focused on addressing this challenge by exploring potential solutions to reduce the dependency on extensive datasets.

CHAPTER 2

PROJECT OVERVIEW

Plan

Introduction

To train a deep learning model, the first stage is data collection, in most cases it is necessary to collect thousands of relevant data, here images, to affine the model. For classification task, this work involves searching for data, cleaning it and, above all, annotating it by an experimented human operator who has not often much time to do this, is a challenge particularly important in the case of images, which is generally more costly to obtain, manipulate and especially annotate.

1 Project Context

The Automated Surface Inspection System (SIAS in French) is designed to detect and classify defects in steel plate. The defect detection algorithm is performing optimally, with satisfactory results, and is designed to identify surface anomalies.

However, although the classification results are also promising, one challenge remains: image labelling, which is a time-consuming and error-prone task. This is why the aim of my internship is to develop deep learning models capable of efficiently detecting and classifying surface defects, while reducing dependence on labelled image databases.

2 Main Project Idea

Clecim arrives to collect a large quantity of steel's images, but due to time constraints it is impossible to give a label at each image. We has got a large unlabelled data and a limited labelled data to build a deep learning model able to classify the previous unlabelled data.

The main objective is to reduce dependence on labeled images. To achieve this, we propose the following strategy to build a self-supervised learning model:

How can we use our data effectively?

We have a large amount of unlabeled data. Our first objective is to find a way to utilize it effectively, which involves identifying a deep learning algorithm capable of working with unlabeled data.

We decided to train a deep learning model to compress this unlabeled data. By doing so, the model will learn the most important features of the unlabeled images in in a small-dimensional space, making it easier for the model to generalize to unseen images of the same type.

Among the various deep learning algorithms, we found that autoencoders were the most suitable for this compression task due to their simplicity and effectiveness.

2.1 Pretraining an Autoencoder (AE) on unlabelled data

We start by pretraining an autoencoder model to learn a compressed representation of the most important information in an image for reconstruction. The autoencoder consists of two main components:

- **Encoder:** This part compresses the image into a small-dimensional space (latent space), capturing essential features and reducing dimensionality.
- **Decoder:** This part reconstructs the original image from the latent space representation, ensuring that the compressed information can be expanded back into a complete image.

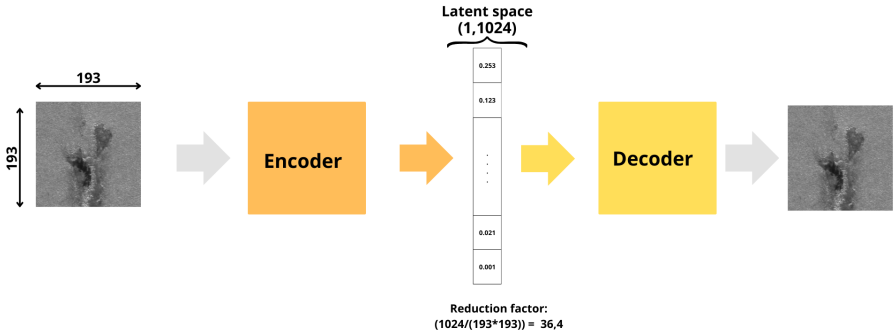


Figure 2.1 – Autoencoder process

The pretraining process helps the model to learn meaningful representations of images, which can then be used for downstream tasks such as classification or anomaly detection, without the need for extensive labeled data.

2.2 Train a classifier on limited labeled data

At this stage, we leverage the previously trained encoder, keeping its weights frozen. We then connect this frozen encoder to a classifier, and train a classifier model with full data labelled. The objective is to utilize the latent space representation, which offers a concise and informative summary of the image, to train a classifier for predicting the image's class.

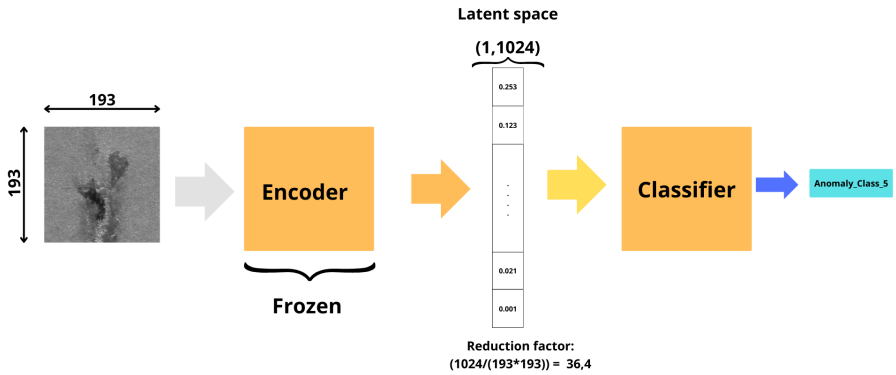


Figure 2.2 – Process involving a frozen encoder and a classifier

2.3 Simply the pretrained autoencoder

When using an autoencoder to compress an image into a vector, the image shape is reduced by a factor of 36. To simplify our classifier, it's necessary to decrease its input shape, which is the latent space. The procedure aims to compress the most significant information of the image into a smaller space. Thus we do again 2.1 and 2.2.

The goal of this procedure is to find the most adaptated shape of latent space (in Autoencoder) given the best classification result.

2.4 Reduction of labeled data of classification

It's the main stage of the internship, here we train again our best classifier model, using only limited labelled data, we compare the gap between this limited classifier and the classifier using full data. We will evaluate also, the robustness of this model into overfitting.

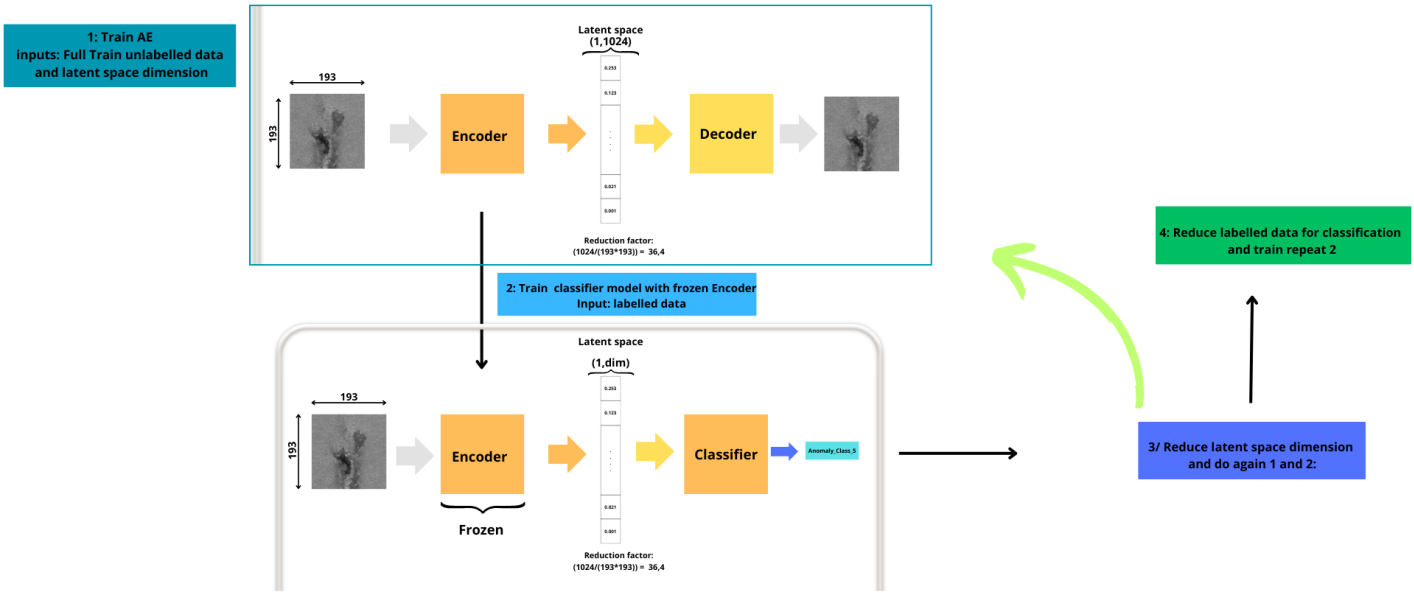


Figure 2.3 – Training procedure

CHAPTER 3

PRELIMINARIES

Plan

1 Deep learning for computer vision

Deep learning is a branch of artificial intelligence that utilizes deep artificial neural networks to perform complex machine learning tasks. In the realm of computer vision, deep learning can be applied to process and analyze images or videos, extracting relevant features for various tasks such as input reconstruction, denoising, generation, and classification. In our specific context, we concentrate on reconstruction and classification tasks for sheet steel[7].

1.1 Convolutional neural networks

Convolutional Neural Networks (CNNs) are one of the most widely used architectures in computer vision. They are designed to process visual data through convolutional layers, which

apply filters to subsets of the image, thereby detecting local features such as edges, textures, and shapes [17].

Definition 1.1 ***(Padding)***

Padding consists of adding zero borders around the input image to control the size of the output after convolution.

- ***Valid padding*** *: No borders are added, and the size of the output is smaller than the input.*
- ***Same padding*** *: Zeros are added around the input image to maintain the same output dimensions as the input.*

1.1.1 Convolution Layer

These layers perform convolution operations to extract features from the input data. The Objective is to downsampling an input [3] There are three types of convolutions: 1D Convolution (Conv1D) for processing sequential data or vectors, 2D Convolution (Conv2D) for image data, and 3D Convolution (Conv3D) for volumetric data such as video or 3D scans [10]. Here we will focus on 2D Convolution.

Definition 1.2 ***Convolution operation***

Given $\mathbf{X} \in \mathbb{R}^{n_1 \times m_1}$*, a filter (or kernel)* $f \in \mathbb{R}^{n_2 \times m_2}$ *(where* $n_1 \geq n_2$ *and* $m_1 \geq m_2$*), and padding* $p \in \mathbb{N}^*$ *(where* $p \leq n_1$*), the convolution operation* $(*)$ *of* $\mathbf{X}$ *by* f *with strides* $s = (s_1, s_2) \in \mathbb{N}_*^2$ *(where* $n_1 > s_1$ *and* $m_1 > s_2$*) is given by* $\mathbf{M} \in \mathbb{R}^{\left(\frac{n_1 - n_2 + 2p}{s_1} + 1\right) \times \left(\frac{m_1 - m_2 + 2p}{s_2} + 1\right)}$ *defined as follows:*

$$\mathbf{M}_{i,j} = (\mathbf{X} * f)_{i,j} = \sum_{l=0}^{n_2-1} \sum_{k=0}^{m_2-1} \mathbf{X}_{(i \cdot s_1 + l, j \cdot s_2 + k)} \cdot f_{l,k}$$

for $i \in [\![0, \frac{n_1 - n_2 + 2p}{s_1}]\!]$ *and* $j \in [\![0, \frac{m_1 - m_2 + 2p}{s_2}]\!]$*, where* $[\![a, b]\!]$ *denotes the set of integers from* a *to* b *inclusive.*

By default, the strides correspond to the size of the filters

Example 1.1 *Let's compute the Convolution operation between two arbitrary matrices* $\mathbf{X} \in \mathbb{R}^{3 \times 4}$*,* $f \in \mathbb{R}^{2 \times 2}$ *with stride* $s = 1$ *and padding =* $"Valid"$ *(*$p = 0$*) define as following:*

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{pmatrix} \quad \textbf{\textit{and}} \quad f = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

$\mathbf{M} = \mathbf{X} * f \in \mathbb{R}^{2\times 3}$

$M_{(i,j)}$	Formule	Résultat
$\mathbf{M}_{(0,0)}$	$\mathbf{M}_{(0,0)} = \sum_{l=0}^{1}\sum_{k=0}^{1} \mathbf{X}_{(0+l,0+k)} \cdot f_{l,k}$ $= 1\cdot 2 + 2\cdot 1 + 4\cdot 1 + 5\cdot 1$	13
$\mathbf{M}_{(0,1)}$	$\mathbf{M}_{(0,1)} = \sum_{l=0}^{1}\sum_{k=0}^{1} \mathbf{X}_{(0+l,1+k)} \cdot f_{l,k}$ $= 2\cdot 2 + 3\cdot 1 + 5\cdot 1 + 6\cdot 1$	18
$\mathbf{M}_{(0,2)}$	$\mathbf{M}_{(0,2)} = \sum_{l=0}^{1}\sum_{k=0}^{1} \mathbf{X}_{(0+l,2+k)} \cdot f_{l,k}$ $= 3\cdot 2 + 1\cdot 1 + 6\cdot 1 + 1\cdot 1$	14
$\mathbf{M}_{(1,0)}$	$\mathbf{M}_{(1,0)} = \sum_{l=0}^{1}\sum_{k=0}^{1} \mathbf{X}_{(1+l,0+k)} \cdot f_{l,k}$ $= 4\cdot 2 + 5\cdot 1 + 7\cdot 1 + 8\cdot 1$	28
$\mathbf{M}_{(1,1)}$	$\mathbf{M}_{(1,1)} = \sum_{l=0}^{1}\sum_{k=0}^{1} \mathbf{X}_{(1+l,1+k)} \cdot f_{l,k}$ $= 5\cdot 2 + 6\cdot 1 + 8\cdot 1 + 9\cdot 1$	33
$\mathbf{M}_{(1,2)}$	$\mathbf{M}_{(1,2)} = \sum_{l=0}^{1}\sum_{k=0}^{1} \mathbf{X}_{(1+l,2+k)} \cdot f_{l,k}$ $= 6\cdot 2 + 1\cdot 1 + 9\cdot 1 + 1\cdot 1$	23

Tableau 3.1 – Example of Convolution operation

$$\mathbf{M} = \mathbf{X} * f = \begin{pmatrix} 13 & 18 & 14 \\ 28 & 33 & 23 \end{pmatrix}$$

Bellow, we have an example on a real steal image:

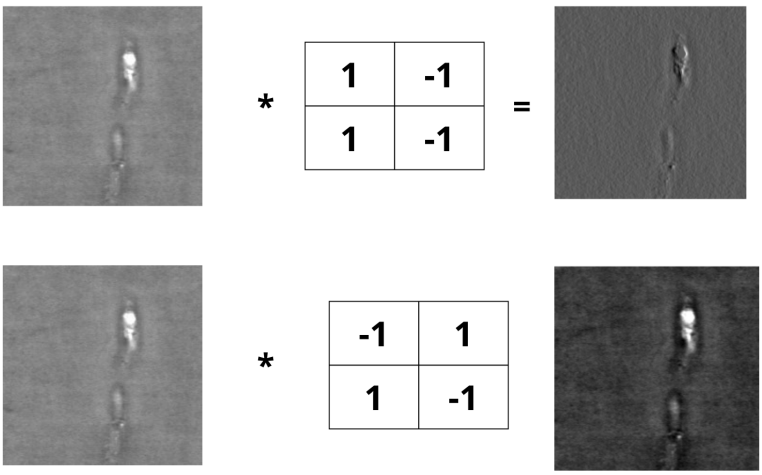


Figure 3.1 – Steel convolution example

Definition 1.3 ***Feature Map***

*A **feature map** is the output generated by a convolutional layer in a neural network. When an image passes through the convolutional layer, filters convolve with the input to produce 2D matrices, each highlighting different aspects like edges or textures. These matrices, collectively known as feature maps, capture the spatial structure and patterns within the image. For images, the feature map is represented as a 3D tensor.*

1.1.2 Pooling layer

Pooling layers are used in neural networks to reduce the spatial dimensions of feature maps, which helps decrease computational complexity and mitigate overfitting. By applying operations like **max pooling** or **average pooling**, these layers condense information by summarizing features over small, localized regions [6].

- **MaxPooling**: Selects the maximum value within each region of the feature map, highlighting the most prominent features.

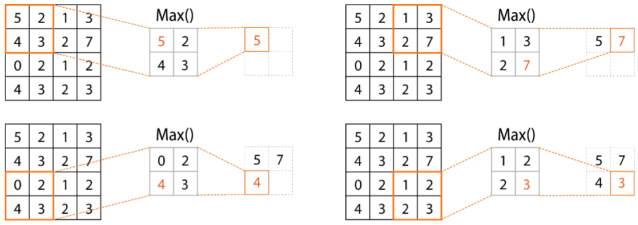


Figure 3.2 – MaxPooling Example: strides (2,2), [4]

- **Average Pooling**: Computes the average value in each region, providing a smoother feature representation.

Example 1.2 *The following Python code demonstrates how to create a Convolutional Layer using TensorFlow:*

```
import tensorflow as tf
from tensorflow.keras import layers

# Define the Convolutional Layer
conv_layer = layers.Conv2D(
    filters=32,           # Number of filters
    kernel_size=(3, 3),   # Size of the convolution kernel
    strides=(1, 1),       # Stride of the convolution
    padding='same',       # Padding method
    activation='relu'     # Activation function
)
```

Listing 3.1 – TensorFlow 2D Convolutional Layer

1.1.3 Fully connected layer

These layers, also known as dense layers, link all the units in the previous layer to those in the next layer to combine the extracted information. They are typically used in the final stages of a CNN to perform high-level reasoning.

1.2 Autoencoder Model

An autoencoder is a type of neural network designed to reproduce its input as closely as possible at its output. This involves encoding the input into a compressed latent representation and then reconstructing the output from this representation [15]. The learning process is self-supervised, as the objective is to minimize the reconstruction loss, which measures the difference between the input and the reconstructed output. Autoencoders are considered unsupervised models because they are trained using unlabeled data.

1.2.1 Autoencoder Architecture

The architecture of an autoencoder typically consists of two main components:

- **Encoder:** The encoder is responsible for compressing the input data into a lower-dimensional latent space representation. This is achieved through a sequence of convolutional layers, each followed by an activation function and a pooling layer. These layers work in tandem to progressively reduce the dimensionality of the input, thereby extracting the most salient features for subsequent processing.

  Here, **z** is a compact, encoded form of the input **X**.

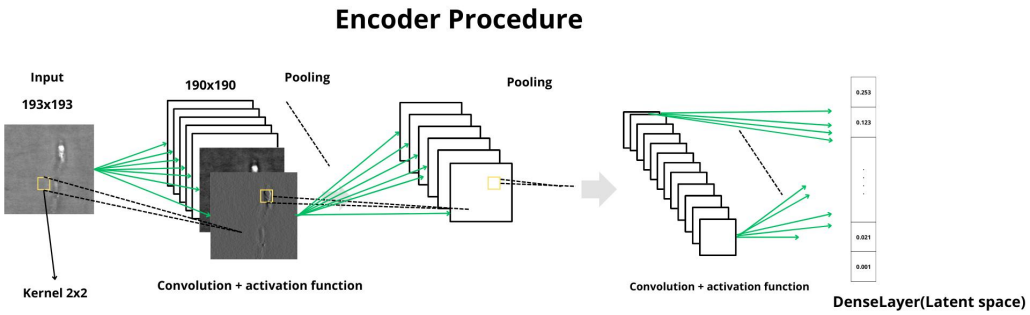


Figure 3.3 – Encoder

- **Decoder:** The decoder reconstructs the input data from the latent representation. It is a mirrored version of the encoder, with layers that progressively increase the dimensionality to return to the original input space. The decoder function $g(\cdot)$ takes the latent representation $\mathbf{z}$ and generates a reconstructed output $\hat{\mathbf{X}}$:

$$\hat{\mathbf{X}} = g(\mathbf{z})$$

The goal is for $\hat{\mathbf{X}}$ to be as close as possible to the original input $\mathbf{X}$.

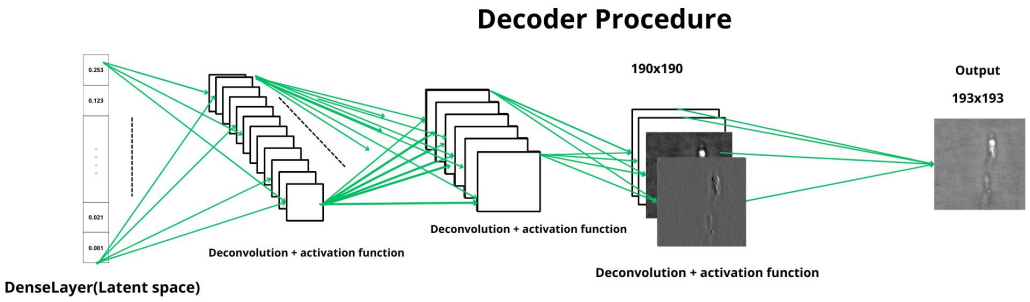


Figure 3.4 – Decoder

1.2.2 Applications and Variants

Autoencoders have a range of applications in feature extraction, dimensionality reduction, and anomaly detection. They are also foundational for various advanced architectures, such as Variational Autoencoders (VAEs) and Denoising Autoencoders. Each variant introduces modifications to the basic autoencoder structure to address specific tasks or challenges:

- **Variational Autoencoders (VAEs):**This variation of Autoencoder is introduced by Diederik P. Kingma and Max Welling [16]. It introduces probabilistic latent variables to allow for the generation of new data samples from the learned distribution in addition to image reconstruction.

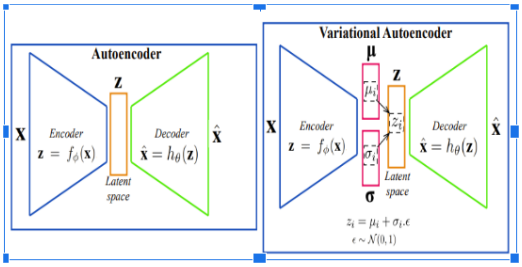


Figure 3.5 – AE VS VAE

A Variational Autoencoder (VAE) can also be used for anomaly detection by leveraging the alibi-detect library [2], which can be either installed directly or customized to meet specific requirements. The VAE is trained to reconstruct normal images. During inference, when an image is reconstructed, a reconstruction score is calculated by comparing the reconstructed image to the original one. Anomalies are detected based on whether this score exceeds a predefined normality threshold.

- **Denoising Autoencoders:** Train on corrupted input data and learn to reconstruct the original, clean data, enhancing the model's robustness.
- **Sparse Autoencoders:** Encourage sparsity in the latent representation, which can lead to more interpretable features.

1.3 Classifier Model

The classifier model is structured in two distinct parts:

1. **Feature Extraction**: The first part involves extracting key features from the input data, using several layers of CNN in the case of images.
2. **Classification**: Once the features have been extracted, the second part of the model uses them to classify the data into different categories, using several dense layers. A Dense layer of N neurons, where N is the number of possible classes, is added to the extracted features to make predictions.

1.4 Training Process

Training a deep learning model involves optimizing the model's parameters to minimize a loss function, which measures the error between the model's predictions and the actual target values [10]. The process typically consists of the following steps:

1.4.1 Loss function

The loss function $\mathcal{L}(\theta)$ is a measure of how well the model's predictions $\hat{y}$ match the true labels y. Common loss functions include Mean Squared Error (MSE) for regression tasks like reconstruction by an autoencoder, and Categorical Cross-Entropy for multi-class classification tasks [11, 12]. The Categorical Cross-Entropy quantifies the difference between the true label distribution and the model's predicted distribution across multiple classes. It's this loss function I used for the multi-class classification task, and it is defined as:

$$L = -\sum_{j=1}^{N}\sum_{i=1}^{C} y_{ji}\log(\hat{y}_{ji})$$

where:

- N is the number of samples.
- C is the number of classes.
- y_{ji} is the true label for the $j-th$ sample and $i-th$ class (1 if the class is correct, 0 otherwise).
- $\hat{y}_{ji}$ is the predicted probability for the j-th sample and i-th class.

The Categorical Cross-Entropy is particularly suitable for multi-class classification problems where each sample belongs to exactly one class. It encourages the model to output a probability distribution over all classes that closely matches the true distribution, which is typically a one-hot encoded vector for the correct class [5].

1.4.2 Optimizer

The optimization process updates the model's parameters (weights and biases) to minimize the loss function. This is typically done using gradient-based methods like Stochastic Gradient Descent (SGD), Adaptive Moment Estimation(Adam)1 widespread use. The parameter update rule is given by:

$$\theta \leftarrow \theta - \eta\nabla_\theta\mathcal{L}(\theta) \tag{3.1}$$

where η is the learning rate, and $\nabla_\theta\mathcal{L}(\theta)$ is the gradient of the loss function with respect to the parameters [5]. I used ADAM optimizer during each training.

Adam Optimization Algorithm

Adam maintains two moving averages for each parameter during training:

- **First Moment (m_t)**: The average of the gradients.
- **Second Moment (v_t)**: The average of the squared gradients.

The update rule for each parameter θ_t is:

$$\theta_{t+1} = \theta_t - \alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where:

- α is the learning rate.
- $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of m_t and v_t.
- ϵ is a small constant to prevent division by zero.

Algorithm 1 Adam Optimization Algorithm

Require: Learning rate α, decay rates β_1, β_2, small constant ϵ to avoid division by zero
Require: Initial parameters θ_0
Initialize first moment vector $m_0 = 0$
Initialize second moment vector $v_0 = 0$
Initialize time step $t = 0$
while not converged **do**
  $t = t + 1$
  Compute gradient g_t of the objective function at θ_{t-1}
  $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ {Update biased first moment estimate}
  $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ {Update biased second moment estimate}
  $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ {Compute bias-corrected first moment estimate}
  $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ {Compute bias-corrected second moment estimate}
  $\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ {Update parameters}
end while
return θ_t {Return optimized parameters}

Adam performs well due to its adaptive learning rate mechanism. It adjusts the learning rate individually for each parameter based on the first and second moments, allowing for:

- **Robustness**: Handles noisy gradients and sparse data efficiently.
- **Fast Convergence**: Quickly finds optimal solutions with fewer tuning requirements.
- **Versatility**: Works well for a wide range of deep learning architectures.

1.4.3 Training Process

During training, the model goes through the following steps in each epoch:

1. **Forward Pass**: The input data is passed through the model to generate predictions $\hat{y}$.
2. **Loss Calculation**: The difference between the predictions $\hat{y}$ and the true labels y is computed using the loss function.
3. **Backpropagation**: The gradients of the loss function with respect to the model's parameters are computed.

4. **Parameter Update**: The parameters are updated using the chosen optimization algorithm.

This process is repeated for several epochs until the model achieves a satisfactory level of accuracy or the loss function converges.

2 Data description

The data used in this project consists of images of steel plates, collected to develop models capable of detecting and classifying surface anomalies. These images serve as the foundation for analysis and training of deep learning models.

2.1 Image Characteristics

The steel plate images have the following characteristics:

- **Dimensions:** 212 x 212 pixels, providing sufficient resolution to capture surface details.
- **Format:** BMP, a lossless format that preserves image quality.
- **Total Number of Images:** 33,000, distributed as follows:
  - 8,000 images of normal steel plates, encompassing various normal surface textures.
  - 25,000 images of steel plates with anomalies, categorized into several defect types.

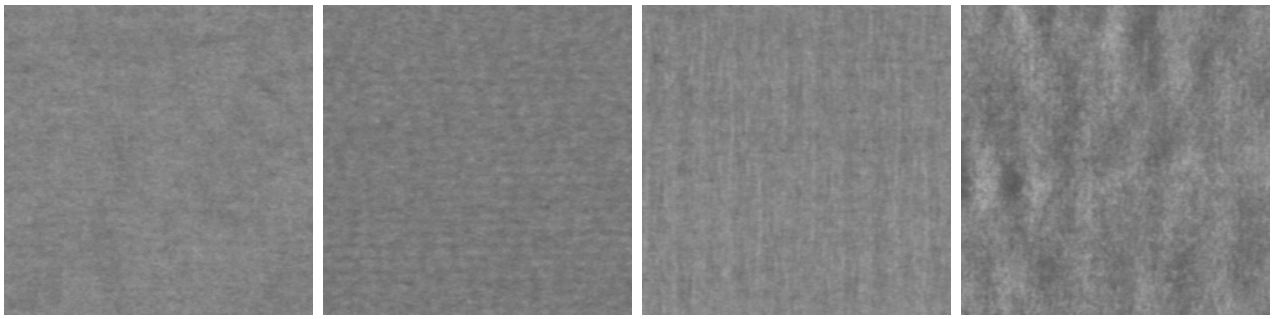
  The dataset is thus structured into N classes of normal images (textures) and M classes of anomalous images.

2.2 What is a defect?

Any deviation in the surface geometry, texture, or appearance of the inspected product that the end user finds unacceptable is classified as a defect. Defects can compromise the functionality, aesthetic appeal, or structural integrity of the product. In the context of galvanized steel strips, approximately thirty different types of flaws are commonly identified, including but not limited to:

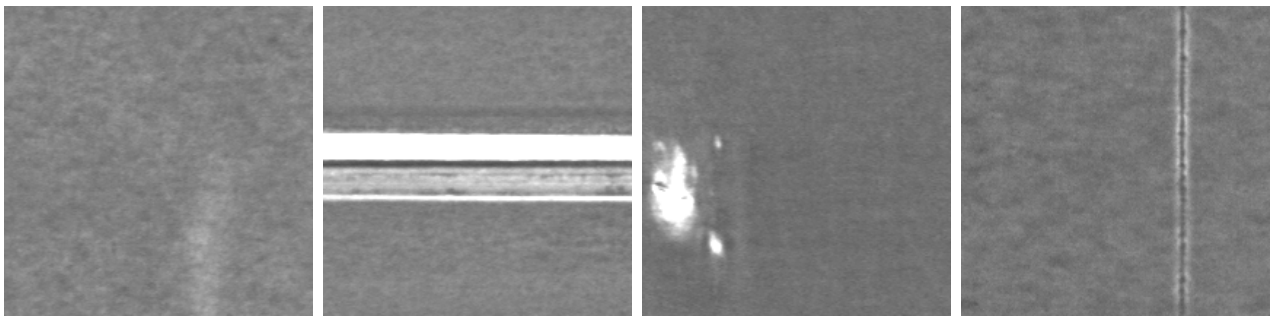
- **Cracks:** Linear defects indicating ruptures in the material or welds connecting two coils.

- **Scratches:** Superficial marks or abrasions typically resulting from friction between the steel surface and other objects during handling, processing, or transport. While often shallow, they can compromise surface treatments and coatings, potentially leading to corrosion.
- **Holes:** Unintended perforations or gaps in the material, which can arise from processing errors, localized corrosion, or mechanical damage. Holes compromise the structural integrity and can render the material unfit for its intended application.
- **Dents:** Indentations or depressions on the surface, usually caused by impacts or pressure during handling or transportation. Dents can alter the material's thickness and compromise its mechanical properties.
- **Blisters:** Raised areas on the surface caused by trapped gas or impurities during the galvanizing process. These can lead to peeling or flaking of the coating and expose the steel to corrosion.
- **Waviness:** Irregular, wave-like deformations on the surface of the strip, often resulting from uneven cooling or rolling processes. This can lead to difficulties in further processing and affect the final product's flatness.
- **Pinholes:** Very small holes or perforations, often too small to be visible to the naked eye, typically resulting from imperfections during the galvanizing process. Pinholes can lead to localized corrosion over time.
- **Pitting:** Small, localized craters or depressions on the surface, usually caused by corrosion or chemical reactions. Pitting can weaken the material and serve as initiation points for cracks.
- **Laminations:** Layers within the steel that have separated or failed to bond properly, often due to inclusions or impurities during the rolling process. Laminations can cause the material to split or delaminate during forming operations.
- **Edge Cracking:** Cracks that develop along the edges of the steel strip, often due to improper handling, overloading during processing, or inherent material weaknesses. Edge cracks can propagate into the material, leading to larger structural issues.
- **Roll Marks:** Repeated patterns or indentations on the surface, caused by imperfections or debris on the rolling equipment. These can affect the surface finish and, in some cases, the material's dimensional accuracy.



(a) Normal **(b)** Normal_2 **(c)** Normal_3 **(d)** Normal_4

Figure 3.6 – Common Normal surface textures



(a) Waviness **(b)** Crack **(c)** Hole **(d)** Cracks

Figure 3.7 – Common Defects in Galvanized Steel Strips

Each of these defects requires careful identification and evaluation during the quality control process to ensure that the galvanized steel strip meets the required standards and specifications.

2.3 Use of Data

These images serve as a basis for:

- **Model Training:** Using the images to develop anomaly detection models.
- **Evaluation:** Testing model performance on images not seen during training.
- **Validation:** Adjusting hyperparameters to optimize model accuracy.

2.4 Data Preprocessing

Before using the images for model training, several preprocessing steps were undertaken:

- **Duplicate Removal:** Eliminating duplicate images to avoid redundancy. Both the training (Train) and test (Test) sets contained two types of duplicates:

  - Name duplicates, which are easy to identify.
  - Data duplicates, which are identical images with different names. This category required special handling.

- **Normalization:** Adjusting pixel values to standardize the images.
- **Balancing:** Adjusting the proportions of normal and anomalous steel plates to improve model performance.

To facilitate effective image preprocessing in Python, I developed a pipeline within a file named **Preprocessing.py**. This pipeline is part of a suite of tools (**Tools_marlain**) I created during my internship. You can access and utilize this preprocessing pipeline in Python as follows:

The table below presents some of the key scripts used to build this pipeline:

Function Name	**Functionality**
`remove_dir`	Removes a directory and all its contents.
`Usage:`	`remove_dir(`*`dir_path`*`)`
`Example:`	`remove_dir('path/to/directory')`
`end_wanted`	Deletes files in a directory that do not end with the specified extensions.
`Usage:`	`end_wanted(`*`file_path`*`, endWtds)`
`Example:`	`end_wanted('path/to/directory', ['.bmp'])`
`remove_sample_in_all`	Moves images from a target directory to another directory after checking for duplicates from a source directory.
`Usage:`	`remove_sample_in_all(`*`source_file`*`, target_file, dir_to_move, Wtd, endWtds)`
`Example:`	`remove_sample_in_all('path/to/source', 'path/to/target', 'path/to/move', True, ['.bmp'])`
`remove_doubling`	Moves duplicate images from a source directory to a destination directory after filtering the specified extensions.
`Usage:`	`remove_doubling(`*`source_file`*`, dir_to_move, Wtd, endWtds)`
`Example:`	`remove_doubling('path/to/source', 'path/to/move', True, ['.bmp'])`

Function Name	Functionality
`split_images_folder`	Splits a folder containing image folders into two separate folders for training and validation.
`Usage:`	`split_images_folder(`*`file_path`*`, train_path, valid_path, Nmax, percent)`
`Example:`	`split_images_folder('path/to/data', 'path/to/train', 'path/to/valid', 150, 0.8)`
`take_sample`	Creates a subset of images by extracting a sample of a certain percentage from the source data.
`Usage:`	`take_sample(`*`file_path`*`, valid_path, percent, Nmax)`
`Example:`	`take_sample('path/to/source', 'path/to/valid', 0.2, 100)`

2.5 Models evaluation

To assess the quality of a new model for classification tasks, it is essential to establish a primary model as a benchmark at the outset. In this benchmark we define some metrics :

- **Global Accuracy**

  Definition 2.1 *The overall accuracy is the ratio of correct predictions to the total number of predictions on the test dataset.*

$$Global\ Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

- **Global Loss**

  Definition 2.2 *The overall loss measures the quality of the model's predictions on the test dataset, often as an average of the loss function calculated for each test example. If L is the loss function (e.g., categorical cross-entropy), then*

$$Global\ Loss = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{y}_i)$$

  where y_i is the true label and $\hat{y}_i$ is the model prediction for the i-th example.

- **Classification Accuracy**

Definition 2.3 *Similar to global accuracy, but specifically refers to the classification of defects families.*

$$Classification\ Accuracy = \frac{Number\ of\ correct\ classification\ predictions}{Total\ number\ of\ classification\ predictions}$$

- **Detection Accuracy**

Definition 2.4 *Detection accuracy reflects the model's ability to distinguish between normal and abnormal images.*

$$Detection\ Accuracy = \frac{Number\ of\ correct\ detections}{Total\ number\ of\ detections}$$

A detection is correct if IoU exceeds a certain threshold (often 0.5).

- **Number of Parameters**

Definition 2.5 *The total number of weights and biases in the model that are learned during training.*

$$Number\ of\ Parameters = \sum_{i=1}^{L}(W_i + b_i)$$

where W_i and b_i are the weights and biases of the i-th layer, respectively.

  - ***Convolution layer:*** *The number of parameters in a convolutional layer is calculated as:*

$$Number\ of\ Parameters = (k_h \times k_w \times c_{in} + 1) \times c_{out}$$

    where k_h and k_w are the height and width of the convolutional kernel, c_{in} is the number of input channels, and c_{out} is the number of output channels. The additional 1 accounts for the bias term [4].

  - ***Dense (Fully Connected) layer:*** *The number of parameters in a dense layer is calculated as:*

$$Number\ of\ Parameters = n_{in} \times n_{out} + n_{out}$$

    where n_{in} is the number of input units (neurons), and n_{out} is the number of output units (neurons). The additional n_{out} accounts for the bias terms [4].

The number of parameters helps us to assess the complexity of a model.

- **Number of Multiplications**

  Definition 2.6 *Measures the computational complexity of the model, indicating the total number of multiplication operations performed during a forward pass.*

  - ***Convolution layer:*** *The number of multiplications in a convolutional layer is calculated as:*

$$\text{Number of Multiplications} = (k_h \times k_w \times c_{in}) \times (H_{out} \times W_{out}) \times c_{out}$$

    where k_h and k_w are the height and width of the convolutional kernel, c_{in} is the number of input channels, H_{out} and W_{out} are the height and width of the output feature map, and c_{out} is the number of output channels [4].

  - ***Dense (Fully Connected) layer:*** *For a fully connected layer, the number of multiplications is calculated as:*

$$\text{Number of Multiplications} = \sum_{i=1}^{L} (n_{i-1} \times n_i)$$

    where n_{i-1} and n_i are the number of neurons in layers $i-1$ and i, respectively.

CHAPTER 4

ACHIEVEMENTS

Plan

Introduction

The quality control of industrial processes is a crucial aspect for companies like Clecim, which specialize in the processing of steel and other materials. In recent years, the integration of machine learning, particularly deep learning, has opened new possibilities for automating defect detection and classification. Among these, the SIAS (Système d'Inspection Automatique de Surface) system developed by Clecim, enriched with deep learning capabilities, aims to identify defects on steel surfaces.

One of the key challenges in developing effective deep learning models is the significant amount of labeled data required to train such systems, especially for tasks like image classification. This issue is particularly prominent when training autoencoders, which are designed to reconstruct images and compress their features into a latent space. Autoencoders are valuable not only for image reconstruction but also for feature extraction in classification tasks. However, identifying appropriate loss functions and reducing the reliance on large datasets are critical factors in improving model performance.

The goal of this project was to explore and develop reconstruction loss functions for autoencoders, analyze their impact on defect detection and classification tasks, and address the limitations posed by the availability of labeled data. The work presented in this report focuses on optimizing autoencoders for defect detection, improving classification performance by experimenting with various loss functions, and exploring strategies to reduce the dependency on large labeled datasets.

1 Searching reconstruction loss functions

The reconstruction loss function quantifies the discrepancy between input data and predicted data, it plays a crutial role during model traning. In this project, we conducted an extensive search for a suitable loss function that is well-adapted to detecting defects in steel using a fixed neural network architecture.

Measuring the quality of the algorithmic reconstruction seeks to is to design algorithms whose quality prediction is in line with the subjective assessment of human observers.

I have chosen two evalutions method to classify the quality of reconstructed image:

1. **Method with full reference**

   In this case autoencoder has an access to a perfect version of the image with which we want to compare to the reconstructed image which is often degraded by compression (by an encoder) and transmission errors.

2. **Contractive method**

   In contractive method, autoencoder has an access to the original image, in this case we will seach areas of interest on image and give a large weight to it compared to the rest of the image. Thus autoencoder will learn with a large rate to reconstruct risk patches in an image.

Among the various possibilities, we decided to focus on Mean Squared Error (MSE), Structural Similarity Index Measure (SSIM), and some regularized variants of these loss functions. Our goal was to identify the most effective loss function for accurately detecting and localizing defects in steel.

1.1 Full reference loss functions

1.1.1 Mean square error

During training, the autoencoder minimizes a reconstruction loss function, which typically measures the difference between the input $\mathbf{X}$ and the reconstructed output $\hat{\mathbf{X}}$. Common choice for the loss function include the Mean Squared Error (MSE) . The loss function $\mathcal{L}_1$ is defined as:

$$\mathcal{L}_1(X, Y) = \frac{1}{M} \sum_{i=1}^{M} \|X_i - Y_i\|^2$$

where M represents the number of data samples, and $\| \cdot \|^2$ is the squared Euclidean norm, which measures the difference between the original image Y_i and the reconstructed image Y_i.

To ensure that the reconstructed image preserves the same contrast as the original, we introduce a regularization term that penalizes the difference between the standard deviations of the original and reconstructed images. The modified loss function becomes:

$$\mathcal{L}_2(X, Y) = \mathcal{L}_1(X, Y) + \frac{\lambda}{M} \sum_{i=1}^{M} |\text{std}(X_i) - \text{std}(Y_i)|$$

where λ is a regularization parameter controlling the trade-off between reconstruction accuracy and contrast preservation, and $\text{std}(\cdot)$ denotes the standard deviation of the image.

1.1.2 Structural Similarity Index (SSIM)

Structural Similarity Index (SSIM) measures visual appearance between two images, because we want to learn identity function it stands to reason that the error function should be perceptually motivated. The main idea is to compare the similarity between each couples of patches (or windows) of target images. SSIM compares luminance, contrast and structure between each pair of windows [21],[20].

Definition 1.1 *(SSIM)*

Let's $X, Y \in \mathbb{R}_+^{N \times N}$ *original and reconstructed image and* $x, y \in \mathbb{R}_+^{n \times n}$ *two patches respectively of* X *and* Y*.*

$$SSIM(x, y) = l(x, y) \cdot c(x, y) \cdot s(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_x\sigma_y + c_2)(\sigma_{xy} + c_3)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)(\sigma_x\sigma_y + c_3)}$$

where

- l *is luminance;*
- c *is contrast;*
- s *is structure;*
- μ_x *(mean) which estimates luminance by measuring the average intensity of* x

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- μ_y *(mean) which estimates luminance by measuring the average intensity of* y*;*
- σ_x *(standard deviation) which measures contrast for each of* x

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2}$$

- σ_y *(standard deviation) which measures contrast for each of* y*;*
- σ_{xy} *(covariance) which measures similarity by estimating the loss of correlation between* x *and* y

$$\sigma_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

- $c_1 = (k_1 L)^2$*,* $c_2 = (k_2 L)^2$ *and* $c_3 = \frac{c_2}{2}$ *are variables used to stabilize the division when the denominator is very small;*
- L *is the dynamic range of the pixel values (e.g., 255 for 8-bit images);*
- $k_1 = 0.01$ *and* $k_2 = 0.03$ *by default.*

After simplification, substituting $c_3 = \frac{c_2}{2}$*:*

$$SSIM(x,y) = l(x,y) \cdot c(x,y) \cdot s(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Remark 1.1 .

1. $0 < SSIM(x,y) \leq 1$, $\forall x, y \in \mathbb{R}_+^{n\times n}$ *and* $c_1, c_2 > 0$
2. $SSIM(x,y) = SSIM(y,x)$, $\forall x, y \in \mathbb{R}_+^{n\times n}$
3. $SSIM(x,y) = 1$ *if* $x = y$

To measure the similariry between to images X and Y, we compute the Mean Structural SIMilarity (MSSIM) between all couple of windows of X and Y. Thus we have:

$$MSSIM(X,Y) = \frac{1}{M}\sum_{j=1}^{M} SSIM(x_j, y_j)$$

where M is the number of local windows in the image and $SSIM(x_j, y_j)$ is the SSIM calculated for the local windows x_j and y_j.

The loss of $MSSIM$ is defined such that:

$$\mathcal{L}_3(X,Y) = \frac{1}{M}\sum_{i=1}^{M}(1 - MSSIM(X_i, Y_i))$$

where M represents the number of data samples. I named this loss function **Loss_SSIM**.

We define also a regularization of **Loss_SSIM** by the norm L_1,

$$\mathcal{L}_4(X,Y) = \mathcal{L}_3(X,Y) + \frac{\lambda}{M}\sum_{i=1}^{M}\sum |flat(X_i - Y_i)|$$

Where $\lambda \in \mathbb{R}^+$. The function $flat(.)$ takes a matrix and flattens it into a one-dimensional vector. I call this loss **Loss_SSIM_L1**

1.2 Contractive Loss funtions

1.2.1 MSE weighted by z-score

When analyzing the images, our initial approach was to observe the distribution of pixel values. We aimed to identify pixel outliers because, in most cases, normal areas in a steel image are represented by gray pixels, which typically have values around 0.5. The following representation illustrates this analysis:

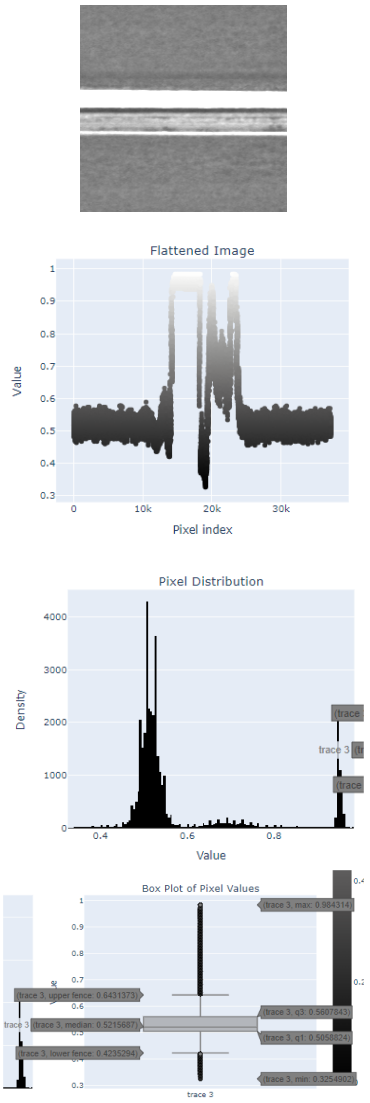


Figure 4.1 – Image analysis

Description:

1. **Original image**: This is a defect metallurgical sample. Lighter and darker areas represent higher and lower pixel values respectively, potentially indicating anomalies.
2. **Flattened image**: A scatter plot of the flattened pixel values. Each point represents a pixel value, with the x-axis showing the pixel index and the y-axis showing the pixel value. This plot shows the overall distribution of pixel values across the entire image.
3. **Pixel distribution**: A histogram displaying the frequency of pixel values. Most pixel values are concentrated around 0.5 to 0.6, with some extreme values reaching up to 0 or 1.
4. **Box plot**: A box plot providing statistical insights into the pixel values. It includes the median, quartiles, and outliers. The median pixel value is around 0.55, and there are several extreme values near 0 and 1.1.

With this statistical analysis, one idea is to give more importance to potential pixels hosting an anomaly. Among the various weighting techniques that sometimes use a threshold, we have

chosen to weight the image by a **z-score**.

Definition 1.2 *The **z-score** is a statistical measurement that describes a value's relationship to the mean of a group of values. It is measured in terms of standard deviations from the mean. A z-score indicates how many standard deviations an element is from the mean. The z-score is calculated using the following formula:*

$$z = \frac{X - \mu}{\sigma}$$

where X is the value, μ is the mean of the group, and σ is the standard deviation.

Application of the Z-Score in Image Weighting

This method uses a contractive approach 2 to compute the loss between an image X and its reconstruction Y. The z-score is used as follows:

$$\mathcal{L}_4(X, Y) = \frac{1}{M} \sum_{i=1}^{M} \| \frac{X_i - \mu_{X_i}}{\sigma_{X_i}} \cdot (X_i - Y_i) \|^2$$

where μ_{X_i} is the mean of the image X_i, and σ_{X_i} is the standard deviation of the image X_i.I calls this loss in the next: **Loss_MSE_z_score**

1.2.2 MSE and Defect zone localization (DZL)

Because the contractive approach produces better results for classification task as we shall see in **Section 3. (Analysis of results)** I decided to focus more one this part. Inspired by [18],[19], which deals with **'Bag of feature'** a technique used in computer vision and image processing to extract and represent features from images in a compact and meaningful way and **SSIM** contruction using image patches, I've built a mask statistic to localise potentially abnormal areas in an image.

Algorithm 2 Outlier mask calculation based on patch z-score

Require: *image*, *patch_shape*, threshold
Ensure: *resized_mask*
patches ← EXTRACT_PATCHES(*image*, *patch_shape*)
std_patches ← STD(*patches*)
z_score ← ZSCORE(*std_patches*)
mask ← *z_score* > *threshold*
outlier_indices ← CONVERTINBINARY(*mask*)
resized_mask ← RESHAPE(*outlier_indices*, *image_shape*)
return *resized_mask*

For each image, the threshold is defined by compute de standard deviation of standard deviation of patches (**STD(STD(patches))**.

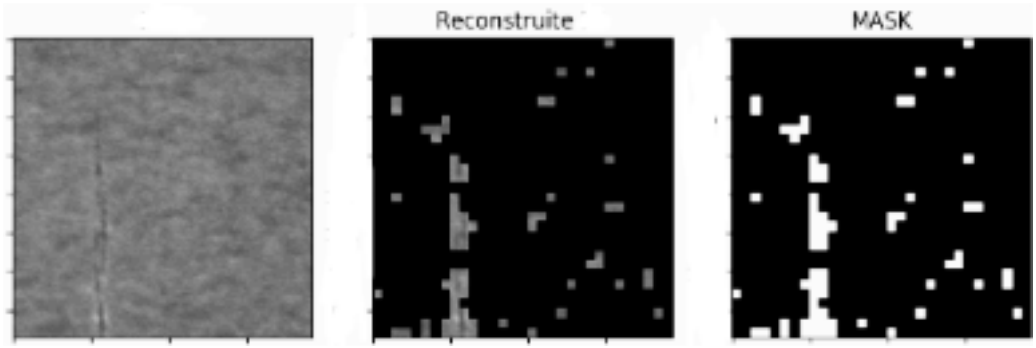


Figure 4.2 – An image, its reconstruction and its **mask_DZL**

The loss function constructed by this mask is defined as follows:

$$\mathcal{L}_5(X, Y) = \mathcal{L}_1(X, Y) + \frac{\lambda}{M} \sum_{i=1}^{M} \|\hat{X}_i \cdot (X_i - Y_i)\|^2$$

Where $\lambda \in \mathbb{R}^+$ and $\hat{X}_i$ is the mask of Defect Zone Localisation (**DZL**) of image X_i, and Y_i the reconstruction of X_i. I have named this loss function in the following: **Loss_MSE_DZL**

2 Analysis of results

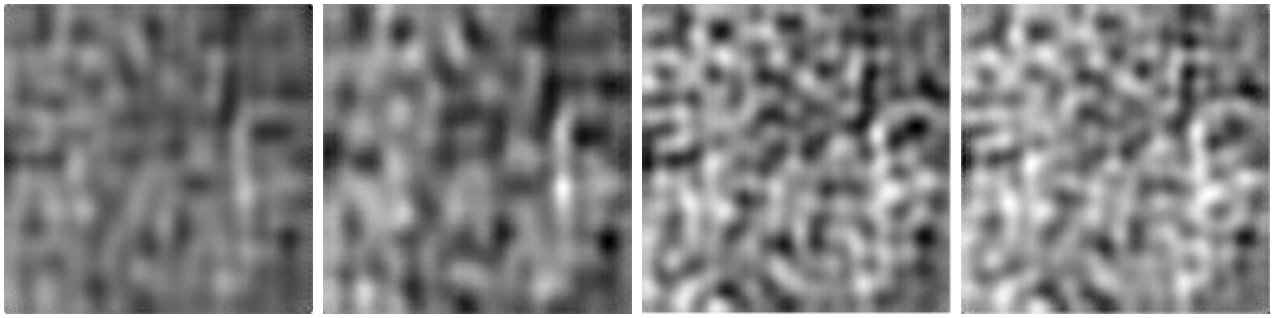
The process of reconstructing images using an Autoencoder (AE) is computationally intensive and requires considerable time to achieve satisfactory results. Identifying an optimal neural network architecture and, more importantly, the appropriate reconstruction loss function, involved numerous trials. Each training session could take up to **8 hours**, emphasizing the complexity and computational demands of the task. Below, I present a comparison of the reconstruction results obtained using different loss functions, highlighting their impact on the performance of the Autoencoder.

To evaluate the reconstruction quality of different loss functions, we defined a Normalized Mean Squared Error (**NMSE**) that measures how close the reconstructed images are to the expected mean value of 0.5. This indicator helps us monitor the training process and assess whether the Autoencoder is effectively reconstructing the images. A high normalized MSE can signal poor reconstruction, often resulting in the generation of blank (white) or entirely black images, which indicates a divergence in the Autoencoder's training. The NMSE if following

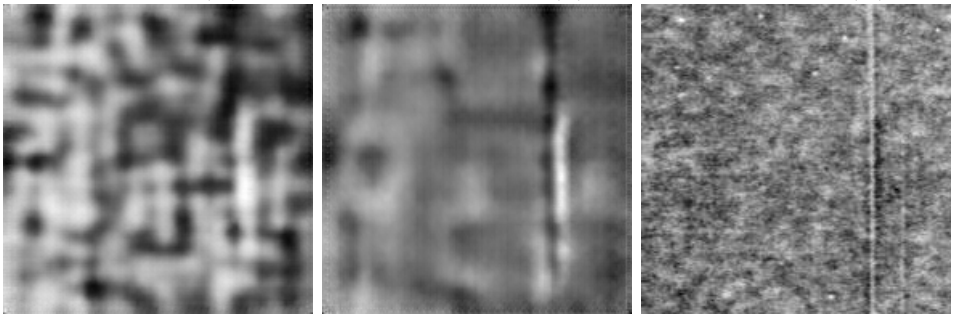
defined:

$$NMSE(X,Y) = \frac{1}{M}\sum_{i=1}^{M}\frac{\|X_i - Y_i\|^2}{\|X_i - 0.5\|^2}$$

- **Examples of recontructions**



(a) Loss_MSE **(b)** Loss_MSE_STD **(c)** Loss_SSIM **(d)** Loss_SSIM_L1



(e) Loss_MSE_Zscore **(f)** Loss_MSE_DZL **(g)** Original

Figure 4.3 – Reconstruction with different losses (2)

The color red and blue are for classification task.

- **Reconstruction and Classification Analysis**

Tableau 4.1 – Performance Metrics for Different Models

Model Name	Data	Class_Accuracy (%)	NMSE	G_Accuracy (%)	G_Loss
Model_Standard	Test	85.81	–	82.39	0.59
Model_1_MSE	Test	47.93	0.0976	49.02	1.63
Model_1_MSE_STD	Test	49.93	0.1253	49.78	1.55
Model_1_SSIM	Test	49.62	0.1246	48.1	1.59
Model_1_SSIM_L1	Test	48.19	0.099	50.2	1.56
Model_1_z_score	Test	66.93	0.1515	66.44	1.06
Model_1_MSE_DZL	Test	67.47	0.1176	65.32	1.13

As expected, models using global reference loss functions (**Loss_MSE, Loss_MSE_STD, Loss_SSIM**, and **Loss_SSIM_L1**) yielded the best reconstruction results when evaluated using the **NMSE**. However, these models performed poorly in classification, with

accuracy not exceeding 50%. Initially, we hypothesized that an autoencoder capable of accurately reconstructing images would have an encoder that effectively compresses the images, benefiting classification. However, in addition to reconstructing the entire image, these models also reconstruct the noise surrounding the defects, introducing unwanted black and white artifacts that add noise to the latent space.

The low classification performance demonstrates that global reference methods primarily minimize the overall reconstruction error across the entire image without focusing on regions of interest. This explains why, in 4.3, the reconstruction of a defect (a vertical scratch) is poorly handled, particularly in examples 4.3a, 4.3b, 4.3c, and 4.3d. Despite good image reconstruction, these methods fail to capture the critical features necessary for effective classification.

When we guide the model towards areas of interest using a loss function like **Loss_z_score**, which weights each pixel based on its **z_score**, or more effectively, the **Loss_MSE_DZL**, which weighs regions based on a determined threshold, we observe a slight decline in overall image reconstruction quality but an improvement in defect reconstruction. This is particularly evident in 4.3f. The use of **Loss_MSE_DZL** as a loss function results in filtered images, allowing for better image compression, especially focusing on the defects. This yields a cleaner latent space with reduced noise, aiding the model's performance.

Although the **Loss_z_score** loss function produces results comparable to **Loss_MSE_DZL**, it operates on the entire image, which leads to black patches appearing across the image, adding noise to the latent space.

Moving forward, we will fix **Loss_MSE_DZL** as the primary loss function to continue our study.

2.1 Effects of reducing latent space

The complexity of a deep learning model is often measured by the number of parameters it contains While a larger number of parameters can theoretically enable a model to learn more intricate patterns from the data, it also increases the risk of overfitting—where the model performs well on training data but fails to generalize to unseen data [12]. Overfitting occurs because the model becomes excessively attuned to the noise and peculiarities of the training data rather than capturing the underlying distribution [5].

Reducing the number of parameters can mitigate this risk, leading to a more robust model that generalizes better to new data. One effective strategy to achieve this is by reducing the size of the latent space in autoencoders. The latent space serves as a compressed representation of the input data; by limiting its dimensionality, we encourage the model to focus on capturing the most essential features [15].

To explore the impact of latent space dimensionality on model performance, I trained three autoencoders, each using the loss function that previously yielded the best classification results (**Loss_MSE_DZL**). The models were configured with different latent space sizes (**Dim**): 1024, 512, and 256 dimensions. The objective was to assess whether reducing the latent space could maintain or even improve classification accuracy while simultaneously reducing the model's complexity and susceptibility to overfitting.

Tableau 4.2 – Impact of reducing latent space.

Dim	Parameters_Trained	Class_Accuracy (%)	NMSE	G_Accuracy (%)	G_Loss
1024	126,664	66.13	0.1119	65.07	1.08
512	67,784	67.47	0.1176	65.32	1.13
256	38,334	65.28	0.1242	64.38	1.18

Reducing the latent space from 1024 to 256 dimensions simplifies the model (as shown by the reduced number of parameters), but is accompanied by slight compromises in classification accuracy and reconstruction quality. The 512-dimensional latent space seems to offer the best compromise, balancing model complexity and performance.

2.2 Effects of reducing labelled data for classification

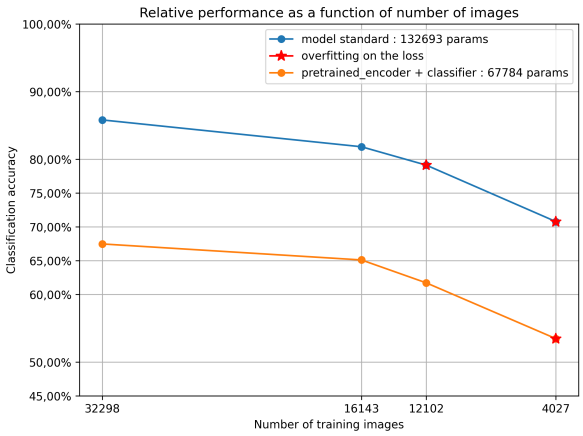


Figure 4.4 – Relative performance as a function of number of images.

Models name	Nb_images	Class _accuracy	Parameters_Trained
Model_1_Full	32298 (100%)	85.81	132693
Model_1_Half	16143 (50%)	81.84	132693
Model_1_3_8	12102 (37.5%)	79.12	132693
Model_1_1_8	4027 (25%)	70.77	132693
Model_select_full	32298 (100%)	67.47	67784
Model_select_half	16143 (50%)	65.1	67784
Model_select_3_8	12102 (37.5%)	61.71	67784
Model_select_1_8	4027 (25%)	53.46	67784

Figure 4.5 – Effects of reducing labelled data

In this part, I evaluated my model to answer to the question of my intership about to build a model able to produce a good classication result with a few data labelled. Since the model did not perform as well as the standard model, we decided to evaluate its robustness in the context of reduced labeled data 4.4, 4.5. Specifically, we assessed the loss in classification accuracy and the potential for overfitting. To achieve this, I trained both model types on the classification task using 100% (32,298 images), 50%, 37.5%, and 25% of the labeled images.

Results and interpretations

Analysis of the results shows that the **Model_standard**, although effective with the full dataset, begins to show signs of overlearning as soon as the dataset is reduced to 37.5%. This overlearning results in a sharp increase in loss during training, indicating that the model overfits to the specific data in the training set, compromising its ability to generalise to new data.

In contrast, the pre-training model demonstrated greater robustness. Even with only 37.5% of the training data, it maintained a reasonable drop in accuracy without showing signs of overlearning. This robustness is explained by the use of a pre-trained encoder that efficiently captures the relevant features of the images, even when the data is limited.

3 Limits of the study for classification task

This section evaluates the impact of different model architectures on classification performance. We compare the standard model (where all parameters are trained for classification) with various Autoencoder Feature Extraction approaches for Classification. Unlike the previous section, here the encoder is designed as a feature extractor, mirroring the architecture of the standard model. This approach allows us to assess the impact of using a frozen encoder for feature extraction on classification performance.

3.1 Model Descriptions

1. **Standard Model (Fully Trained):** All parameters are trained for the classification task. (Model_Standard)

2. **Standard Model with Additional Dense Layer:** The standard model with an added dense layer of 115 neurons. This model exhibited overfitting.(Model_Standard_115)

3. **Frozen Autoencoder Feature Extraction:** Only the last dense layers are trained for classification, while the encoder remains frozen. (Model_1_AEFC)

4. **Frozen Autoencoder with Additional Dense Layer:** Similar to model 3, but with an added dense layer of 115 neurons. (Model_1_AEFC_115)

5. **Partially Unfrozen Autoencoder:** Similar to model 3, but with the last convolutional layer of the feature extractor also trained along with the dense layers. (Model_1_AEFC_Conv)

3.2 Results and interpretation

Model	Class_Accuracy	Loss	Parameters_Trained	Overfitting	Dataset
Model_Standard	85.81%	0.59	132693	False	False
Model_Standard_115	85.68%	0.55	166280	True	Test
Model_1_AEFC	56.18%	1.36	34197	False	Test
Model_1_AEFC_115	68.76%	1.04	67784	False	Test
Model_1_AEFC_Conv	73.98%	0.93	108053	False	Test

Tableau 4.3 – Feature extraction **model standard** used as **encoder**

The analysis of the table comparing the different classification models reveals several key insights:

- **Performance of the Model Standard**

1. (**Model_Standard**)
   The standard model achieves the highest classification accuracy at 85.81%. This suggests that fully training the model for the specific classification task is highly effective.
2. **Impact of Adding Dense Layers** Adding a dense layer with 115 neurons to the standard model (**Model_Standard_115**) does not significantly improve performance (85.68% vs. 85.81%) with overfitting presence. This indicates that the standard model already possesses sufficient capacity for the classification task, Complexity is increased by adding a dense layer of additional parameters to adjust, which leads to model overfitting, in line with the fact that an overly complex model is more susceptible to overfitting.

- **Performance Autoencoder Feature Extraction for classification**

1. **Model_1_AEFC**: Models using a pretrained encoder show a significant drop in performance compared to the standard model. The **Model_1_AEFC** model only achieves 56.18% accuracy, suggesting that using a frozen pretrained encoder limits the model's adaptability to the specific classification task. which in my opinion would mean that feature extraction is not complete when you want to use the extractor for classification, remembering here that the extractor is an encoder trained to compress images and not for classification task.
2. **Improvement with Increased Complexity**: Adding a dense layer with 115 neurons to the pretrained model (**Model_1_AEFC_115**) significantly improves performance, increasing accuracy from 56.18% to 68.76%. This indicates that increasing the capacity of the classification part can partially compensate for the limitations of a frozen pretrained encoder.
3. **Impact of Partial Unfreezing of the Encoder**: The model **Model_1_AEFC_Conv**, which allows the last convolutional layer of the encoder to be trained, shows a substantial improvement in performance (73.98%). This suggests that a certain degree of adaptation of the encoder to the specific task is beneficial.

3.3 Implications

- Using a frozen pretrained encoder for feature extraction may limit classification performance, likely due to a loss of task-specific information.

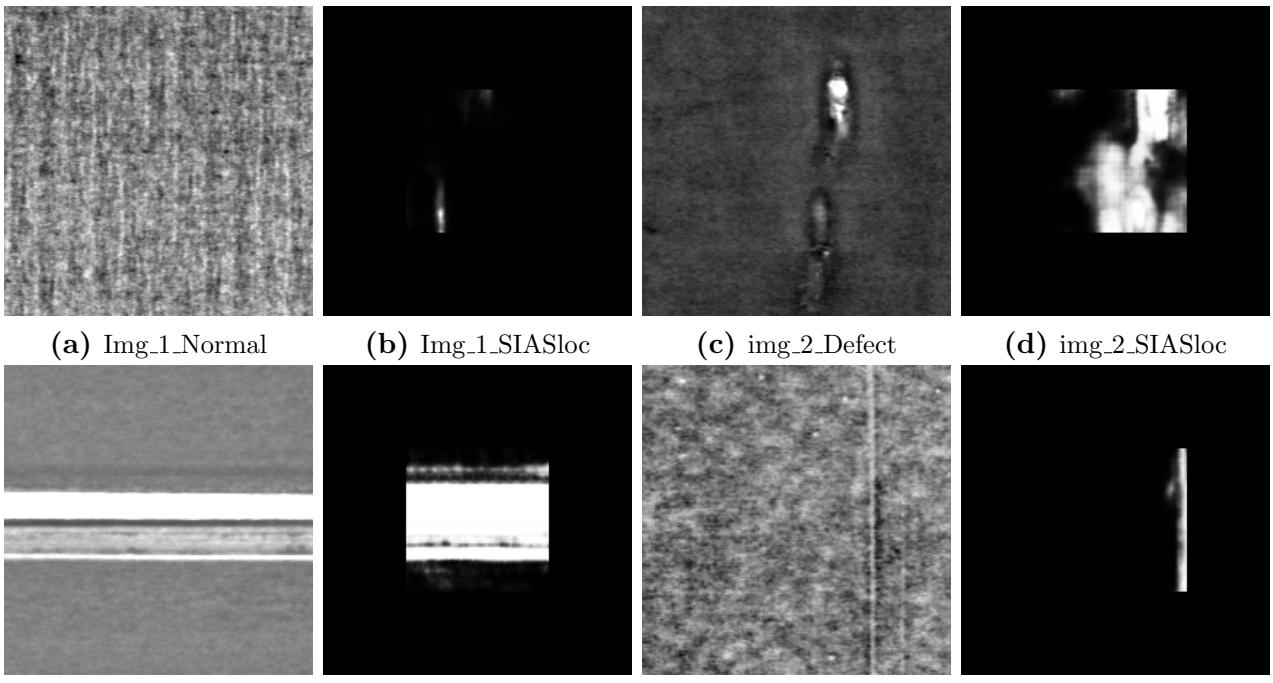
- Increasing the complexity of the classification part (by adding dense layers) can partially compensate for the limitations of a frozen encoder.
- Allowing partial adaptation of the encoder (by unfreezing certain layers) can significantly improve performance, suggesting a trade-off between transfer learning and task-specific adaptation.

These results highlight the importance of balancing the use of pretrained features with the need for adaptation to the specifics of the classification task. They also suggest that a two-stage approach (pretraining the encoder followed by classification) may require careful design of the classification architecture or a fine-tuning strategy to achieve optimal performance.

4 Implications and Future Work

4.1 Using Images of localisation model for loss function

The localization model, developed by Clecim within the **SIAS Deep Learning** framework, is a deep learning model designed specifically for defect localization. This model has demonstrated high efficiency in accurately identifying defects. It works by generating a mask for the central region of an image, focusing on a 98×98 area within a 192×192 image.



(a) Img_1_Normal **(b)** Img_1_SIASloc **(c)** img_2_Defect **(d)** img_2_SIASloc

(e) img_3_Defect **(f)** img_3_SIASloc **(g)** img_4_Defect **(h)** img_4_SIASloc

Figure 4.6 – Localisation examples

- **Advantages**

  1. It finds defects when they exist in the central zone of 98×98
  2. Instead of providing a binary mask like the one in **Defect Zone Localization (DZL)** 1.2.2, this model generates a mask with values ranging between 0 and 1.

- **Drawbacks**

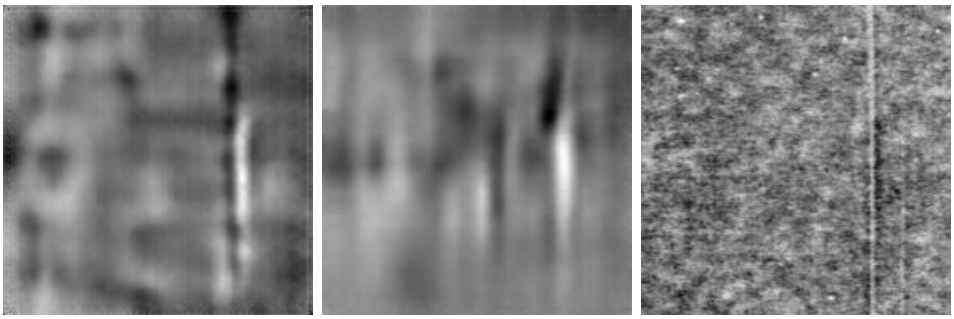
  1. The input shape must be exactly $(192, 192, 1)$ to be compatible with the pretrained localization model.
  2. The model only detects defects within the central zone. If a defect is located outside this detection area, it will not be identified by the model, as illustrated in image 4 4.6g and 4.6h.

4.1.1 Results and interpretation

The Autoencoder was trained by using the following loss function:

$$\mathcal{L}_6(X, Y) = \mathcal{L}_1(X, Y) + \frac{\lambda}{M} \sum_{i=1}^{M} \|\hat{X}_i \cdot (X_i - Y_i)\|^2$$

Where $\lambda \in \mathbb{R}^+$ ($\lambda = 20$ in this example) and $\hat{X}_i$ is the localization mask of image X_i. I called this loss : **Loss_MSE_SIASloc** (SIAS Localisation)



(a) Loss_MSE_DZL **(b)** **Loss_MSE_SIASloc** **(c)** Original

Tableau 4.4 – Performance Metrics for Different Models using SIASloc and DZL

Model Name	Dim	Class_Accuracy (%)	NMSE	G_Accuracy (%)	G_Loss	Parameters_Trained
Model_1_MSE_DZL	512	67.47	0.1176	65.32	1.13	67784
Model_2_SIAS_Loc	512	70.15	0.1357	67.66	1.0	67784

The mask generated by the SIAS localization method produces image reconstructions that closely resemble those generated by **DZL**. This similarity is because **DZL** is a purely statistical localization method, driven by a threshold that may not always be optimal. In contrast, **SIASloc** utilizes a neural network, which takes the context of the image into account, allowing it to more accurately detect whether the central zone of the image contains anomalies. Although **SIASloc** delivers very promising results, extending its localization region to encompass the entire image could significantly improve the outcomes. In some defects, like in 4.7c, **SIASloc** only locates part of the defect 4.6h, but still manages to classify it correctly. This raises the question of how effective it could be if it were able to generate complete masks for the entire image.

4.2 Improve the architecture of the autoencoder

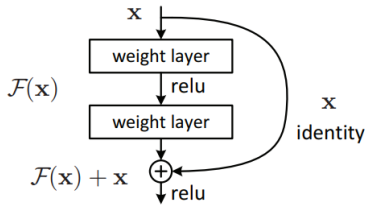
The literature highlights the **ResNet** (Residual Network) architecture, a revolutionary deep learning model introduced by He et al. [13]. ResNet effectively addresses the challenges of training very deep neural networks, particularly the vanishing gradient problem, by incorporating residual blocks with shortcut connections. These shortcuts allow the network to bypass certain layers, facilitating the training of models with hundreds of layers and significantly improving performance.

4.2.1 Architecture of ResNet

The core component of ResNet is the **residual block**, where the output is the sum of the original input and the function applied to it:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

where:



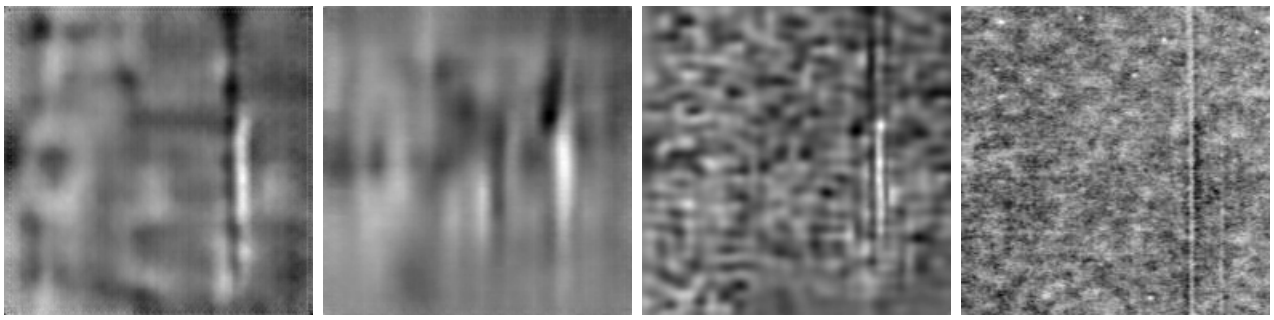
- $\mathbf{x}$ is the input to the residual block.
- $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the convolutional operations with weight parameters $\{W_i\}$ applied to $\mathbf{x}$.
- $\mathbf{y}$ is the output of the residual block.

Figure 4.8 – Residual Block Architecture [1]

I utilized this residual block to construct an Autoencoder, which I have named ResAE (Residual Autoencoder). Here a latent space has represented by a feature maps 1.3 (*shape* =

$6, 6, 32)$).

Results and Interpretation



(a) Loss_MSE_DZL **(b)** **Loss_MSE_SIASloc** **(c)** **ResAE_Loss_MSE** **(d)** Original

Tableau 4.5 – Performance Metric the Models ResAE using MSE

Model Name	Dim	Class_Accuracy (%)	NMSE	G_Accuracy (%)	G_Loss
Model_1_MSE_DZL	512	67.47	0.1176	65.32	1.13
Model_2_SIAS_Loc	512	60.95	0.1357	61.71	1.2
Res_AE_MSE	(6,6,32)	53.41	0.0707	52.91	1.45

The training of **Autoencoders (AEs)** yields impressive results for reconstruction tasks. However, when it comes to classification, utilizing only the encoded representations with a simple classifier often produces suboptimal outcomes. Although it gives slightly better results for the same loss function (MSE) than our previous architectures

These poor classification results, despite nearly identical reconstructed images, confirm our hypothesis regarding the use of contractive methods for the Autoencoder (AE). Furthermore, while the vertical scratch representing the defect is accurately reconstructed, the noise surrounding the defect is also reproduced, and in some cases amplified. This added noise could potentially contaminate the latent space, hindering the effective classification of the defect.

4.3 Artificial defects

I didn't have enough time to explore this part. Adding artificial defects to deep learning models during training has become an important technique for improving model robustness and generalization. This document summarizes the key effects and results of this approach.

- Advantages

1. **Improved Robustness:** Models trained with artificial faults show greater resilience to disturbances and noisy data, and have already demonstrated better generalisation on the **SIAS** data.
2. **Enhanced Data Augmentation:** The addition of artificial defects serves as a form of data augmentation, helping to diversify the training set and reduce overfitting [14, 23].
3. **Better Anomaly Detection:** Models learn to recognize a broader spectrum of possible defects, potentially improving performance in anomaly detection tasks [9].
4. **Learning of Invariant Features:** Models tend to learn more robust and invariant features when trained with artificial defects [22].

- **Drawbacks**

1. **Potential for Overfitting to Artificial Defects:** If not carefully designed, models might overfit to the specific types of artificial defects used in training.
2. **Complexity in Defect Design:** Determining the appropriate types and levels of artificial defects to introduce can be challenging and may require domain expertise.

CONCLUSION

My internship at CLECIM provided a valuable opportunity to explore and address the challenges associated with defect detection in steel production using deep learning.

Throughout this project, we have systematically investigated different approaches to optimising autoencoders for defect detection and classification in steel surface images. We first explored several reconstruction loss functions, including full reference loss and contractive loss functions, to improve the quality of image reconstruction. Our analysis revealed that while global reference methods such as MSE and SSIM give good results for global reconstruction, they do not accurately detect defects, as they focus on the average error over the whole image but also on areas that add noise to the defect area we wish to detect, hence the use of contractive methods for AE training.

We also examined the effects of reducing the latent space of the autoencoder and limiting the number of labelled images used for classification. These studies showed that reducing the latent space led to a slight loss in reconstruction quality, but had a negligible impact on classification, for a reasonable reduction in latent space (512 for 193 *193 images, for example). This reduces the number of parameters in the AE, but more importantly in the classifier, making it more robust to the reduction in labelled data, unlike the **Model_strandard**, which shows overfitting earlier.

In the final stages of this work, we addressed the limitations of our study by proposing future improvements, including improving the architecture of the autoencoder and exploiting synthetic data to increase fault detection. These ideas form the basis of future work, which aims to refine defect detection models and create more robust and data-efficient solutions for industrial quality control systems.

Overall, this experience not only contributed to CLECIM's innovation in the steel industry but also provided me with deeper insights into the practical applications of deep learning in industrial settings. The skills and knowledge gained during this internship will undoubtedly be instrumental in my future career endeavors in the field of artificial intelligence and machine learning.

ANNEX: REUSABLE TOOLS OVERVIEW

Throughout my internship, several Python tools and scripts were developed to facilitate data preprocessing, model training, and evaluation. These tools are stored in the `Tools_Models` folder and can be easily reused or adapted for future projects. Below is an overview of the key tools:

1. Data Preprocessing Tools

Several scripts for data preparation are included in `Preprocessing.py` and `ModelHandler_Mrl.py`. These handle tasks such as duplicate image removal, dataset splitting, sample creation, dataset for train, for Test and for Validation adding the data augmentation.

2. Autoencoder Training

Custom autoencoder models were built with flexible input configurations. The training can be controlled via the `1_Autoencoder.ipynb` notebook.

3. Classifier Training

Both standard classification models and `pretrained_encoder + classifier` setups are supported. Notebooks are named `1_Classifier_Model.ipynb` and allow for hyperparameter configuration and architecture modification.

4. Model Evaluation and Comparison

The evaluation of autoencoders and classifiers is performed via dedicated notebooks. Tools include:

- `1_Evaluate_Classif_report.ipynb`: Generates evaluation reports for classifiers.
- `1_Compare_Classifs_Models.ipynb`: Compares classification model performance.

5. Notebook Automation

To address GPU memory issues during long training sessions, the `run_notebooks` script was developed. This script automates the sequential execution of Jupyter notebooks, ensuring efficient GPU usage.

BIBLIOGRAPHY

[1] Residual Neural Network (ResNet).

[2] Alibi detect.

[3] Kamel IDRIS v Agathe, Baptiste et Yanis UGA/DAPI Thibaut. Les réseaux autoencodeurs (AE) Un exemple d'apprentissage non supervisé, .

[4] Kamel IDRIS v Agathe, Baptiste et Yanis UGA/DAPI Thibaut. Réseaux convolutifs (CNN), .

[5] Christopher M. Bishop. Pattern Recognition and Machine Learning, 2006.

[6] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A Theoretical Analysis of Feature Pooling in Visual Recognition, 2010.

[7] Jason Brownlee. Deep Learning for Computer Vision, 2019.

[8] Jason Brownlee. Autoencoder Feature Extraction for Classification, December 2020.

[9] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness, 2018.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning, 2016.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning, 2016.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning, 2009.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty, 2019.

[15] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks, 2006.

[16] Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders, 2019.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. 2012.

[18] Jonathan Masci, Ueli Meier, Gabriel Fricout, and Jurgen Schmidhuber. Object Recognition with Multi-Scale Pyramidal Pooling Networks, 2012.

[19] Cordelia Schmid. Bag-of-features for image classification.

[20] Zianou Ahmed Seghir. *Évaluation de la qualité d'image*. Université de Mentouri – Constantine, 2012.

[21] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, Avril 2004.

[22] Dong Yin, Raphael Gontijo Lopes, Jon Shlens, Ekin D Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision, 2019.

[23] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017.