STEEL DEFECT DETECTION

WITH HIGH-FREQUENCY CAMERA IMAGES

By

Md Abir Hasan (170201063)

Mihir Roy (170201028)

B. Sc. Engineering
In
Computer Science and Engineering

Department of Computer Science and Engineering
Faculty of Electrical and Computer Engineering
Bangladesh Army University of Science and Technology

FEBRUARY 2022

STEEL DEFECT DETECTION WITH HIGH-FREQUENCY CAMERA IMAGES

This thesis is submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering.

By

Md Abir Hasan (170201063)

Mihir Roy (170201028)

Supervised by
Taher Muhammad Mahdee
Assistant Professor
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

Department of Computer Science & Engineering
Bangladesh Army University of Science & Technology
Saidpur Cantonment, Nilphamari, Bangladesh.

The thesis titled **"Steel defect detection with high-frequency camera images"** submitted by Md Abir Hasan, Roll No. 170201063 & Mihir Roy, Roll No. 170201028, Session 2017-2018 has been accepted as satisfactory in fulfillment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering (CSE) as B.Sc. Engineering to be awarded by the Bangladesh Army University of Science & Technology (BAUST).

Board of Examiners

1. _____ Chairman

Dr. Md. Shamim Akhter
Professor and Head of the Department,
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

2. _____ Supervisor

Taher Muhammad Mahdee
Assistant Professor,
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

3. _____ Member

Md. Mamun Hossain
Assistant Professor,
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

4. _____ Member

Md. Moazzem Hossain
Assistant Professor,
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

5. _____ Member

Md. Sydur Rahman
Lecturer,
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

6. _____ Member

Urmi Saha
Lecturer,
Department of Computer Science & Engineering (CSE)
Bangladesh Army University of Science & Technology (BAUST)

CANDIDATE'S DECLARATION

It is hereby declared that the content of this thesis is original and any part of it has not been submitted elsewhere for the award of any degree or diploma.

Signature of the Candidate

Date:

Signature of the Candidate

Date:

DEDICATION

We dedicate this book to our beloved parents, our respected teachers, and everyone we love.

Acknowledgment

First of all, we are grateful to almighty Allah who enabled us to complete this thesis successfully. Thereafter our sincerest thanks and gratitude to our honorable thesis supervisor Taher Muhammad Mahdee, Assistant Professor of Computer Science & Engineering, Md Mamun Hossain, Assistant Professor of Computer Science & Engineering, and Dr. Md. Shamim Akhter Professor & Head of Department of Computer Science & Engineering, Bangladesh Army University of Science & Technology, for their valuable suggestions, positive pieces of advice, encouragement, and sincere guidance throughout our thesis work. We also convey our special thanks and gratitude to all of the respected teachers of the department. We would like to thank all of our friends and the staff of the department for their valuable suggestions and assistance. Finally, we would like to thank our parents for their steady love and support during our study period.

Abstract

We tried to detect steel defect with high-frequency camera images and this detection is done by a trained ML and Deep Learning process. We focused on the problem of steel defect detection, where we are given images of steel sheets taken by high-frequency cameras and aim to mark pixel-wise defected areas. From previous work, by other researchers, we have found that they researched on image segmentation. We used Severstal steel images (dataset). Russian Severstal Company Severstal mainly operates in the steel and mining industry. The company conducted a Kaggle competition by providing the data of defective steel images. We found a CSV file that includes datasets and defects are classified by segmentation. We took a task to train and test those defects from that images and gave data with some different types of methods. We explored five methods to solve the task. Two deep learning (CNN, Xception), one clustering (KNN), and two classifications (SVM, Random forest) methods.

By Comparing to their performances we archived the highest accuracy of 89.80% with the deep learning method Xception. Among those five methods, we can say Xception can give us the highest number of defect detection with less errors. We showed the comparison of deep learning and machine learning outcomes in a single research of steel defect detection.

Keywords:

Random Forest, SVM (Support Vector Machine), KNN (K- Nearest Neighbor), CNN (Convolutional Neural Network), Xception, Machine Learning (ML), Deep Learning (DL)

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Our technology is growing up and up every day. Nowadays using deep learning and machine learning methods we have improved computer vision ability. Now we can find tiny details (like steel surface defects) about something using this computer vision. This chapter will discuss the introduction of steel defect, how it occurs, defect detection, background and present state, problem statement, objectives, scopes and limitations, organization of the report, and the summary.

1.2 Steel Defect and Detection

Steel is the most important engineering and construction material nowadays. It is used in every step of our lives. After materializing steel from elements, the production process of steel sheets is especially delicate. This process starts with heating steel, then rolling to drying and cutting. Several machines touch flat steel by the time it's ready to ship. Steel defect occurs when those steel sheets are in the production process. Before providing these steel sheets outside the industry, it needs to undergo a careful inspection to avoid defects, and thus localizing and classifying surface defects is crucial. Hence, automating the inspection process would accelerate steel sheet production. By using deep learning and machine learning models with steel sheets images we can detect those defects. This thesis is targeted at finding an efficient model of detecting defects on steel sheets with images from high-frequency cameras. The future goal is to ensure all industries provide defect-less steel sheets. Otherwise, items or other structures built of this steel might be put at risk.

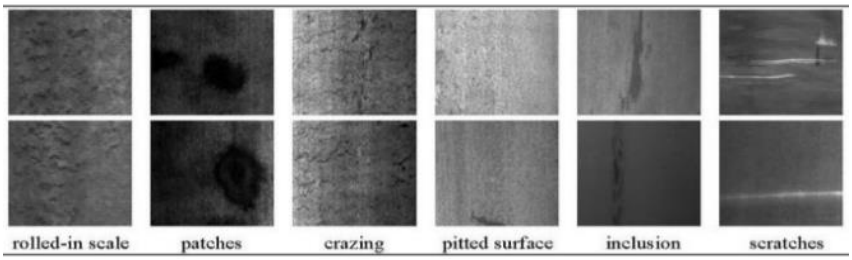

Figure 1.1: Several types of defects.

1.3 Background and Present State

Many researchers are trying to improve this detection process. So they are trying to evaluate different models or trying to experiment on existing models to find better processes.
Because we are using images to detect a defect, image segmentation is needed for this type of operation.
J. Long, E. Shelhamer, and T. Darrell researched “Fully convolutional networks for semantic segmentation”. Semantic segmentation could be considered a per-pixel classification problem. The most popular CNN-based method of semantic segmentation is the Fully Convolutional Network (FCN) [1]. This helps in improving computer vision. Because FCNs for semantic segmentation can improve accuracy by transferring pre-trained classifier weights, combining different layer representations, and learning end-to-end on whole images.

Weidong Zhao, Feng Chen, Hancheng Huang, Dan Li, Wei Cheng, did research on "A New Steel Defect Detection Algorithm Based on Deep Learning" [14]. They used Faster R-CNN and tried to find dice efficient for this.
Qianlai Sun (孙前来), Yin Wang (王银) and Zhiyi Sun (孙志毅) did research on ”Rapid surface defect detection based on singular value decomposition using steel strips as an example”[15]. They presented an improved SVD method that is more conducive to real-time defect detection.

Kun Qian did research on “Automated Detection of Steel Defects via Machine Learning based on Real-Time Semantic Segmentation” [16]. He applied a series of machine learning algorithms of real-time semantic segmentation, utilizing neural networks with encoder-decoder architectures based on U-net and feature pyramid network (FPN).

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab researched “Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs” [2]. This is a semantic segmentation task with help of atrous convolution which is a powerful tool in dense prediction tasks. They used Atrous Spatial Pyramid Pooling (ASPP) to segment image objects at multiple scales. They improved the localization of the object boundary by combining the final DCNN layer with a fully connected Conditional Random Field (CRF).

H. Noh, S. Hong, and B. Han researched “Learning deconvolution network for semantic segmentation.”[3]. they proposed a novel segmentation algorithm by learning a deconvolution network based on the VGG 16-layer net.
R. Girshick wrote on “Fast R-CNN” [4]. He describes everything about Fast R-CNN and its speed.
T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie researched on “Feature pyramid networks for object detection.”[5]. they exploit the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct the feature pyramids with marginal extra cost. A top-down architecture with lateral connections is developed for building high-level

semantic feature maps at all scales.

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick wrote on “Microsoft coco: Common objects in context.”[6]. they present a new dataset to advance the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding.

M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele researched “The cityscapes dataset for semantic urban scene understanding.”[7]. they introduce Cityscapes, a benchmark suite and large-scale dataset to train and test approaches for pixel-level and instance-level semantic labeling.

Z. Zhang, A. G. Schwing, S. Fidler, and R. Urtasun researched “Monocular object instance segmentation and depth ordering with CNN.”[8]. they tackled the problem of instance-level segmentation and depth ordering from a single monocular image. Towards this goal, they took advantage of convolutional neural nets and trained them to directly predict instance-level segmentation where the instance ID encodes the depth ordering within image patches.

Z. Zhang, S. Fidler, and R. Urtasun researched “instance-level segmentation for autonomous driving with deep densely connected MRFs.”[9]. They formulate the global labeling problem with a novel densely connected Markov random field and show how to encode various intuitive potentials in a way that is amenable to efficient mean-field inference.

B. Romera-Paredes and P. H. S. Torr researched “Recurrent instance segmentation.”[10]. they propose a new instance segmentation paradigm consisting of an end-to-end method that learns how to segment instances sequentially.

L. Shen, Z. Lin, and Q. Huang researched “Relay backpropagation for effective learning of deep convolutional neural networks.”[11]. they considered the issue from an information-theoretical perspective and proposed a novel method Relay Backpropagation, which encourages the propagation of effective information through the network in the training stage.

F. Chollet. Wrote about “Xception: Deep learning with depthwise separable convolutions.”[12]. He presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution).

Matthew Browne, Saeed Shiry Ghidary wrote about “Convolutional Neural Networks for Image Processing: An Application in Robot Vision” [17]. In this paper, they briefly described convolutional neural networks.

Zhu Fan, Jia-Kun Xie, Zhong-yu Wang, Pei-Chen Liu researched “Image Classification Method Based on Improved KNN Algorithm” [18]. KNN costs too much time when classifying images, which is not qualified to actual application scenes. They did an improved algorithm proposed in this paper. The test time has been greatly shortened and the efficiency of the KNN algorithm is improved by increasing the screening of the data set.

Baoxun Xu, Yunming Ye, Lei Nie researched “An improved random forest classifier for image classification” [19]. They proposed an improved random forest algorithm for image classification. This algorithm is designed for analyzing very high dimensional data with multiple classes whose well-known representative data is image data.

1.4 Problem Statement

Steel defect detection with high-frequency camera images based on an image processing system. Image processing needs high memory storage and processing power for a large number of datasets. We faced this memory storage issue during our experiment. We used 20GB ram but this isn't enough for the total dataset. For this reason, we used half of the dataset and continued our experiment. Basic algorithms are used in this experiment, we noticed some of them, and especially KNN and Xception take a lot of time to run. To make automated detection these operations need to be done faster. Due to the imbalance between different kinds of labels, more data augmentation processes such as rotation and scaling on the images should be performed on the rare defect classes.

1.5 Objectives

The objectives of this thesis are:

i) To find an efficient model between classifications, clustering, and deep learning methods.
ii) To achieve better accuracy of detecting defects using different types of methods.
iii) To learn how those algorithms are working.
iv) To show the comparison of deep learning and machine learning outcomes in a single research.

1.6 Scopes and limitations

Deep learning and Machine learning applications can potentially improve the accuracy of defect detection. It is under an image processing system. Because we use images of steel to detect the defect. Using neural networks that can learn from data without supervision. Deep learning applications can detect, recognize and analyze crucial defects from images. We take the operation of other machine learning algorithms too so that you can understand the learning between

classification, clustering, and deep learning methods. You can gain knowledge about using different methods (SVM, Random forest, KNN, CNN, Xception) in a single image processing task.

In the problem statement, we said that image processing needs high memory storage and processing power for a large number of datasets. Normally you can face this problem if you are using a low-end device. Collecting new datasets is a hard task. These are the limitations that we faced.

1.7 Organization of the Report

This report is organized into six chapters. After this chapter of the introduction, In, Chapter 2 reviews the "Literature Review" of our study including some other related research on steel defect detection. Chapter 3 discusses the proposed methodology for finding a better model for steel defect detection. Chapter 4 describes the implementation of the methodology with dataset collection and preprocessing. Chapter 5 presents the result of our implementation of different types of deep and machine learning models. Finally, Chapter 6 summarizes the conclusion of our work and our future work of study.

1.8 Summary

In this chapter, we provide an introductory idea and importance of steel defect detection. Background and present state about this image processing task. Scope and limitations that are helpful to others. The objectives of our study all are presented in this chapter.

Chapter 2

Literature Review

2.1 Overview

A literature review is a summary of past research on a particular subject. In this chapter, we will discuss the related works that were previously researched on steel defect detection using deep and machine learning methods. It is assumed that by mentioning a previous work in the field of study, we have read, evaluated, and assimilated that work into the work at hand. Also will discuss the significance of the study, stakeholders of the thesis, and open issues with a final summary of this chapter.

2.2 Related Works

For steel defect detection, we need the input and that is an image of a steel sheet. For input, the output is a same-size segmented image with dense defect area marks. So the output is a pixel-wise label for the given input image and it is an image processing segmentation task. Semantic and Instance segmentation are two different sub-task. In the past, computer vision was not as powerful as the present. So, it is a very hard task to segment objects from images. But after the invention of CNN, this problem of segmentation has turned into an easy task.

Semantic segmentation is considered a per-pixel classification problem. The popular CNN-based The method of semantic segmentation is the Fully Convolutional Network (FCN) [1], which converts fully connected layers into 1x1 convolutional layers and achieves end-to-end per-pixel total prediction. FCNs for semantic segmentation can improve accuracy by transferring pre-trained classifier weights, combining different layer representations, and learning end-to-end on whole images. The traditional FCN method causes the problem of resolution loss. Two main methods are proposed to solve this problem. The first method [2] uses “atrous convolution”, which enlarges the feature map with the help of linear interpolation. The second utilized deconvolution [10] to learn the upsampling process.

Compared to semantic segmentation, the instance segmentation's goal is to predict class labels and also pixel-wise instance masks to localize varying numbers of instances presented in images. There are mainly two methods to solve instance segmentation problems: i) proposal-based methods and ii) segmentation-based methods. Proposal-based methods are related to object detection (Fast R-CNN, Feature pyramid networks) [4, 5]. Mask R-CNN is a widely-used method in this steam,

which proposes to add a fully convolutional network branch based on (Feature pyramid networks) [5]. They achieve great performance on (Common objects in context, the cityscapes dataset for semantic urban scene understanding) [6, 7]. The other is segmentation-based methods, which use the output of semantic segmentation as input and gain instance-aware segmentation results later. Among them, (Monocular object instance segmentation and depth ordering with CNN, Instance-level segmentation for autonomous driving with deep densely connected MRFs) [8, 9] proposed to use a graphical model to guess the order of instances and (Recurrent instance segmentation, Relay backpropagation for effective learning of deep convolutional neural networks) [10, 11] take advantage of RNN to obtain one instance in each time step.

2.3 Significance of the Study

Steel defect isn't some avoidable problem. The small defect can cause major damage. Steel is an important engineering and construction material. Building construction, car, boat, ship almost every transport vehicle, necessary products all are using steel. If those are made with defective steel sheets, a large amount of risk will face human beings like sinking ships, product blasts, building crushes, etc. To lower those risks we researched this topic. We used deep and machine learning methods to detect those defects efficiently and accurately which can be used in automated defect detection software in the future. The steel industry can use that and can provide pure and defect-less steel sheets.

2.4 About Stakeholders of the Thesis

To lower those risks we researched this topic. We used deep and machine learning methods to detect those defects efficiently and accurately which can be used in automated defect detection software in the future. This research can help the steel industry to provide pure and defect-less steel sheets. We used Severstal steel images (dataset). The Russian Severstal company mainly operates in the steel and mining industry. The company conducted a Kaggle competition by providing the data of defective steel images. This research can provide information to them with a defect detection process.

2.5 Open Issues

We found some open issues after analyzing the previous works done by other researchers. Most of their work is related to the image segmentation part of image processing and improving computer vision. Some researchers tried to find accuracy with different models. But we didn't find comparison type research on steel defect detection. So we tried to show the comparison of deep learning and machine learning outcomes in a single research.

2.6 Summary

Most of the part of this chapter focused on previous research and its outcomes. First, we provided information about related works and studies of those authors. Then we discussed the stakeholders of the study and some open issues that are needed to solve.

Chapter 3

Methodology

3.1 Overview

We will discuss the proposed methodology for steel defect detection with the use of high-frequency camera images. The deep and machine learning approaches we used in this research and the theories or principles behind them are going to describe.
In this chapter, we are going to discuss Dataset collection, Pre-processing of data, Feature extraction (Train and Test splitting), Machine and Deep learning model approach.

3.2 Proposed Methodology

In our proposed methodology, we suggested a total of five approaches for steel defect detection model creation and evaluation. The flowchart for the proposed methodology is given below:

3.2.1. Flowchart

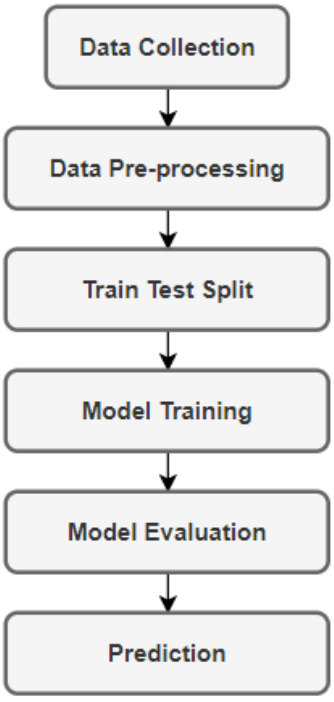

Figure 3.1: A basic overall methodology diagram for steel defect detection.

Described flow chart for deep learning and machine learning approach given below:

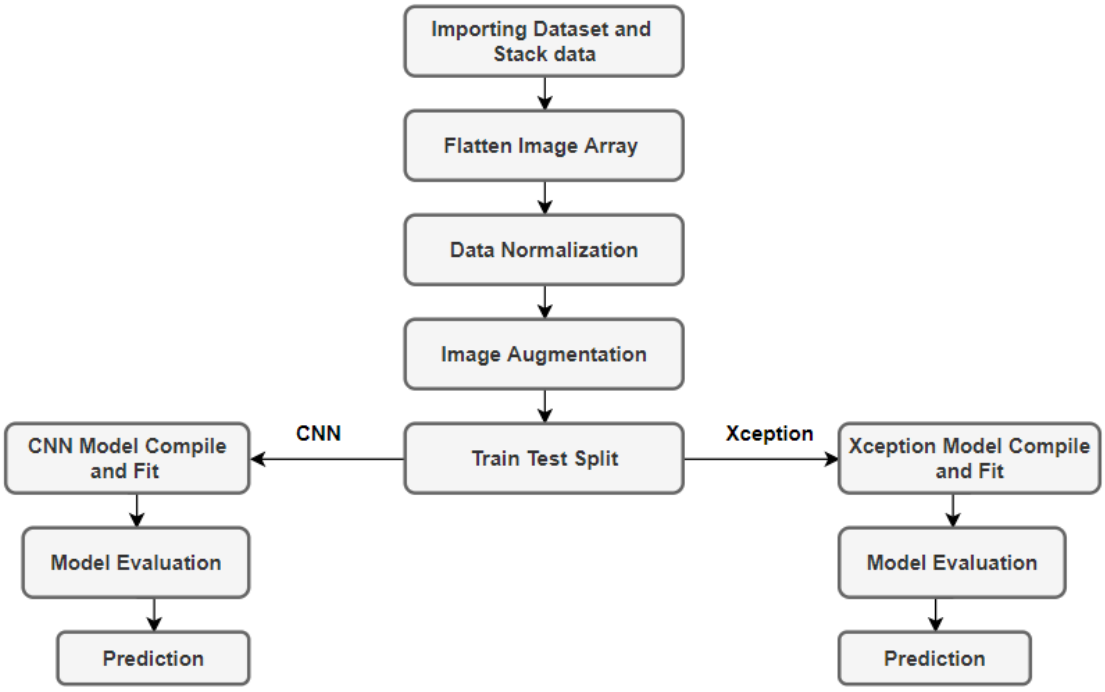

Figure 3.2: Diagram for Deep learning methods.

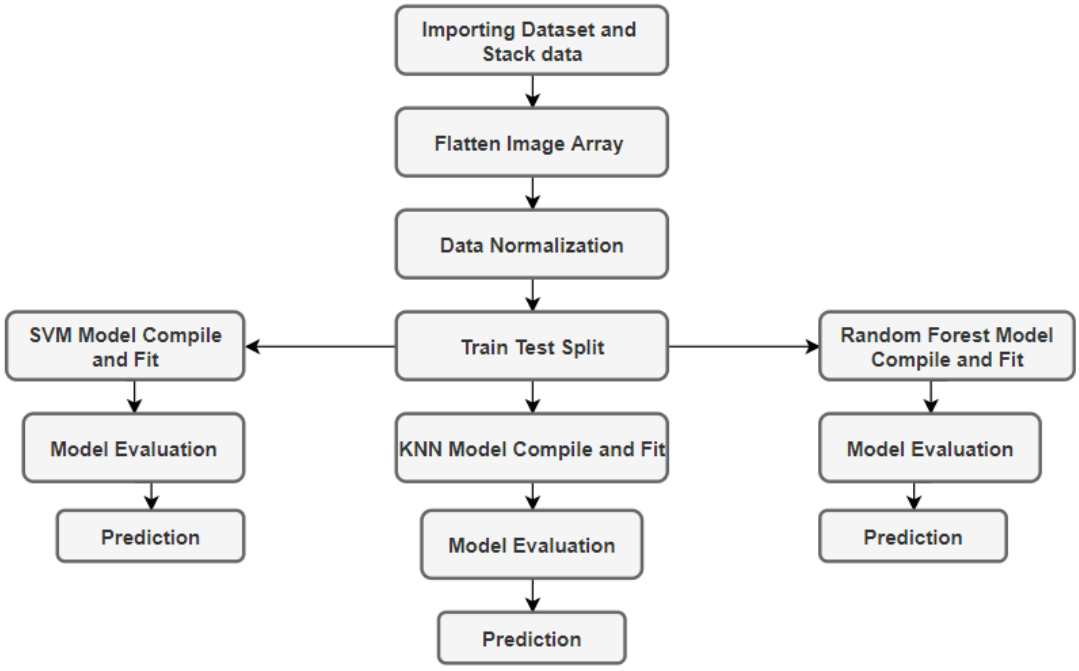

Figure 3.3: Diagram for Machine learning methods.

3.3 Data Collection

A specific dataset is a collection of specific data in which data is arranged in some order. A dataset can contain any data from a series of an array to a database table but if that dataset is specified for a single topic then data are not different types. A tabular dataset is most used nowadays because it can be understood as a database table or matrix, where each column corresponds to a particular variable, and each row corresponds to the fields of the dataset." Comma Separated File," or CSV, is the most commonly used file type for tabular datasets. Collecting and preparing the dataset is one of the hardest parts while creating an ML/AI project. Because new data is not available to collect on your own. There are a lot of datasets available nowadays which are freely available for the public to work on. Some popular sources are:

- Kaggle Datasets
- UCI Machine Learning Repository
- Datasets via AWS
- Google's Dataset Search Engine
- Microsoft Datasets
- Government Datasets

3.3.1 Dataset and Features

We used Severstal steel images (dataset). The Russian Severstal company mainly operates in the steel and mining industry [12]. The company conducted a Kaggle competition by providing the data of defective steel images. So basically we collected data from Kaggle. The dataset we are using is one captured with high-frequency images of steel sheets that are shown in figure 3.4. And the CSV file we are using is shown in Appendix A.1.

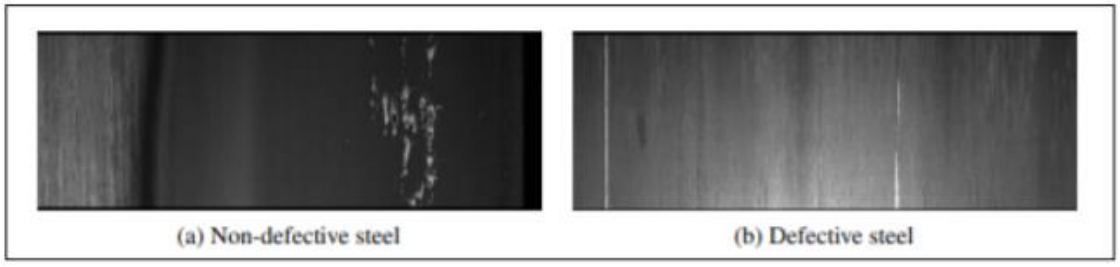

Figure 3.4: Examples of the defect and non-defect images in the SEVERSTAL dataset [12].

The images are 1600 × 256 × 1 in size and a total of 12568 images. 5902 images are with defects and 6666 images are without defects out of all the images. There are four types of label defects 1, 2, 3, and 4. A total of 897 images represent a class 1 defect, 247 images represent a class 2 defect, 5150 images represent a class 3 defect, and 801 images represent a class 4 defect. However, 6239 images have one defect class, 425 images have two defect classes, and only two images have three defect classes. And the important thing is we flatten all the images for use in this project. So there are a large number of features.

Due to memory space, we compute 3100 data to get a result. In figure 3.5, we can see a Total of 393 images represent a class 1 defect, 107 images represent a class 2 defect, 2258 images represent a class 3 defect, and 341 images represent a class 4 defect.

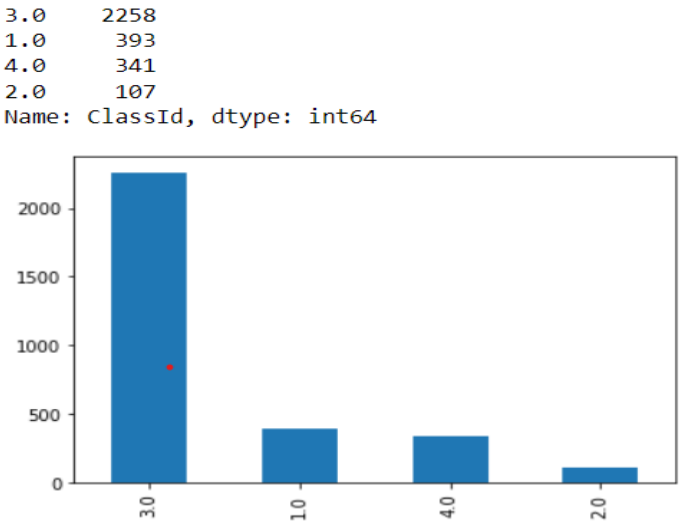

Figure 3.5: Count of defects with different classes.

3.4 Data Pre-processing

Data preprocessing is the procedure for preparing raw data for use in learning models. It's the initial and most important stage in building a learning model. It is not always the case that we come across clean and prepared data when working on a machine or deep learning project. And, before doing any data-related activity, it is necessary to clean the data and format it. As a result, we use a data preprocessing activity for this. Open or real-world data sometimes contains noise, missing values, and is in an inappropriate format that cannot be used directly in learning models. Data preparation is a necessary step for cleaning data and preparing it for a machine learning model, which also boosts the accuracy of the model. We also did pre-process on data. Now we are going to describe pre-processing steps for our work with code. Implementation of libraries that are used in this work will be discussed in chapter 4.

3.4.1 Importing Data

We did use Kaggle to gather data. We're going to use them now to find our goal. First, root the main folder, which contains the datasets. For images, a path is specified, and for datasets, a path is specified. Now stack all of the data images in the dataset and classid them into separate arrays. We can now access the data for our operation only by calling those array variables.

```
root = '.\\steel defect'
```

```
images = glob(os.path.join(root, "images", '*.jpg'))
```

```
dataset = pd.read_csv('steel defect\\train.csv')
```

```
X = []
y = []
for index in range(dataset.shape[0]):
    for i in images:
        image_name = os.path.basename(i)
        if image_name!=dataset['ImageId'].iloc[index]:
            continue
        with rio.open(i, 'r') as f:
            X.append(f.read(1))
            y.append(int(dataset['ClassId'].iloc[index]))
        break
```

```
X = np.stack(X)
y = np.array(y)
```

3.4.2 Flatten Image Array

Flattening is a technique for transforming multi-dimensional arrays into a single-dimensional array. It is commonly used in Deep Learning to feed 1-D array data to classification models. Multidimensional arrays consume more memory than 1-dimensional arrays, which is why we flatten the Image Array before processing the data to our model. Most of the time, we'll be working with a dataset with a big number of photos, therefore flattening aids in memory reduction and model training time reduction. We did this process too. And the CSV dataset file we are using also is a collection of flattening images.

```
n_samples = X.shape[0]
X_data = X.reshape((n_samples, -1))
```

3.4.3 Data Normalization

Normalization is the process of converting data into any range or simply onto the unit sphere. Its primary goal is to change data so that it is either dimensionless or has comparable distributions. We normalized data for the same reason.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_data = sc.fit_transform(X_data)
```

3.4.4 Image Augmentation

In this process, we specified the steel defect image size as 120x120. For the deep learning approach, this is a very helpful part because it can improve the ability of the fitting model. We applied this to resize the image input for the deep learning methods.

```
l1=[]
l2=[]
for ImageId,ClassId,EncodedPixels in tqdm(dataset.values):
    image=cv2.imread("steel defect\\images/{}".format(ImageId),cv2.IMREAD_COLOR)
    if image is not None:
        image=cv2.resize(image,dsize=(120,120))
        l1.append(image)
        l2.append(ClassId)
```

3.4.5 Feature Scaling and Label Encoding

For the deep learning method, we did extra feature scaling on images and labels encoded into categorical on classid. We scaled images intensity into the range [0-255].

```
X= np.array(l1)
X = X/255
y = encoder.fit_transform(l2)
y = to_categorical(y)
```

3.4.6 Data Train and Test Split Evaluation

A strategy for measuring the performance of a machine learning algorithm is the train-test split. It may be used for any supervised learning technique and can be utilized for classification or regression tasks. Taking a dataset and separating it into two subgroups is the technique. The training dataset is the initial subset, which is used to fit the model. The second subset is not used to train the model; instead, the dataset's input element is given to the model, which then makes

predictions and compares them to the predicted values. The test dataset is the name given to the second dataset. The goal is to evaluate the machine learning model's performance on new data that was not used to train the model. We set 80% data for trains and 20% data for tests.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
```

3.5 Method Description with Model Training

We used both machine and deep learning methods to find the accuracy of defect detection. Total five methods are applied here. Two deep learning (CNN, Xception) and three machine learning techniques including clustering (KNN) and classification (SVM, Random Forest) methods. Description of these models and what we used in our work are given below:

3.5.1 Deep Learning Models

- **Convolutional Neural Network (CNN)**
  The CNN [17] model is a type of neural network that allows us to extract higher representations for image data. Unlike traditional image recognition, which requires the user to define the image characteristics, CNN takes the image's raw pixel data, trains the model, and then extracts the features for improved classification. After importing the dataset we create two arrays. One of them contains a class and one image id. Then we put the imageid on the stack. Flattening image arrays then normalizes that data. Importing necessary library classes we check the dataset shape. We did image argumentation. Here we resize those images into 120x120 and put that with classid in a different array. Encode classid array by label encoder. Then splitting train and test data. Then make a model and compile it. Convolution layers, pooling layers, and fully linked layers are among the building components of the CNN architecture. A typical design comprises one or more completely linked layers followed by a stack of many convolution layers and a pooling layer.

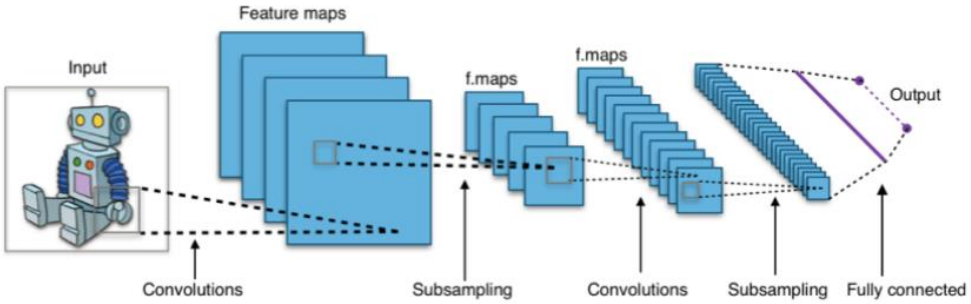

Figure 3.6: CNN Architecture.

The CNN model we constructed is given below and the summary of this model is shown in Appendix B.2

```
model=Sequential()
model.add(Conv2D(32,(3,3),input_shape=(120,120,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(64,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(64,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(4,4)))
model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(256,activation="relu"))
model.add(Dense(4,activation="softmax"))
model.summary()
```

- **Xception**

  Xception [13], is a Depthwise Separable Convolutions-based deep convolutional neural network architecture developed by Google Corporation. The "extreme" form of an Inception module is known as Xception. In Inception, 1x1 convolutions were used to compress the original input, and different types of filters were applied to each of the depth spaces based on the input spaces. Xception just reverses this process. Instead, it applies the filters to each depth map individually before compressing the input space using 1X1 convolution across the depth. This approach is nearly equivalent to a depthwise separable convolution. One further difference exists between Inception and Xception. After the initial operation, the existence or absence of a non-linearity. Both processes are followed by a ReLU non-linearity in the Inception model; however, Xception does not add any non-linearity. In figure 3.7, the overall Xception architecture is shown. The data initially passes via the entering flow, then eight times through the middle flow, and lastly through the exit flow. Batch normalization is applied to all Convolution and SeparableConvolution layers (not shown in the picture). A depth multiplier of 1 (no depth expansion) is used in all SeparableConvolution layers. We did image argumentation. Here we resize those images into 120x120 and put that with classid in a different array. Encode classid array by label encoder. Then splitting train and test data. Then make a model and compile it. GlobalAvaragepooling2D, BatchNormalization, and fully linked layers are among the building components of the CNN architecture.

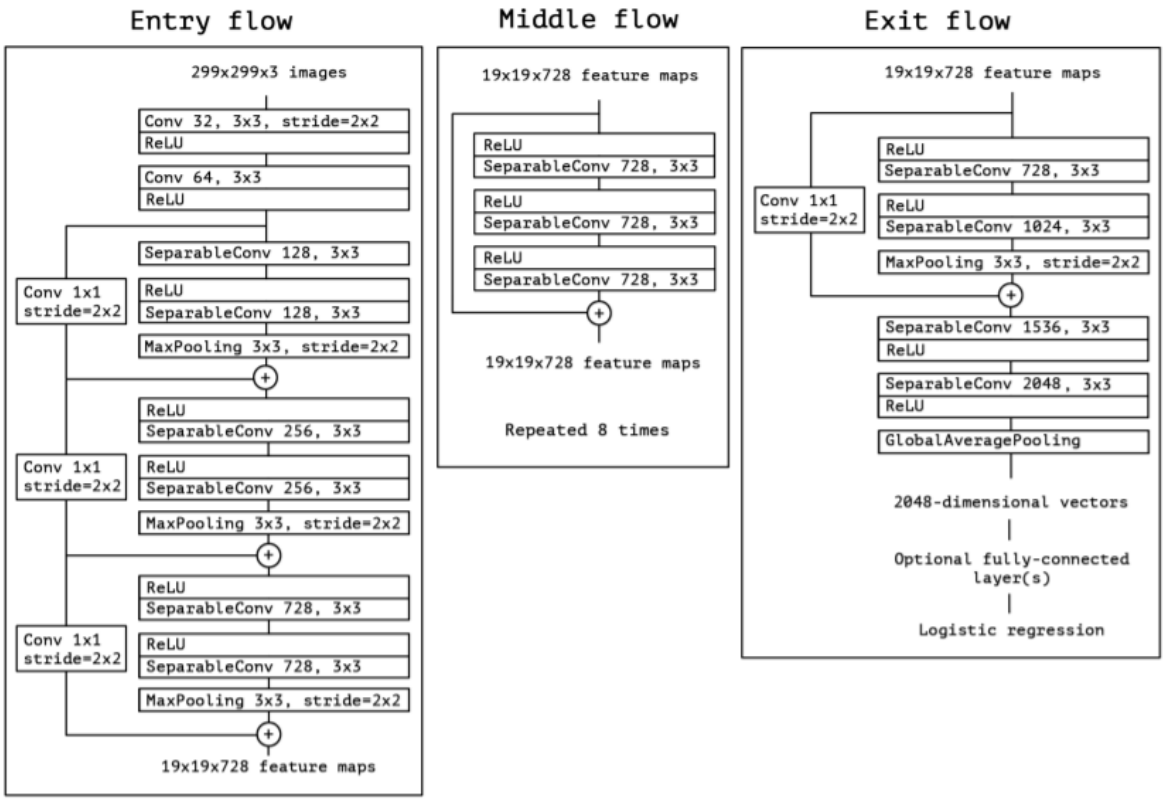

Figure 3.7: Overall Xception Architecture

The Xception model we constructed is given below and the summary of this model is in code which Github link is given in Appendix B.1.

```
Classification_Model = keras.applications.xception.Xception(include_top = False, input_shape = (120,120,3))
```

```
layer = Classification_Model.output
layer = GlobalAveragePooling2D()(layer)

layer = Dense(1024, activation='relu')(layer)
layer = BatchNormalization()(layer)
layer = Dropout(0.3)(layer)

layer = Dense(512, activation='relu')(layer)
layer = BatchNormalization()(layer)
layer = Dropout(0.3)(layer)

layer = Dense(64, activation='relu')(layer)
predictions = Dense(4, activation='softmax')(layer)
model = Model(inputs=Classification_Model.input, outputs=predictions)
model.summary()
```

3.5.2 Machine Learning Models

1. Classification Models

- **Support Vector Machine (SVM)**
  The Support Vector Machine is part of the supervised learning algorithm. This model image is nicely categorized. In high-dimensional spaces, it works well. When the number of features is higher than the number of samples, it's critical to prevent over-fitting when selecting the kernel function and regularization term. We build two arrays after importing the dataset. One of them has a class and an image id in it. The picture id is then added to the stack. After flattening the picture array, normalize the data. We divided the dataset into train and test after normalization. For this SVM model, we'll utilize X train, X test, and y train, y test. We fit X and Y trains by importing the SVM class and setting the Radial Buffer Function (RBF) kernel. It's ready to compile. From figure 3.8, we can get a visual idea of how SVM classifies data. The SVM method helps in the discovery of the optimal line or decision boundary, which is known as a hyperplane. The SVM method locates the closest point of the lines from both classes. Support vectors are the names given to these data points. Margin is the distance between the data points or vectors and the hyperplane. The purpose of SVM is to increase this margin as much as possible. The ideal hyperplane is the one with a long-range margin.

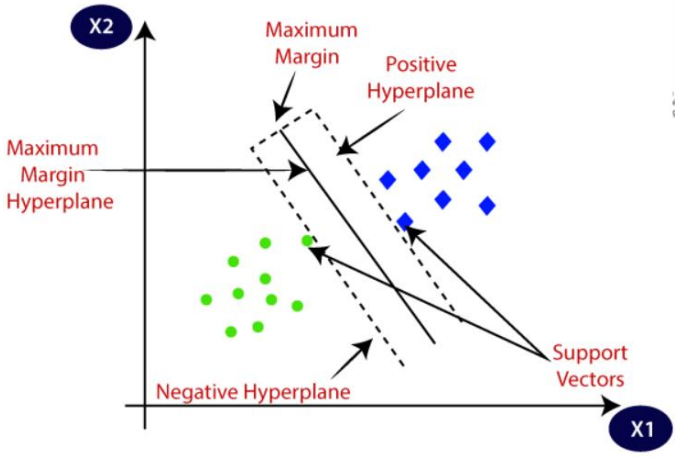

Figure 3.8: Overview of SVM.

The SVM model we used is given below:

```
from sklearn.svm import SVC
svm_classifier = SVC(kernel='rbf', random_state=0)#radial basis function
svm_classifier.fit(X_train,y_train)
svm_predict = svm_classifier.predict(X_test)
```

- **Random Forest**
  The Random Forest [19], is a well-known machine learning algorithm that uses the supervised learning method. In machine learning, it may be utilized for both classification and regression issues. It is based on ensemble learning, which is a method of integrating several classifiers to solve a complicated issue and increase the model's performance. We build two arrays after importing the dataset. One of them has a class and an image id in it. The picture id is then added to the stack. After flattening the picture array, normalize the data. We achieve our accuracy by importing the relevant library classes for this model and then using them. The structure of the random forest is shown in figure 3.9.

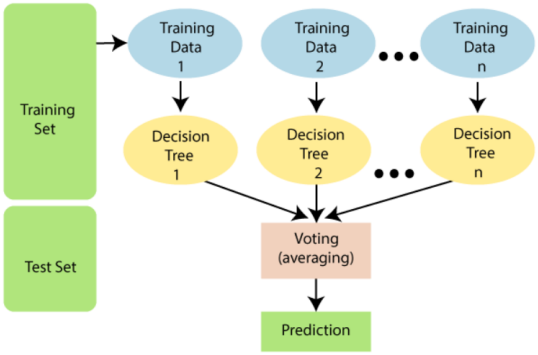

Figure 3.9: Random Forest Structure.

The Random Forest model we used is given below:

```
from sklearn.ensemble import RandomForestClassifier
clf_rndm= RandomForestClassifier(n_estimators = 1000)
clf_rndm.fit(X_train, y_train)
rndm_pred = clf_rndm.predict(X_test)
```

2. Clustering Model

- **K-Nearest Neighbor (KNN)**
  The K-Nearest Neighbor method is based on the Supervised Learning approach and is one of the most basic Machine Learning algorithms. It assumes that the new case/data and existing cases are comparable and places the new case in the category that is most similar to the existing categories. It saves all of the available data and classifies a new data point based on its resemblance to the existing data. This means that fresh data may be quickly sorted into a well-suited category using this method as it arises. After importing the dataset, we build two arrays: K- Nearest Neighbor and N-Nearest Neighbor. One of them has a class and an image id in it. The picture id is then added to the stack. After flattening the image array, normalize the data. We divided the dataset into train and test after normalization. For this KNN model, we now employ the X train, X test, and y train, y test.

Initialize neighbor to 2 by importing the KNeighborClassifier class. Then we added the X and Y trains. It's time to compile. The main topic is to choose k and compute the distance between k and the category point. A category point's closest distance fits under that k point class.

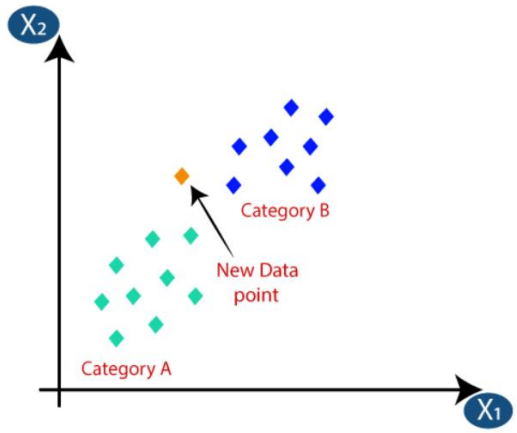

Figure 3.10: KNN Classification

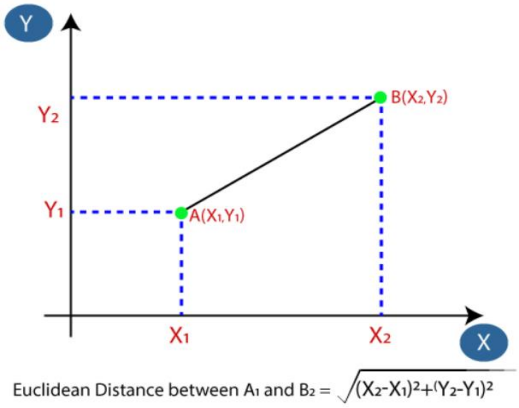

Figure 3.11: Euclidean Distance between two points

The Random Forest model we used is given below:

```
neighbors = list(range(2,9,1))
```

```
from sklearn.neighbors import KNeighborsClassifier

for k in neighbors:
    print("For k = ",k)
    knn_classifier = KNeighborsClassifier(n_neighbors = k, p=2)
    knn_classifier.fit(X_train, y_train)
    knn_pred = knn_classifier.predict(X_test)
    print("Confusion matrix : \n",metrics.confusion_matrix(knn_pred,y_test))
    print("Accuracy Score : ",metrics.accuracy_score(knn_pred,y_test))
    print("Classification report : \n",metrics.classification_report(knn_pred,y_test))
    print("\n")
```

3.6 Assess Performance

It is vital to know how effectively a model gained from the deep and machine learning algorithm works on detecting steel defects in an individual after training different deep and machine learning algorithms on the Severstal Kaggle dataset. We need to utilize some type of performance indicator to compare different models to assess their quality. In Chapter 5, we go through exactly how we measured the performance of several models.

3.7 Summary

In this chapter, we focused on the proposed methodology that we are using in this research. We described our methodology steps, data collection, data preprocessing, and method description with model training. Every step is explained fluently concerning our work.

Chapter 4

Implementation

4.1 Overview

We will explore the implementation of our study in this chapter, which will be based on the proposed methodology outlined in the previous chapter. By reporting on the implementation, a clear picture of the research objective and the process by which we attempted to achieve the research aim is provided. The reader can assess the validity of our research based on how it was implemented. This chapter covers the actual application of our findings, including the environment setup and model creation. The following are the actions we take to put our research into action:

- Environment setup
- Utilizing necessary libraries.

4.2 Environment Setup

For implementing our methodology we are using a Core i7 processor laptop with 20 GB RAM and NVIDIA GTX 1680 4GB GPU. We use the python version 3.7 programming language for our work. Python is one of the most popular and frequently used programming languages in the industry. Python comes with a large number of libraries. Python is renowned as a beginner's level programming language because of its simplicity and ease of use. Python wants its developers to be more productive from development to deployment and maintenance. Python's portability is another reason for its tremendous popularity. When compared to C, Java, and C++, Python's programming syntax is straightforward to learn and has a high degree of abstraction. We used Jupyter NoteBook and VS Code software editor platform to write and run code. They are free, open-source, an interactive web tool that allows academics to mix software code, computational output, explanatory text, and multimedia resources in a single document. By installing Anaconda we set up most of the necessary packages. Other packages like Keras, Tensorflow, and Resterio are manually installed. There is a big issue to notice, for every individual python version almost every package version is different. We set up those versions of packages that are suitable for python version 3.7.

4.3 Importing Necessary Libraries

We imported necessary libraries from packages that are needed for our code. Down below we are going to introduce those libraries and packages.

- **Glob**
  A module that returns all file paths matching a pattern. We may use glob to look for a certain file pattern, or we can use wildcard characters to look for files whose filename matches a specific pattern.
- **Resterio**
  It is a helpful raster processing module that allows us to read and write multiple different raster formats in Python. Rasterio is dependent on GDAL, and when we load the module, Python automatically registers all known GDAL drivers for reading supported formats.
- **Os**
  Python's OS module includes methods for creating and deleting directories (folders), retrieving their contents, altering and identifying the current directory, and more. To interface with the underlying operating system, we must first import the os module.
- **Numpy**
  The most essential Python module for scientific computing is NumPy. It's a Python library that includes a multidimensional array object, derived objects (like masked arrays and matrices), and a variety of routines for performing fast array operations, including mathematical, logical, form manipulation, sorting, selecting, I/O, discrete Fourier transforms, simple linear algebra, simple statistical operations, random simulation, and more.
- **Pandas**
  It is a Python package that provides data structures that are fast, adaptable, and expressive, making working with "relational" or "labeled" data simple and natural. Its purpose is to provide a framework for conducting realistic, real-world data analysis in Python.
- **StandardScaler**
  It works by transforming our data into distribution with a mean of 0 and a standard deviation of 1. This is done feature-by-feature in the case of multivariate data (in other words independently for each column of the data). Given the data distribution, each value in the dataset will be subtracted from the mean and then divided by the overall dataset's standard deviation (or feature in the multivariate case).
- **Scikit-Learn**
  For the Python programming language, Scikit-learn is a popular machine learning package. Scikit-learn is a collection of machine learning tools that include mathematical, statistical, and general-purpose algorithms that may be used to build a range of machine learning technologies. Scikit-learn is a free tool that may be used to create a variety of machine learning and related technology algorithms.

- **Matplotlib**
  Matplotlib is a visualization application that makes use of a Python-based low-level graph charting toolkit. Matplotlib is mostly developed in Python for platform portability, with minor portions written in C, Objective-C, and Javascript. Sequential In Keras, the simplest technique to build a version is sequential. It allows you to create a version layer with the help of layers. The weights of each layer match the weights of the layer below it.
- **Sequential**
  In Keras, the simplest technique to create a model is sequential. It allows you to layer-by-layer construct a model. The weights of each layer match depth-wise the weights of the layer below it. To add layers to our model, we utilize the ‘add ()' method. A Sequential version is appropriate for an irrefutable stack of layers with one entry tensor and one output tensor for each layer. There are several inputs and outputs in your version. There are several inputs and outputs on each of your layers. Layer sharing is required.
- **Keras**
  Keras is a Python-based deep mastering API that stands on the summit of TensorFlow's system mastering platform. It was created to enable quick experimentation. It is critical to be able to go from concept to outcome as quickly as possible when doing research.
- **TensorFlow**
  TensorFlow is a Python library for real-time numerical computing that was built and released by Google. It's a foundation library that may be used to quickly generate Deep Learning models, or it can be used in conjunction with wrapper libraries to simplify methods built on top of TensorFlow.
- **Tdqm**
  A module is used to create progress meters or progress bars.
- **RandomForestClassifier**
  A random forest is a Meta estimator that uses averaging to increase predicted accuracy and control over-fitting by fitting some decision tree classifiers on various sub-samples of the dataset.
- **SVC**
  SVC is an expression for "C-Support Vector Classification." Libsvm is used in the implementation. The fit time scales at least quadratically with the number of samples, and beyond tens of thousands of data, it may be unworkable.
- **Dense()**
  Construct fully connected layers, which perform categorization at the capacities extracted with the help of convolutional layers and downsampled with the help of pooling layers.
- **Dropout**
  A dropout is a training approach in which neurons are chosen at random and not recorded during the process. They "disappeared" randomly. This strategy ensures that their contribution to downstream neuron activation is eliminated temporally at the forward pass and that any weight modifications to the neuron are not performed at the backward pass.

This method has the main advantage of preventing all neurons in a layer from maximizing their weights at the same time. This adaptation, which occurs in random groupings, prevents all neurons from converging to the same objective, resulting in weight decorrelation. It helps to reduce the overfitting problem.

- **Conv2D()**
  Constructs convolutional layers in two dimensions. As parameters, it accepts the number of filters, filter kernel size, padding, and activation function.
- **Max-pooling2D()**
  The max-pooling layer algorithm is used to create a two-dimensional pooling layer. As parameters, it accepts the pooling filter size and stride.
- **Flatten**
  Image of the outcome of knocking down Keras layers in-depth research. The flatten feature converts multi-dimensional input tensors to a single dimension, allowing you to model your entry layer and design your neural community model, then skip the relevant information into each neuron of the model properly. We flatten our image intensity matrix into a 1D array. That's why number of feature is large.
- **LabelEncoder**
  As a one-shot numeric array, encode category information. LabelEncoder is a tool for normalizing labels. It may also be used to convert non-numerical labels to numerical labels (as long as they are hashable and comparable).
- **To-categorical**
  A Numpy array (or) a vector with integers that represent distinct categories may be transformed into a NumPy array (or) a matrix with binary values and columns equal to the number of categories in the data using the function to categorical().
- **CV2**
  Computer vision enables computers to do the same tasks as people while maintaining the same level of efficiency. The CV is an abbreviated form of computer vision in OpenCV, which is described as a branch of research that helps computers in understanding the content of digital pictures such as photographs and videos. The goal of computer vision is to figure out what's going on in the images. It extracts the description from the images, which may be an item, a written description, a three-dimensional model, or something else entirely. Cars, for example, can be equipped with computer vision, which will be able to recognize and react to various items on the road, such as traffic lights, people, traffic signs, and so on.
- **Softmax**
  In a neural network that does multi-class classification, the softmax layer is often the last output layer (for example object recognition). The term is derived from the softmax function, which accepts numerous score values (as input) and squashes them into numbers in the range of 0 to 1 with a sum of 1.

- **Adam**
  Adam optimization is a stochastic gradient descent approach based on adaptive estimates of first-order and second-order moments. The 1st-moment estimations' exponential decline rate. In the majority of circumstances, Adam is the best adaptive optimizer. Good for sparse data: the adaptive learning rate is ideal for sparse data.
- **ReLU**
  Every other non-linear activation feature that has gained a reputation within the deep researching sector is the ReLU feature. Rectified Linear Unit (ReLU) is an abbreviation for Rectified Linear Unit. The main advantage of using the ReLU function over other activation capabilities is that it does not activate all of the neurons at the same time.
- **GlobalAvaragePooling2D**
  GlobalAveragePooling2D takes a 4D tensor as input. It uses the mean for all of the channels' height and width dimensions. The dimension, as a result, is 2D (batch dim, n channels). The same is done by GlobalMaxPooling2D, but with a maximum operation. It applies average pooling on the spatial dimensions until each spatial dimension is one.
- **BatchNormalization**
  Batch normalization is a technique for standardizing the inputs to a layer of a deep learning neural network automatically. Batch normalization has the effect of drastically speeding up the training process of a neural network, as well as improving the model's performance in some situations via a little regularization impact.

4.4 Summary

We detailed how we built up our research environment and which libraries we used for our research in this chapter. Following that, we discussed why we chose Python as our programming language of choice for our work. The following chapter will continue the topic of the outcomes by proposing a model for detecting steel defects using this implementation described in this chapter.

Chapter 5

Results and Analysis

5.1 Overview

In this chapter, we'll go through the outcomes acquired using the methodology and procedure mentioned in the previous two chapters. We'll compare the model's performance in more detail, as well as examine the performance measurement measures that were employed.

5.2 Performance Measurement Metrics

Performance metrics are measurable statistics that are used to track operations inside a company utilizing important indicators such as activities, employee behavior, and productivity. These metrics track and assess how well a company's overall objectives are being accomplished.

5.2.1 Confusion Matrix

A confusion matrix is a summary of classification problem prediction outcomes. The number of successful and unsuccessful predictions is collected and split down by class using count values. The confusion matrix shows the various ways in which your classification model becomes confused while making predictions. It informs you not only about the faults made by your classifier but also about the sorts of errors that are being made.

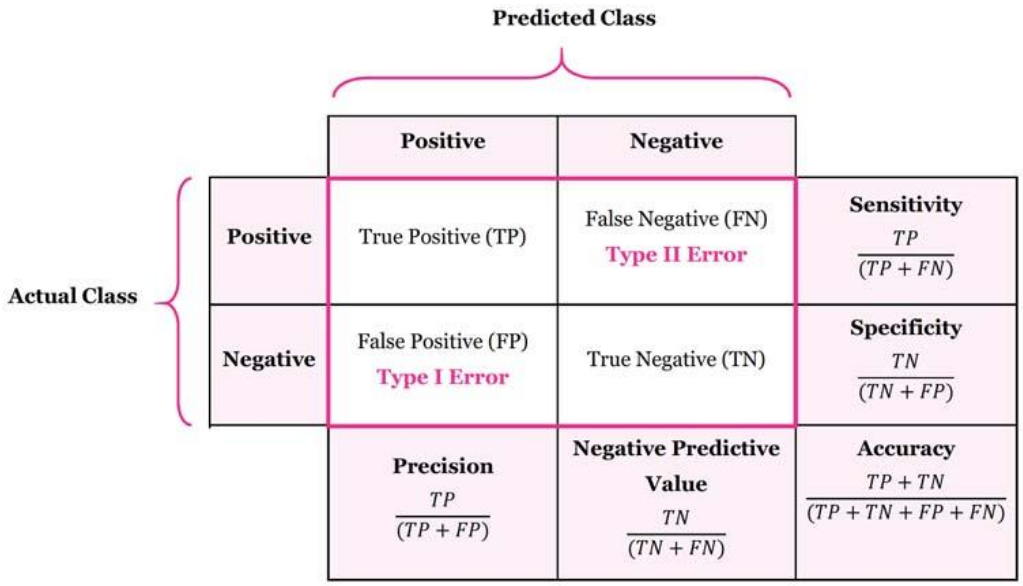

Figure 5.1: Confusion matrix (2x2) and with Advanced classification metrics.

Description of confusion matrix given below:

- **True Positive (TP):** The amount of right predictions that an example is positive, or the number of positive classes correctly recognized as positive, is known as True Positive (TP).
- **False Negative (FN):** The number of inaccurate predictions that an example is negative, or the number of positive classes wrongly classified as negative, is known as False Negative (FN).
- **False Positive (FP):** The amount of inaccurate predictions that an example is positive, or the number of negative classes incorrectly recognized as positive, is known as false positive (FP).
- **True Negative (TN):** The amount of correct predictions that an example is negative (True Negative (TN): is the number of negative classes correctly recognized as negative.
- **Sensitivity:** True Positive Rate or Recall are other terms for sensitivity. It is a measurement for the set of positive cases classified as such by a classifier. It ought to be higher because the higher the sensitivity model accuracy will be high.
- **Specificity:** True Negative Rate is another name for specificity. It's a measurement for the number of negative samples classified as such by the classifier. A high level of specificity is required too.
- **Precision:** The ratio of the total number of accurately categorized positive cases to the total number of predicted positive examples is referred to as precision. It demonstrates the accuracy of positive prediction.
- **Accuracy:** The proportion of correct guesses in the total number of forecasts is known as accuracy. The most often used statistic for evaluating a model, however, is not a strong predictor of its performance.
- **F1 Score:** The recall (sensitivity) and precision are weighted in the F1 score. When attempting to find a compromise between Precision and Recall, the F1 score may be a useful option. It enables the issue of distinguishing between models with low recall and high precision or vice versa to be solved by combining recall and precision into a single equation.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5.3. Experimental Results

We applied a total of five methods for defect detection. Here we are going to see the result after evaluation of the used model for this thesis. They gave us different results of detection. We can identify which method is better among the five of them.

5.3.1 Classification Reports of Machine Learning Approaches

- **Random Forest:**
  The confusion matrix and classification report for this are given below:

20	2	6	0
6	10	1	0
66	3	442	64
0	0	17	3

Figure 5.2: Random Forest Confusion Matrix.

	precision	recall	f1-score	support
1	0.22	0.71	0.33	28
2	0.67	0.59	0.62	17
3	0.95	0.76	0.84	555
4	0.04	0.15	0.07	20
accuracy			0.73	620
macro avg	0.47	0.55	0.47	620
weighted avg	0.88	0.73	0.79	620

Figure 5.3: Classification Result of Random Forest.

Finally, we got 73.38% detection accuracy using a random forest classifier.

- **Support Vector Machine (SVM):**
  The confusion matrix and classification report for this are given below:

20	0	2	0
1	5	1	0
71	10	443	65
0	0	0	2

Figure 5.4: SVM Confusion Matrix.

	precision	recall	f1-score	support
1	0.22	0.91	0.35	22
2	0.33	0.71	0.45	7
3	0.99	0.75	0.86	589
4	0.03	1.00	0.06	2
accuracy			0.76	620
macro avg	0.39	0.84	0.43	620
weighted avg	0.96	0.76	0.83	620

Figure 5.5: Classification Result of SVM.

Finally, we got 75.80% detection accuracy using the Support vector machine.

- **K-Nearest Neighbor (KNN):**
  The confusion matrix and classification report for this are given below

57	3	33	3
8	11	7	0
27	1	402	59
0	0	4	5

Figure 5.6: KNN Confusion Matrix.

	precision	recall	f1-score	support
1	0.62	0.59	0.61	96
2	0.73	0.42	0.54	26
3	0.90	0.82	0.86	489
4	0.07	0.56	0.13	9
accuracy			0.77	620
macro avg	0.58	0.60	0.53	620
weighted avg	0.84	0.77	0.80	620

Figure 5.7: Classification Result of KNN.

Finally, we got 76.61% detection accuracy using the Support vector machine.

5.3.2 Report of Deep Learning Approaches

- **Convolutional Neural Network (CNN):**
  By setting up batch size 128 and epochs 15 we evaluate the CNN model. This model gives us 75.81% detection accuracy. This model's accuracy and loss curves are given below:

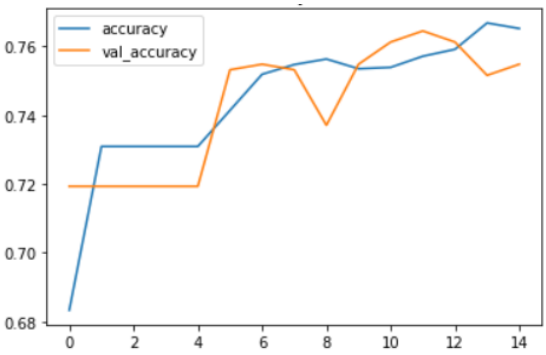

Figure 5.8: CNN Accuracy Graph

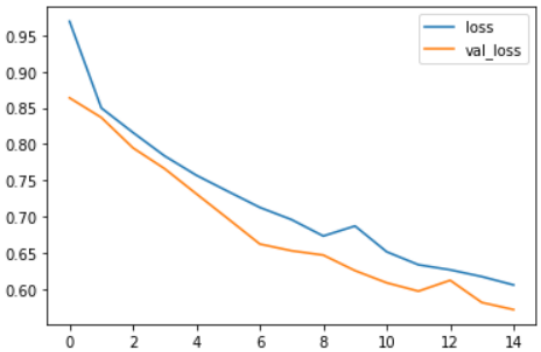

Figure 5.9: CNN Loss Graph

- **Xception:**
  By setting batch size 128 and epochs 30 we evaluate the Xception model. This model gives us 89.80 defect detection accuracy. This model's accuracy and loss curves are given below:

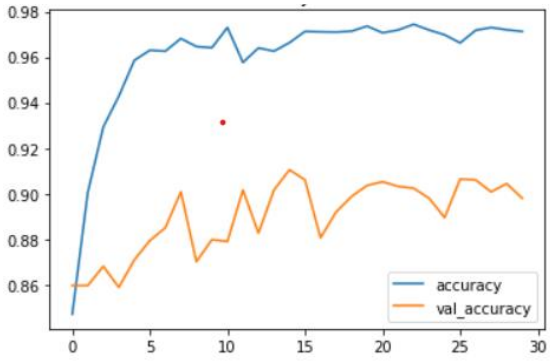

Figure 5.10: Xception Accuracy Graph

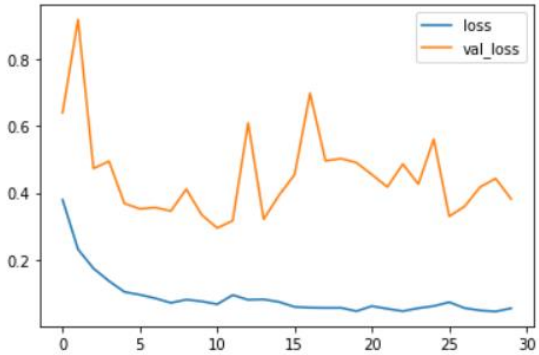

Figure 5.11: Xception Loss Graph

5.4 Comparison Table

	No.	Model	Accuracy(%)
Machine Learning	**1**	Random Forest	73.38
	2	Support Vector Machine	75.80
	3	K-Nearest Neighbor	76.61
Deep Learning	**4**	Convolutional Neural Network	75.81
	5	Xception	**89.80**

Table 5.1: Model Comparison Table

We can observe that Random forest has a detection accuracy of 73.38 percent. Then we use SVM, which has a 75.80% better testing score than random forest. We used the KNN clustering model, which has a detection accuracy of 76.61 percent. KNN outperforms the other two machine learning algorithms in terms of detection accuracy. We employ two deep learning models: CNN and Xception. The CNN model has a score of 75.81 percent, whereas the Xception model has a score of 89.80 percent. As a result, we may conclude that KNN is superior to other Machine Learning

models. Xception outperforms CNN in the Deep Learning model. Finally, the task was performed on all of the models and the results were compared, with Xception achieving the best accuracy of 89.80 percent.

5.5 Summary

We began by learning about the key terms used in performance measuring metrics. Then we learned about the confusion matrix and how they functioned. We exhibit the results of all of the models we've constructed so far in terms of analysis. Finally, we created a comparison table to analyze the accuracy of the models and decided on the Xception model due to its accuracy.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The goal of this thesis is to develop a model for identifying defects on steel sheets using images from high-frequency cameras. The long-term objective is to guarantee that all industries can supply defect-free steel sheets. Otherwise, items or other structures built of this steel might be put at risk. To complete the objective, we looked at five different approaches. There are two deep learning methods (CNN and Xception), one clustering method (KNN), and two classification methods (SVM and Random forest). In a single study of steel defect identification, we attempted to demonstrate the contrast between deep learning and machine learning outcomes. In terms of detection accuracy, KNN exceeds the other two machine learning algorithms by 76.61 percent. In the Deep Learning model, Xception exceeds CNN with an accuracy of 89.80%. As a result, among those five methods, Xception can give us the highest number of defect detection with less error and we can state that deep learning approaches are a superior alternative for detecting steel defects.

6.2 Future Plan

On the rare defect classes, further data augmentation techniques such as rotation and scaling on the images should be done due to the imbalance between different kinds of labels. Measure accuracy without resizing the image. To see what the difference in accuracy is, we need to test accuracy without altering the size. Different image sizes are used to test accuracy. It's possible that increasing the size of the image will improve the accuracy of detection defects. That is something we must investigate. It is possible to generate more accurate results by combining different models in a single system. This is something more that has to be looked at.

References

[1] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence, 40(4):834–848, 2017

[3] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In Proceedings of the IEEE international conference on computer vision, pages 1520–1528, 2015

[4] R. Girshick. Fast R-CNN. In Proceedings of the IEEE international conference on computer vision, pages 1440–1448, 2015

[5] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and ´ S. Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2117–2125, 2017

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft coco: Common objects in context. In the European conference on computer vision, pages 740–755. Springer, 2014.

[7] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3213–3223, 2016.

[8] Z. Zhang, A. G. Schwing, S. Fidler, and R. Urtasun. Monocular object instance segmentation and depth ordering with CNN. In Proceedings of the IEEE International Conference on Computer Vision, pages 2614–2622, 2015

[9] Z. Zhang, S. Fidler, and R. Urtasun. Instance-level segmentation for autonomous driving with deep densely connected MRFs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 669–677, 2016.

[10] B. Romera-Paredes and P. H. S. Torr. Recurrent instance segmentation. In the European conference on computer vision, pages 312–329. Springer, 2016.

[11] L. Shen, Z. Lin, and Q. Huang. Relay backpropagation for effective learning of deep convolutional neural networks. In the European conference on computer vision, pages 467–482. Springer, 2016.

[12] Severstal: Steel Defect Detection

[13] F. Chollet. Xception: Deep learning with depth-wise separable convolutions. In

Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1251–1258, 2017.

[14] Weidong Zhao, Feng Chen, Hancheng Huang, Dan Li, Wei Cheng, "A New Steel Defect Detection Algorithm Based on Deep Learning", *Computational Intelligence and Neuroscience*, vol. 2021, Article ID 5592878, 13 pages, 2021.

[15] Qianlai Sun (孙**前来**), Yin Wang (**王**银) and Zhiyi Sun (孙**志毅**)," Rapid surface defect detection based on singular value decomposition using steel strips as an example" AIP Advances **8**, 055209 (2018).

[16] Kun Qian, "Automated Detection of Steel Defects via Machine Learning based on Real-Time Semantic Segmentation" ICVIP 2019: Proceedings of the 3rd International Conference on Video and Image Processing, Pages 42–46, December 2019.

[17] Matthew Browne, Saeed Shiry Ghidary, "Convolutional Neural Networks for Image Processing: An Application in Robot Vision" Australasian Joint Conference on Artificial Intelligence AI 2003: Advances in Artificial Intelligence pp 641-652.

[18] Zhu Fan, Jia-Kun Xie, Zhong-yu Wang, Pei-Chen Liu, " Image Classification Method Based on Improved KNN Algorithm" *J. Phys.: Conf. Ser.* 1930, 2021 International Conference on Sensing Technology and Applications (ICSTA 2021) 23-25 April 202

[19] Baoxun Xu, Yunming Ye, Lei Nie, "An improved random forest classifier for image classification" In proceedings of the 2012 IEEE International Conference on Information and Automation, 6-8 June 2012.

Appendix A

A.1: CSV Dataset

	A	B	C
1	ImageId	ClassId	EncodedPixels
2	0002cc93b	1	29102 12 29346 24 29602 24 29858 24 30114 24 30370 24 30626 24 30882 24 31139 23 31395 23 31651 23 31907 23 32163 23 32419 23 32675 23 77918 27 78174 55 78429 60 78685 64 7
3	0007a71bf	3	18661 28 18863 82 19091 110 19347 110 19603 110 19859 110 20115 110 20371 110 20627 110 20883 110 21139 110 21395 110 21651 110 21962 55 293125 251 293381 251 293637 251
4	000a4bcdc	1	37607 3 37858 8 38108 14 38359 20 38610 25 38863 28 39119 28 39375 29 39631 29 39887 29 40143 29 40399 29 40655 30 40911 30 41167 30 41423 30 41679 31 41935 31 42191 31 424
5	000f6bf48	4	131973 1 132228 4 132483 6 132738 8 132993 11 133248 13 133503 16 133757 19 134012 22 134267 24 134522 26 134777 29 135032 31 135287 34 135542 36 135796 40 136050 43 136
6	0014fce06	3	229501 11 229741 33 229981 55 230221 77 230468 92 230623 10 230724 95 230845 11 230869 28 230979 97 231094 63 231235 97 231344 70 231490 99 231593 79 231746 183 232002 1
7	0025bde0c	3	8458 14 8707 35 8963 48 9219 71 9475 88 9731 88 9987 89 10243 89 10499 90 10755 90 11011 91 11267 91 11523 92 11779 92 12035 93 12291 93 12547 94 12803 94 13060 94 13316 94
8	0025bde0c	4	315139 8 315395 15 315651 16 315906 17 316162 18 316418 19 316674 19 316930 20 317186 20 317441 22 317697 23 317953 23 318209 24 318465 25 318721 25 318977 26 319233 27 3
9	002af848d	4	290800 6 291055 13 291311 15 291566 18 291822 18 292077 19 292333 19 292588 20 292844 20 293099 21 293354 22 293610 22 293865 23 294121 23 294376 24 294632 24 294887 25 2
10	002fc4e19	1	146021 3 146275 10 146529 40 146783 46 147038 52 147292 59 147546 65 147800 70 148055 71 148311 72 148566 73 148822 74 149077 75 149333 76 149588 77 149844 78 150100 78 1
11	002fc4e19	2	145658 7 145901 20 146144 33 146386 47 146629 60 146872 73 147115 86 147364 93 147620 93 147876 93 148132 93 148388 93 148644 93 148900 93 149156 93 149412 93 149668 46
12	0030401a5	4	186833 1 187089 3 187344 6 187600 7 187855 10 188111 11 188366 14 188622 16 188877 18 189133 20 189388 22 189644 24 189900 26 190155 28 190411 28 190666 29 190922 29 1911
13	0046839bc	3	152926 1 153180 4 153434 6 153689 8 153943 11 154198 12 154453 14 154708 15 154963 15 155218 16 155473 16 155728 17 155983 18 156238 18 156493 19 156748 19 157003 20 157
14	005d86c25	3	331 18 587 53 843 89 1099 124 1355 159 1611 177 1867 177 2123 177 2380 177 2636 177 2892 177 3148 177 3404 177 3660 177 3916 177 4172 177 4428 177 4684 177 4940 177 5196 17
15	005f02e20	3	318652 8 318863 6 318903 17 319114 15 319158 20 319368 21 319413 22 319622 26 319667 26 319876 32 319922 28 320130 36 320177 30 320385 38 320432 32 320641 39 320687 35 3
16	005f19695	3	123137 7 123393 19 123649 32 123905 44 124161 57 124417 69 124673 82 124929 94 125185 107 125441 119 125697 126 125953 125 126209 124 126465 123 126721 122 126977 121 1
17	00862162f	4	215548 3 215798 9 216051 12 216306 13 216560 15 216815 16 217070 17 217326 17 217582 17 217837 18 218093 18 218349 18 218605 18 218861 18 219116 19 219372 19 219628 18 21
18	0088260da	3	26044 13 26283 40 26522 67 26762 92 27001 119 27231 155 27452 195 27686 217 27934 225 28182 234 28430 242 28678 250 28930 254 29186 254 29442 255 29698 255 29954 255 3021
19	008d0f87b	3	72425 21 72641 61 72856 102 73071 142 73287 182 73522 203 73778 203 74034 202 74291 201 74547 201 74803 201 75059 200 75315 200 75572 199 75828 199 76084 198 76340 198 76
20	008ef3d74	1	356336 4 356587 11 356838 18 357089 25 357340 32 357591 38 357843 43 358094 48 358345 54 358596 59 358847 65 359100 68 359356 69 359612 69 359867 70 360123 69 360379 68 3
21	008ef3d74	2	375439 5 375687 14 375935 24 376182 34 376430 43 376678 52 376925 63 377173 72 377420 82 377668 91 377916 101 378163 111 378411 120 378659 127 378906 134 379154 141 379
22	0095cd374	3	10674 18 10926 48 11178 52 11430 56 11682 60 11934 64 12186 68 12438 72 12692 74 12947 75 13203 75 13458 76 13714 76 13969 77 14224 77 14480 77 14735 78 14990 79 15246 79 1
23	00ac8372f	1	101742 3 101998 12 102253 19 102301 22 102509 30 102549 34 102765 76 103020 79 103276 81 103532 81 103788 81 104043 82 104299 82 104555 81 104811 80 105067 80 105324 78 1

Appendix B

B.1: Source Code Link

https://github.com/StrikerAbir/Steel-Decfect-Detection-with-High-Frequency-Camera

B.2: CNN Model Summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 118, 118, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 39, 39, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 1028
=================================================================
Total params: 139,780
Trainable params: 139,780
Non-trainable params: 0
```