Pseudo defect reduction in Surface Inspection System at CRM Bara

Made by:
Vipul Sharma
Department of Computer Science
University of Minnesota, Morris

Under the guidance of:
Mr. Ashish Tiwari
Principal Technologist
Insrumentation & Control
Automation Division
Tata Steel

Contents

March 19, 2019

1 About the company

Tata Steel has been in business of steel-making for the last 111 years. It was established in Jamshedpur, India in the year 1907 and has been an integral part of the 150-year-old Tata group. Bringing to reality the vision of its founder, J.N.Tata, who inspired the steel and the power industry in India, the Tata Steel Group is the 10th largest steel manufacturer in the world and is known to be the hallmark of corporate citizenship and business ethics. Tata Steel is one of the world's most geographically diverse steel producers with operations in 26 countries and commercial presence in 50 countries. It had a production capacity of 27.5 MnTPA as of March 31, 2018. The company operates with a completely integrated value chain that extends from mining to finished steel goods. The company is driven by innovation, guided by values and poised for future. Its aspirations for growth are supported by its efforts of continual improvements in its processes, building efficiency and adding value to its products while meeting stakeholder expectations across the value chain. Consistent with the vision and values of the founder J.N. Tata, Tata Steel strives to strengthen IndiaâĂŹs industrial base through effective utilization of staff and materials. The means envisaged to achieve this are cutting-edge technology and high productivity, consistent with modern management practices. Tata Steel recognizes that while honesty and integrity are the essential ingredients of a strong and stable enterprise, profitability provides the main spark for economic activity. Overall, the Company seeks to scale the heights of excellence in all it does in an atmosphere free from fear, and thereby reaffirms its faith in democratic values.

2 Cold Rolled Steel

Tata Steel offers a comprehensive range of cold-rolled steel comprising both continuously-annealed and batch-annealed products. This means we can tailor our offer to meet your precise requirements for form ability, strength, surface finish or flatness. These advanced and ultra high-strength cold-rolled steels DP600, DP800, DP1000 and HQ1500 are lightweight and enable design freedom for complex parts. Their high yield and ductility ensure shape accuracy and tolerance, which enable thickness reduction and maintained in-service performance for difficult-to-form parts. Coupled with consistent quality, its wide product selection provides opportunities for the customers to optimize both the products and processes. The benefits of our cold-rolled steel include consistent and reliable end product quality, lighter and stronger products, repeatable, trouble-free processing, maximized yield and reduced waste and rework.

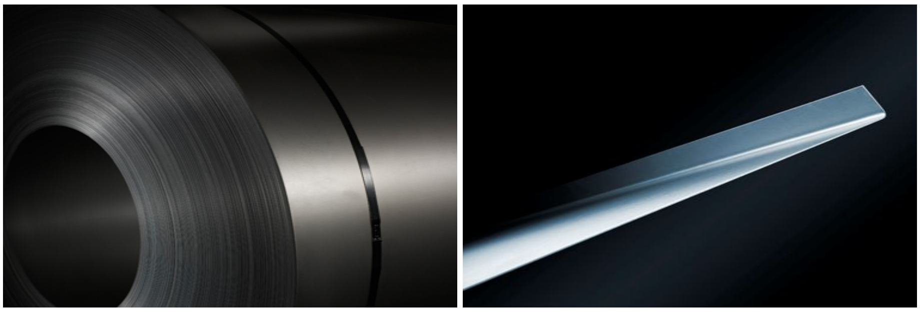


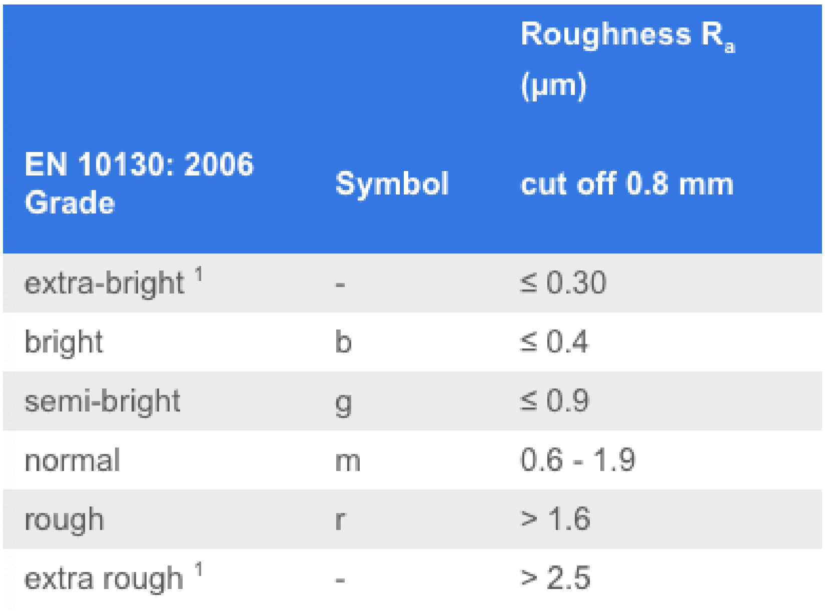
Figure 2.1: Cold Rolled Steel

The steel can be used in various appliances including automotive components, bathtubs, construction and building components, domestic appliances, drums and pressure vessels, electrical goods, electrolytic coating, feed stock for galvanizing and coating, furniture, radiators, tubes and sections etc. The grades of cold-rolled steel offered by Tata Steel include advanced high-strength steels and grades for enamelling and forming. Cold-rolled steel in the following conditions which include annealed and skin-passed coils, full hard coils, sheets, discs and slit wide coils. Tata Steel produces cold-rolled and annealed strip by either the continuously-annealed (CA) or batch-annealed (BA) processes. Surface quality defects that do not influence the formabillty or the application of surface coatings are permitted. They are defects such as pores,minor scratches, slight indentations, small grooves or slight discoloration. The better side must be free of defects that can spoil the uniform appearance of a high-quality paint or of an electrolytic coating. As a rule, the upper side of the strip is inspected; on request, the strip can be wound so that the inspected side is the underside. Cold-rolled annealed and skin-passed steel is available in several surface textures. Unless specified otherwise, Tata Steel will supply normal roughness. Figure 2.2 shows the range of surface textures according to EN 10130:2006.

The phase II expansion of CRM BARA includes installation of 0.3 MnTPA hot rolled skin passing mill (HSPM) to meet the increased demand of Hot-Rolled, Pickled, Skin passed and Oiled products (HRSPO) in the Automotive sector for high-end customers. In order to cater to the input requirement of HSPM, the production capacity of the existing pickling line has also been increased to 0.68 MnTPA from the designed capacity of 0.5 MnTPA.

3 Abstract

Automatic Surface Inspection System is of more value now a days as compared to the quality control inspection in industrial area and one of its major problems is, metallic defect detection which is performed in complex industrial scenarios. Most image processing techniques involves basic machine learning techniques which can only be operated under specific environment including illumination and defect contours conditions. This report is based on classifying metallic defects using twofold procedure applied in surface inspection system. A cascaded architectural auto encoder is used to differentiate the defects which transforms



		Roughness R_a (µm)
EN 10130: 2006 Grade	Symbol	cut off 0.8 mm
extra-bright [1]	-	≤ 0.30
bright	b	≤ 0.4
semi-bright	g	≤ 0.9
normal	m	0.6 - 1.9
rough	r	> 1.6
extra rough [1]	-	> 2.5

Figure 2.2: Range of surface textures according to EN 10130:2006

the input image into pixel wise prediction mask. The defects are then classified into their specific classes via a convolution neural network.This project is done at CRM Bara where the surface inspection system lies. There is a server room that controls the logistics of the software that has been built to gain information about the rolled coils. The surface inspection system offers various outputs regarding the coils. This included defect statistics, image provision of defects, online live graphs regulators of the moving coils, an offline classifier and a review application. Currently the offline classifier holds 27 different defect statistics which are being categorized using employees. It is thus significantly inaccurate because of of issues like uneven illumination, strong reflection, background noise and even man held detection problems coming from being too tired etc. Traditional image processing methods including structural, threshold, spectral and even model based methods tend to give problems in adjusting their threshold values when a new situation arises, thus sometimes leading to the redesign of the algorithm itself which is a tedious task. For the data set available, there are twenty seven types of defects : Carbon Shots, Coil Mark Break, DENT, Edge Damage, Fold Mark, Gouge Mark, High Temperature Scale, Hole, Lamination, Length Wise Mark, Line Stoppage, Over pickle, Pickling Patch Type, Pitting, Pseudo, RIS, Roll Mark, Scratch, Secondary Scales, Silver, Squeeze Roll Mark, Stone Rubbing, Tongue Mark, Uncl, Underpickle, Watter Carry Over and Width Wise Mark and each of these defects are categorized into five different severity levels including : Severity 1, Severity 2, Severity 3, Severity 4 and Severity 5 in which 5th is the highest and 1st being the lowest.

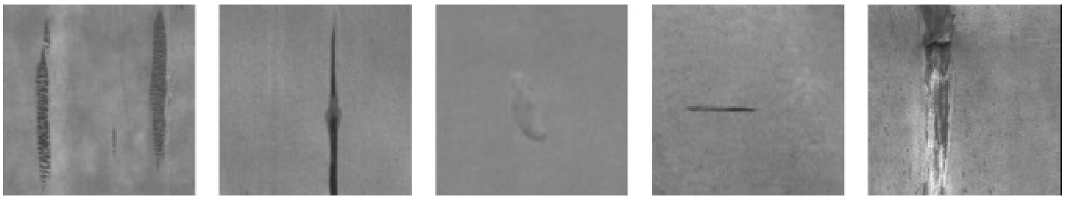


Figure 3.1: Defects on Steel. From the left: Oxide scales, Entrapped slag, Roll Mark, Horizontal Scratch, Sharp Scarring

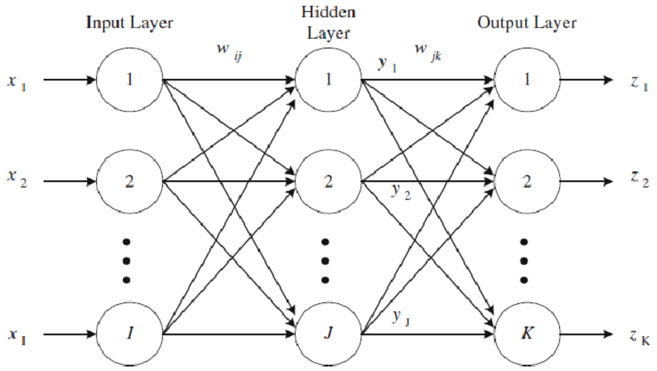


Figure 4.1: A basic neural network

4 Neural Network

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. ANNs are processing devices (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the mammalian cerebral cortex but on much smaller scales. A large ANN might have hundreds or thousands of processor units, whereas a mamalian brain has billions of neurons with a corresponding increase in magnitude of their overall interaction and emergent behavior. Although ANN researchers are generally not concerned with whether their networks accurately resemble biological systems, some have. For example, researchers have accurately simulated the function of the retina and modeled the eye rather well.Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. (Figure 4.1) The hidden layers then link to an 'output layer' where the answer is output.Most ANNs contain some form of 'learning rule' which modifies the weights of the connections according to the input patterns that it is presented with.Although there are many different kinds of learning rules used by neural networks, this demonstration is concerned only with one; the delta rule. The delta rule is often utilized by the most common class of ANNs called 'backpropagational neural networks' (BPNNs). Back propagation is an abbreviation for the backwards propagation of error.With the delta rule, as with other types of back-propagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. (Figure 4.2) Within each hidden layer node is a sigmoidal activation function which polarizes network activity and helps it to stabilize. Back propagation performs a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is that theoretical solution with the lowest possible error. The error surface itself is a hyperparaboloid but is seldom 'smooth' as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minimum' which is not the best overall solution.Since the nature of the error space can not be known a prioi, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.Once a neural network is 'trained' to a satisfactory level it may be used as an analytic tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no back propagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.It is also possible to over-train a neural network, which means that the network has been trained exactly to respond to only one type of input; which is much like rote memorization. If this should happen then learning can no longer occur and the network is referred to as having been "grand mothered" in neural network jargon. In real-world applications this situation is not very useful since one would need a separate grand mothered network for each new kind of input.Neural networks are universal approximators, and they work best if the system you are using them to model has a high tolerance to error. They work very well for capturing associations or discovering regularities within a set of patterns, where the volume, number of variables or diversity of the data is very great, the relationships between variables are vaguely understood or the relationships are difficult to describe adequately with conventional approaches.

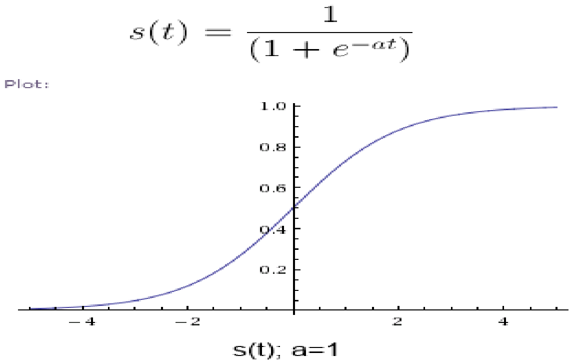


Figure 4.2: Sigmoid Activation Function

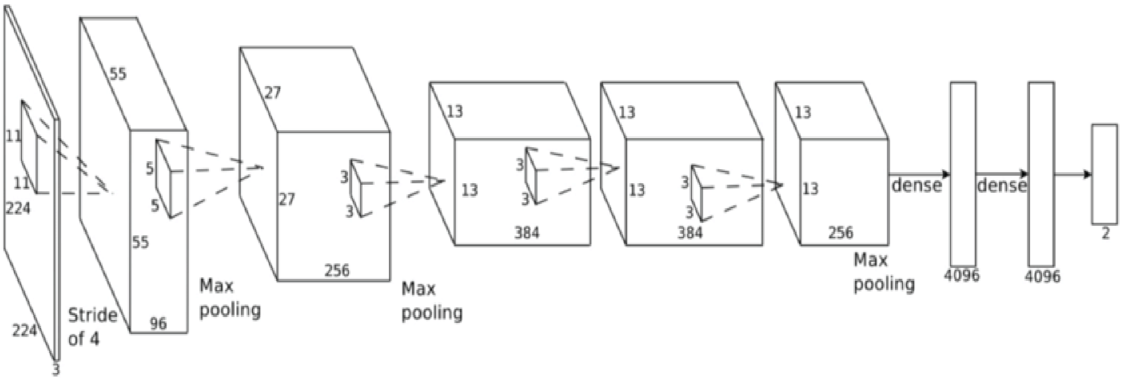
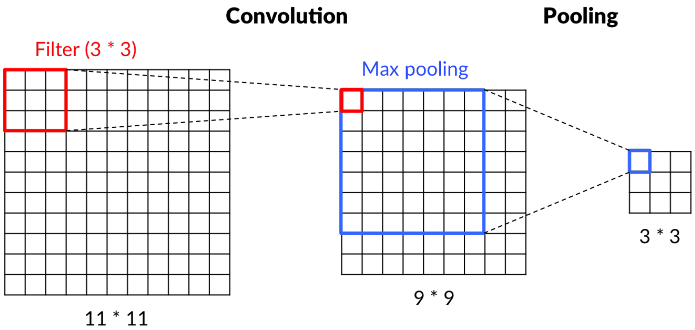
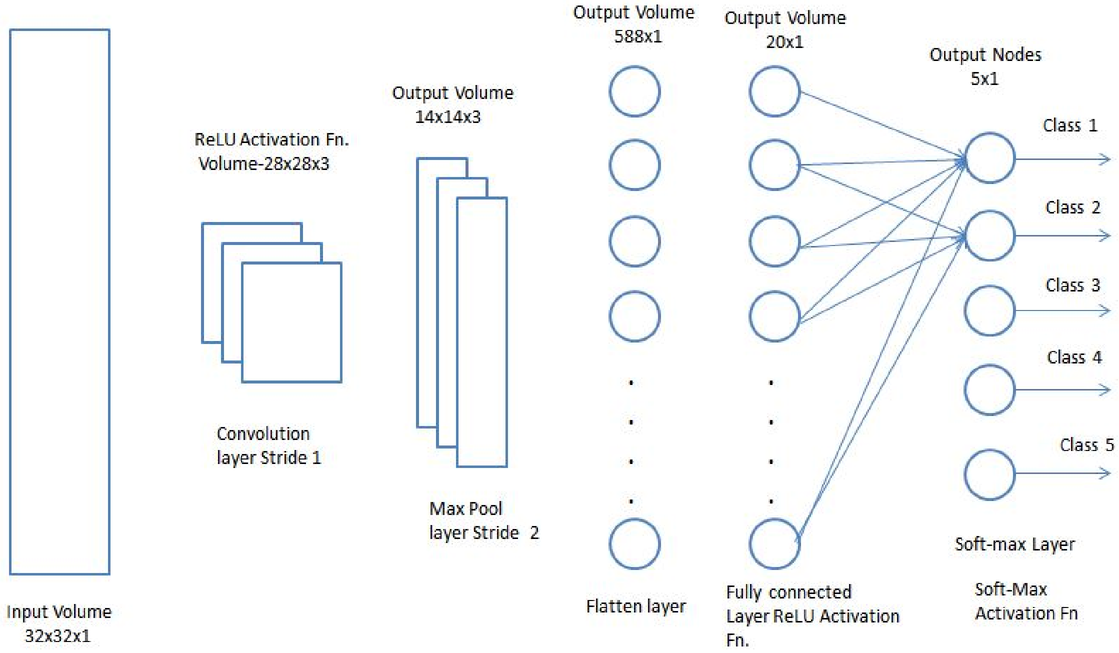


Figure 5.1: A basic convolution neural network (called Alexnet introduced 2012)

5 Convolution Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area. Since an image is a matrix it is flattened out in a single vector and fed into a multilevel perceptron for classification purposes. A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. A convolutional neural network design consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers. Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution (although cross-correlation is a related operation). This only has significance for the indices in the matrix, and thus which weights are placed at which index. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation. Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer.[15] Another example is average pooling, which uses the average value from each of a cluster of neurons at the prior layer.Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.





6 SQUINS

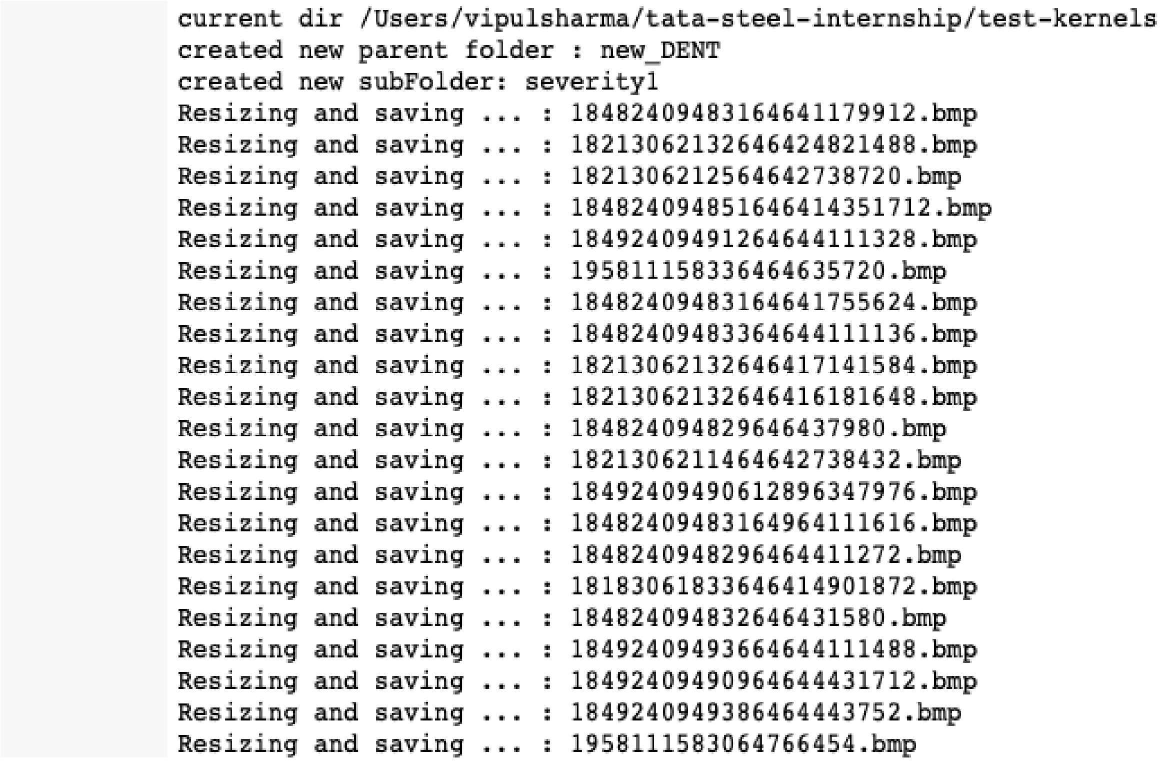
Figure 6.1: Surface Inspection System at CRM Bara

At Tata Steel, the need of the hour for all the manufacturing units is to maximize yield with minimum resources and produce "zero" rejects. When producing steel, a step towards the goal is to make the surface defects as minimum as possible for meeting the strictest quality specifications. Therefore, an inspection system that is reliable, efficient and accurate enough to detect all surface defects is required which enables root cause analysis and elimination, faster quality grading and reduced need for manual inspection. Tata Steel automation division thus offers an online vision product for detecting surface defects automatically and tracking product performance at every stage. Through the right combination of lighting, camera, detection and classification software, the system provides accurate real time surface inspection for various surface properties, defects and line speeds. The principals of operations are done with acquiring and processing images, detecting flaws in these images and segmenting them into various defects as described in the above section. The output is then converted into real time result display of severity detection up to size 1 millimeters. The image capturing is done with two CCD cameras both above and below the rolled steel which can be tuned based on the region of interests like exposure rate, sensitivity mode and illumination profiles. Other application of SQUINS include feature extraction, for the purpose of extracting relevant defect features for the purpose of classification, an offline classifier, a Plug-n-Play multiple operator stations, SQUINS player for offline analysis and visualization and a web based report generation engine to generate reports based on search criteria.

7 DATA SANITIZING

As discussed above there are twenty seven types of defects : Carbon Shots, Coil Mark Break, DENT, Edge Damage, Fold Mark, Gouge Mark, High Temperature Scale, Hole, Lamination, Length Wise Mark, Line Stoppage, Over pickle, Pickling Patch Type, Pitting, Pseudo, RIS, Roll Mark, Scratch, Secondary Scales, Silver, Squeeze Roll Mark, Stone Rubbing, Tongue Mark, Uncl, Underpickle, Watter Carry Over and Width Wise Mark and each of these defects are categorized into five different severity levels including : Severity 1, Severity 2, Severity 3, Severity 4 and Severity 5 in which 5th is the highest and 1st being the lowest. These images are of various sizes and we would be needing same size images to process in analyzing and further classifying the data. The packages used here are in Python. Therefore, we are now going to convert these images to equal sizes of 200 by 100 pixels.

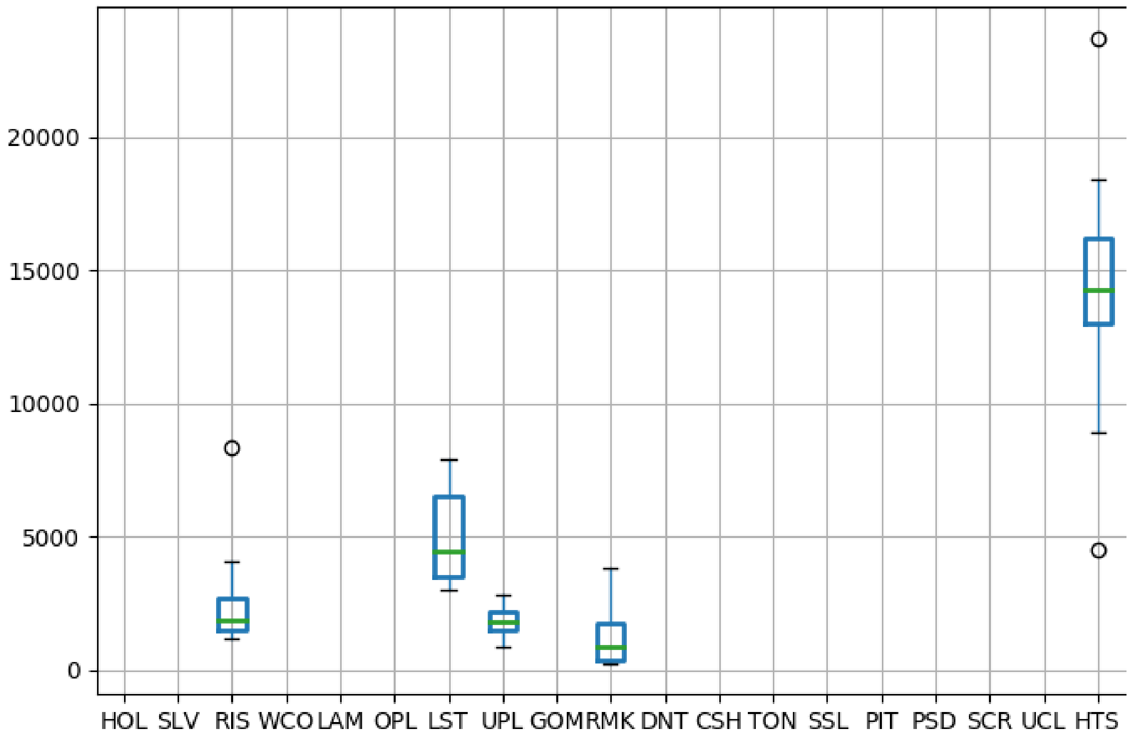
```
import os, sys
from os.path import dirname, basename
from PIL import Image
from resizeimage import resizeimage
from os import listdir


def checkSubFolder(newFolderPath):
    # check if sub folder exist inside the parent folder, if not then create
    if os.path.isdir(newFolderPath) == False:
        os.makedirs(newFolderPath)
        print("created new subFolder: {}".format(basename(newFolderPath)))
    return


def checkParentFolder(newFolderName):
    # check if parent folder exist, if not then create
    if os.path.isdir(newFolderName) == False:
        os.makedirs(newFolderName)
        print("created new parent folder : {}".format(basename(newFolderName)))
    return


def resizeAndSave(filepath, size, folderName, subFolderName, dest):
    newFolderName = dest + "/" + "new_" + folderName
    checkParentFolder(newFolderName)
    newFolderPath = newFolderName + "/" + subFolderName
    checkSubFolder(newFolderPath)
    destination = newFolderName + "/" + subFolderName
    # loop through each severityN sub folders
    # resize and save files to corresponding location
    for filename in os.listdir(filepath):
        if filename.endswith('.bmp'):
            with Image.open(filepath + "/" + filename) as image:
                convert = resizeimage.resize_cover(image, size)
                convert.save(destination + "/new_" + basename(filename), image.format)
                print("Resizing and saving ... : {} ".format(filename))
    return


def main(target, dest):
    if os.path.isdir(dest) == False:
        os.makedirs(dest)
    # loop throught all folders under current path
    for (dirpaths, dirs, files) in os.walk(target):
        # for subdir in subdirs:
        if not dirs:
            parentPath = dirname(dirpaths)  # get parent folder's name
            parentName = basename(parentPath)  # folder name like DENT
            subFolderName = basename(dirpaths)  # folder name like severity1
            resizeAndSave(dirpaths, [200, 100], parentName, subFolderName, dest)


if __name__ == '__main__':
    rootdir = os.getcwd()
    print("current dir {}".format(rootdir))
    # define the target folder that contains target files
    targetFolder = "../SQUINSDefectLibrary"
    # define the folder for saving new files
    destFolder = "../SQUINSDefectLibraryResized"
    main(targetFolder, destFolder)
```



```
current dir /Users/vipulsharma/tata-steel-internship/test-kernels
created new parent folder : new_DENT
created new subFolder: severity1
Resizing and saving ... : 18482409483164641179912.bmp
Resizing and saving ... : 18213062132646424821488.bmp
Resizing and saving ... : 1821306212564642738720.bmp
Resizing and saving ... : 184824094851646414351712.bmp
Resizing and saving ... : 18492409491264644111328.bmp
Resizing and saving ... : 195811158336464635720.bmp
Resizing and saving ... : 18482409483164641755624.bmp
Resizing and saving ... : 18482409483364644111136.bmp
Resizing and saving ... : 18213062132646417141584.bmp
Resizing and saving ... : 18213062132646416181648.bmp
Resizing and saving ... : 184824094829646437980.bmp
Resizing and saving ... : 1821306211464642738432.bmp
Resizing and saving ... : 18492409490612896347976.bmp
Resizing and saving ... : 18482409483164964111616.bmp
Resizing and saving ... : 1848240948296464411272.bmp
Resizing and saving ... : 18183061833646414901872.bmp
Resizing and saving ... : 184824094832646431580.bmp
Resizing and saving ... : 18492409493664644111488.bmp
Resizing and saving ... : 18492409490964644431712.bmp
Resizing and saving ... : 1849240949386464443752.bmp
Resizing and saving ... : 1958111583064766454.bmp
```

This enables us to have similar sized images in proper directories. We will now analyze the defects from the data available from the surface inspection system. From the barplot below we can see there are five types of defects that the barplots produced : RIS (median at 1900), LIS (median at 4500), UPL (median at 1760), RMK (median at 700) and HTS (median at 14100). We are now going to find edges for each of these defects based on their maximum severity level, that is, from severity level 5.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

df =  pd.read_csv('../coil-information-specific.csv')
boxprops = dict(linestyle='-', linewidth=2, color='k')
medianprops = dict(linestyle='-', linewidth=2, color='k')
ax= df.boxplot(boxprops=boxprops,medianprops=medianprops)
ax.set_xlabel('Defects')
ax.set_label('Defect Statistics ')
plt.figure(figsize = (15,15))
plt.show(block=True)
```



8 DEFECT LISTING

Based on the above plot we are now going to analyze each of these defects in Python. The defects are named as follows : High Temperature Scale, Line Stoppage, Rolled in Scale and Under pickle respectively. The density plots and RGB scales were taken out of each of these defects and the edges were found. The libraries used were from package named skimage. The library filters has module sobel which finds the edge magnitude using the Sobel transform. Below (Fig 6.1 and 6.2) are the RGB scaled picture with the density plots of each of these defects. (Fig 6.3) The Sobel edge map of each of these defects is also shown below these pictures to clarify that the defect is really present in the rolled steel.

The code is shown below.

```
if __name__ == ’__main__’:
    import imageio
    import matplotlib.pyplot as plt
    from skimage import data, io, filters
    %matplotlib inline

    # Assuming each picture as pic
    pic1 = imageio.imread(’/new_HighTempratureScale/severity5/1908210085264446004.bmp’)
    pic2 = imageio.imread(’~/new_LineStoppage/severity5/18272404274311848641521816.bmp’)
    pic3 = imageio.imread(’~/new_RIS/severity5/18432906431264641270112.bmp’)
```

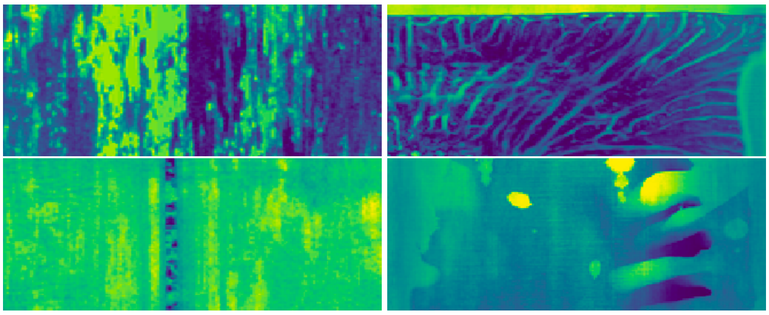


Figure 8.1: Top left to bottom right: High Temperature Scale, Line Stoppage, Rolled In Scale, Underpickle Defects

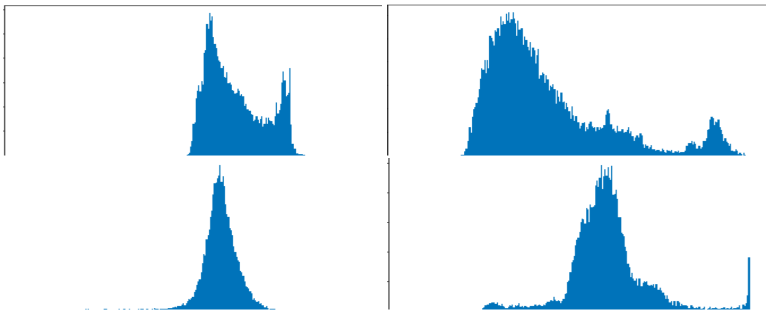


Figure 8.2: Top left to bottom right: Density plots High Temperature Scale, Line Stoppage, Rolled In Scale, Underpickle scaled between [0,250]

```
pic4 = imageio.imread(’~/new_Underpickle/severity5/190113110153137620407614.bmp’)

plt.figure(figsize = (15,15))
plt.imshow(pic)

# Dimensions and factors of each pic
print(’Type of the image : ’ , type(pic))
print(’Shape of the image : {}’.format(pic))
print(’Image Hight {}’.format(pic.shape[0]))
print(’Image Width {}’.format(pic.shape[1]))
print(’Dimension of Image {}’.format(pic.ndim))
print(’Maximum RGB value in this image {}’.format(pic.max()))
print(’Minimum RGB value in this image {}’.format(pic.min()))

plt.title(’All channels’)
plt.ylabel(’Height {}’.format(pic.shape[0]))
plt.xlabel(’Width {}’.format(pic.shape[1]))
plt.imshow(pic[ :])
plt.show()
```

```
#Density plots
plt.hist(pic.ravel(),256,[0,256])
plt.show()

# Finding edges
edges = filters.sobel(pic)
io.imshow(edges)
```

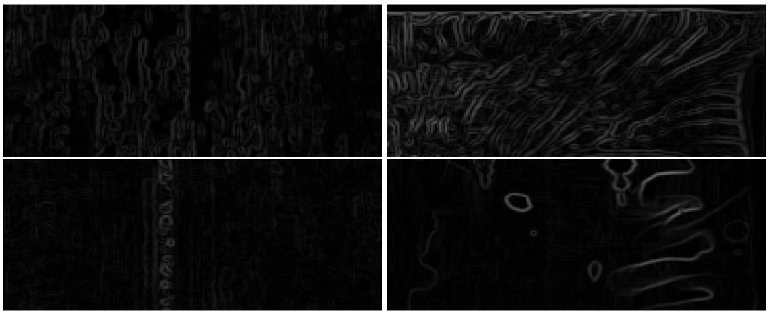


Figure 8.3: Top left to bottom right: High Temperature Scale, Line Stoppage, Rolled In Scale, Underpickle Edges

9 Convolution Auto Encoder

Convolutional AutoEncoders (CAEs) approach the filter definition task from a different perspective: instead of manually engineer convolutional filters we let the model learn the optimal filters that minimize the reconstruction error. These filters can then be used in any other computer vision task. CAEs are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.CAEs are a type of Convolutional Neural Networks (CNNs):

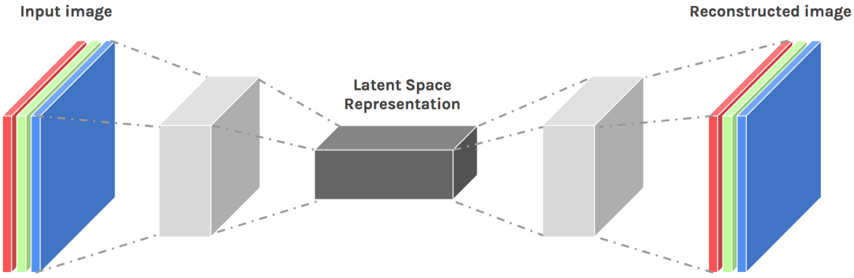


Figure 9.1: A basic autoencoder

the main difference between the common interpretation of CNN and CAE is that the former are trained end-to-end to learn filters and combine features with the aim of classifying their input. In fact, CNNs are usually referred as supervised learning algorithms. The latter, instead, are trained only to learn filters able to extract features that can be used to reconstruct the input. CAEs, due to their convolutional nature, scale well to realistic-sized high-dimensional images because the number of parameters required to produce an activation map is always the same, no matter what the size of the input is. Therefore, CAEs are general purpose feature extractors differently from AEs that completely ignore the 2D image structure. In fact, in AEs the image must be unrolled into a single vector and the network must be built following the constraint on the number of inputs. In other words, AEs introduce redundancy in the parameters, forcing each feature to be global (i.e., to span the entire visual field)1, while CAEs do not. Therefore, in general an autoencoder is a transformation unit, through which the input image is converted into a multidimensional feature image for feature extraction and representation. The acquired feature maps consists of the most important features. The decoder thus fine tunes the pixel level labels by merging the context information from the feature maps learned in the hidden layers. Moreover, the decoder network can use an up sampling operation to restore the final output to the same size as the input image.

10 Detection

Since the metallic surface defects are the local anomalies in the homogeneous texture, defects and background textures have different feature representations. We utilize the autoencoder network to learn the representation of defect data and find the common features of metallic surface defects. Therefore, the problem of metallic surface defect detection is turned into object segmentation problem. The input defect image is transformed to a pixel-wise prediction mask with the encoder-decoder architecture.

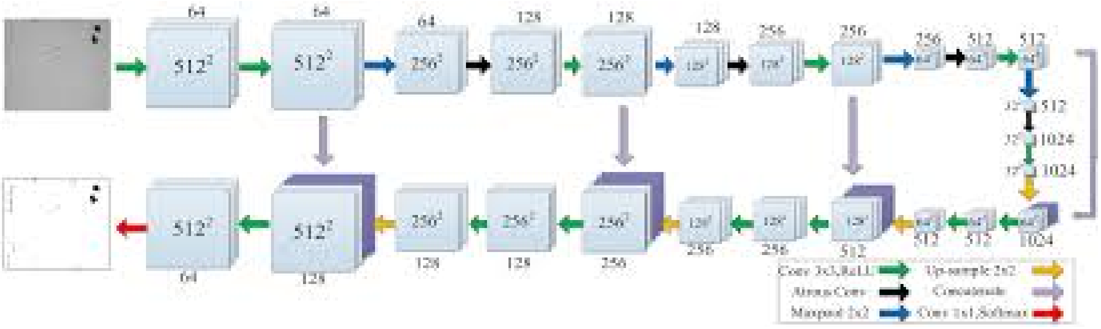
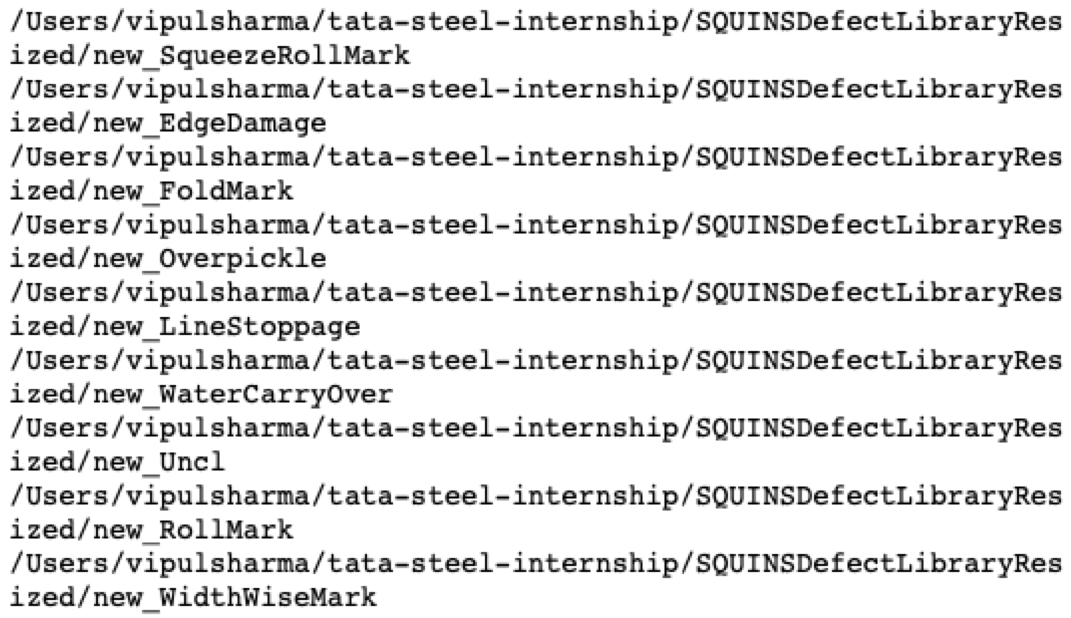


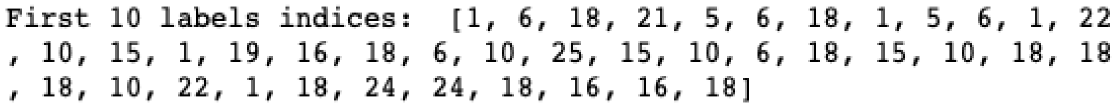
Figure 10.1: The architecture of the autoencoder network

The autoencoder structure shown in the Figure 10.1 has image as an input to the encoder which is towards the right of the picture. The encoder section contains 10 convolution layers, with each containing 3 x 3 convolution operations and subsequent rectified linear unit (ReLu) activation functions as non linear operations. Each of the two convolution layer is followed by 2 x 2 max pooling layer operation with stride 2. To reduce the loss of semantic information we double the number of features after each max pooling layer. The decoder is then applied with two convolution layer having a 2 x 2 sampling operation. The result is then concatenated to the corresponding feature map from the encoder section to obtain the final feature map. At the final layer, 1 x 1 convolution with a softmax layer us attached to the autoencoder network is applied to transform the output to a probability map which is then re sized to the same size of the input image. Two of the same architectures are applied in which the output of the first autoencoder is served as he input to the second to fine tune the pixel labels.

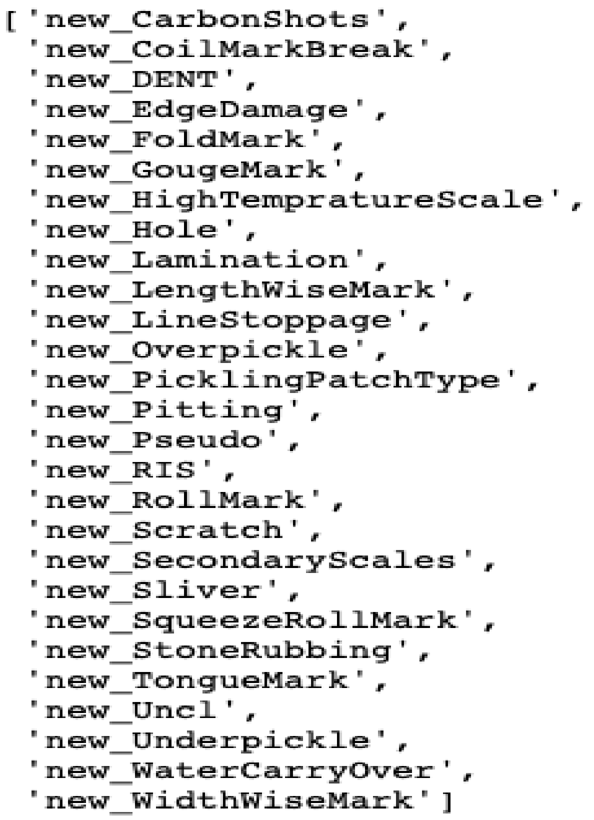
```
from __future__ import absolute_import, division, print_function

import os
import pathlib
import random
data_root = ('~/tata-steel-internship/SQUINSDefectLibraryResized')
data_root = pathlib.Path(data_root)
print(data_root)
for item in data_root.iterdir():
    print(item)

all_image_paths = list(data_root.glob('*/*/*'))
all_image_paths = [str(path) for path in all_image_paths]
random.shuffle(all_image_paths)
image_count = len(all_image_paths)
image_count

all_image_paths[:10]

label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
label_names

label_to_index = dict((name, index) for index,name in enumerate(label_names))
label_to_index

all_image_labels = [label_to_index[pathlib.Path(path).parent.parent.name]
                    for path in all_image_paths]
print("First 40 labels indices: ", all_image_labels[:40])
```



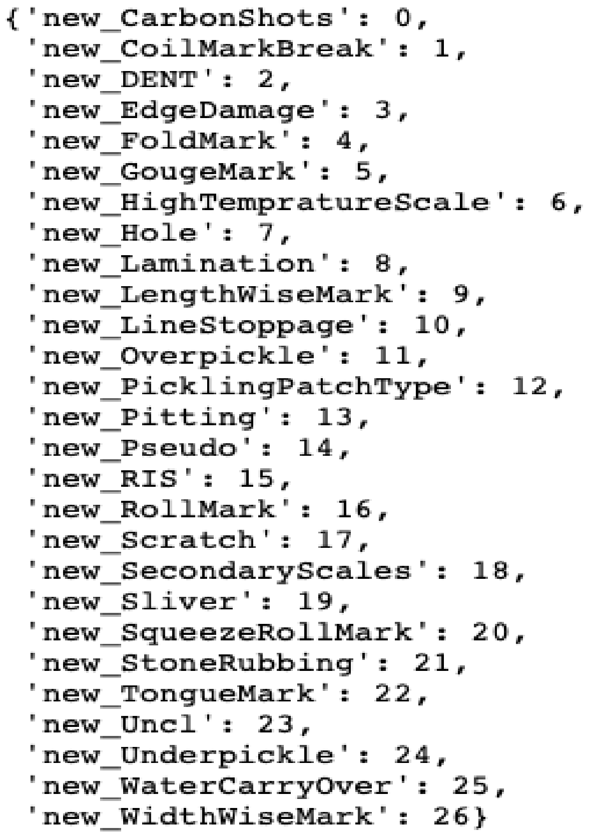
```
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_SqueezeRollMark
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_EdgeDamage
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_FoldMark
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_Overpickle
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_LineStoppage
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_WaterCarryOver
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_Uncl
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_RollMark
/Users/vipulsharma/tata-steel-internship/SQUINSDefectLibraryRes
ized/new_WidthWiseMark
```



```
First 10 labels indices:  [1, 6, 18, 21, 5, 6, 18, 1, 5, 6, 1, 22
, 10, 15, 1, 19, 16, 18, 6, 10, 25, 15, 10, 6, 18, 15, 10, 18, 18
, 18, 10, 22, 1, 18, 24, 24, 18, 16, 16, 18]
```



```
['new_CarbonShots',
 'new_CoilMarkBreak',
 'new_DENT',
 'new_EdgeDamage',
 'new_FoldMark',
 'new_GougeMark',
 'new_HighTempratureScale',
 'new_Hole',
 'new_Lamination',
 'new_LengthWiseMark',
 'new_LineStoppage',
 'new_Overpickle',
 'new_PicklingPatchType',
 'new_Pitting',
 'new_Pseudo',
 'new_RIS',
 'new_RollMark',
 'new_Scratch',
 'new_SecondaryScales',
 'new_Sliver',
 'new_SqueezeRollMark',
 'new_StoneRubbing',
 'new_TongueMark',
 'new_Uncl',
 'new_Underpickle',
 'new_WaterCarryOver',
 'new_WidthWiseMark']
```



```
{'new_CarbonShots': 0,
 'new_CoilMarkBreak': 1,
 'new_DENT': 2,
 'new_EdgeDamage': 3,
 'new_FoldMark': 4,
 'new_GougeMark': 5,
 'new_HighTempratureScale': 6,
 'new_Hole': 7,
 'new_Lamination': 8,
 'new_LengthWiseMark': 9,
 'new_LineStoppage': 10,
 'new_Overpickle': 11,
 'new_PicklingPatchType': 12,
 'new_Pitting': 13,
 'new_Pseudo': 14,
 'new_RIS': 15,
 'new_RollMark': 16,
 'new_Scratch': 17,
 'new_SecondaryScales': 18,
 'new_Sliver': 19,
 'new_SqueezeRollMark': 20,
 'new_StoneRubbing': 21,
 'new_TongueMark': 22,
 'new_Uncl': 23,
 'new_Underpickle': 24,
 'new_WaterCarryOver': 25,
 'new_WidthWiseMark': 26}
```

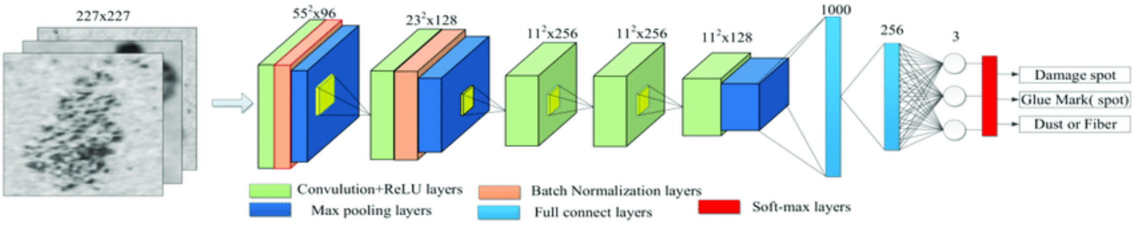
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.contrib.layers import fully_connected

tf.enable_eager_execution()

X_train = all_image_paths[:12896]
Y_train = all_image_labels[:12896]
X_test = all_image_paths[12897:]
Y_test = all_image_paths[12896:]

max_train_idx = len(X_train)
max_test_idx = len(X_test)

num_inputs = 20000
num_hid1 = 10000
num_hid2 = 5000
num_hid3 = num_hid1
num_ouputs = num_inputs
lr = 0.01
actf = tf.nn.relu

def next_batch(batch_size):
    X_batch = []
    Y_batch = []
    for i in range(batch_size):
        X_batch.append(random.choice(X_train))
    Y_batch.append([label_to_index[pathlib.Path(path).parent.parent.name]
                    for path in X_batch])
    return X_batch, Y_batch

X=tf.placeholder(tf.float32,shape=[None,num_inputs])

initializer = tf.variance_scaling_initializer()
w1=tf.Variable(initializer([num_inputs,num_hid1]),dtype=tf.float32)
w2=tf.Variable(initializer([num_hid1,num_hid2]),dtype=tf.float32)
w3=tf.Variable(initializer([num_hid2,num_hid3]),dtype=tf.float32)
w4=tf.Variable(initializer([num_hid3,num_output]),dtype=tf.float32)

b1=tf.Variable(tf.zeros(num_hid1))
b2=tf.Variable(tf.zeros(num_hid2))
```

```
b3=tf.Variable(tf.zeros(num_hid3))
b4=tf.Variable(tf.zeros(num_output))

hid_layer1=actf(tf.matmul(X,w1)+b1)
hid_layer2=actf(tf.matmul(hid_layer1,w2)+b2)
hid_layer3=actf(tf.matmul(hid_layer2,w3)+b3)
output_layer=actf(tf.matmul(hid_layer3,w4)+b4)
loss=tf.reduce_mean(tf.square(output_layer-X))

optimizer=tf.train.AdamOptimizer(lr)
train=optimizer.minimize(loss)

init=tf.global_variables_initializer()

num_epoch=100
batch_size=500

with tf.Session() as sess:
    sess.run(init)
    for epoch in range(num_epoch):

        num_batches=mnist.train.max_train_idx//batch_size
        for iteration in range(num_batches):
            X_batch,y_batch=next_batch(batch_size)
            sess.run(train,feed_dict={X:X_batch})

        train_loss=loss.eval(feed_dict={X:X_batch})
        print("epoch {} loss {}".format(epoch,train_loss))


results=output_layer.eval(feed_dict={X:X_test})
```

The autoencoder should consist of a threshold module at the end as an independent module. It has a threshold assigned to final prediction mask and gives out a value of 0 when the final image after binarization is less than or equal to the threshold and a value of 1 otherwise. The defect regions are further accounted for accurate defect countours using blob analysis.

11 CLASSIFICATION

The defect regions are classified into categories during the classification. The usual convolution neural network will take in the gray scale images with an output size of 227 x 227. The inputs are then transferred to the first convolution layer with kernel size 11 x 11, stride equal to 4 giving an output size of 55 x 55 x 96. It is then passed through a max pooling layer of kernel size 3 x 3, stride of 2 producing an output of 27 x 27 x 96. The output from the first layers are then transferred to the second convolution layer with kernel size 5 x 5, stride equal to 1 giving an output size of 23 x 23 x 128. It is then passed through a max pooling layer of kernel size 3 x 3, stride of 5 producing an output of 11 x 11 x 128. The output is then passed into three similar convolution layer of kernel size 3 x3, stride of 1 and padding equal to 1 producing an output of 11 x 11 x 256 (for 2 layers) and 11 x 11 x 128 (for the last layer). Then comes the max pooling layer of kernel size 3 x 3 and stride 2 to give the output 5 x5 x 128. All the above outputs are then transferred into two fully connected layers with kernel size 1000 and 256 respectively to classify the defected regions.



```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import numpy as np

data_root = ('~/tata-steel-internship/SQUINSDefectLibraryResized')
data_root = pathlib.Path(data_root)
all_image_paths = list(data_root.glob('*/*/*'))
all_image_paths = [str(path) for path in all_image_paths]
random.shuffle(all_image_paths)
image_count = len(all_image_paths)
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
label_to_index = dict((name, index) for index,name in enumerate(label_names))
all_image_labels = [label_to_index[pathlib.Path(path).parent.parent.name]
                    for path in all_image_paths]

batch_size = 1000
num_classes = 27
epochs = 50

img_rows, img_cols = 227, 227

X_train = all_image_paths[:12896]
Y_train = all_image_labels[:12896]
X_test = all_image_paths[12897:]
Y_test = all_image_paths[12896:]

y_train = keras.utils.to_categorical(Y_train, num_classes)
```

```
y_test = keras.utils.to_categorical(Y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(11, 11),
                 activation='relu',
                 input_shape=(227,227,1)))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Conv2D(25, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Conv2D(9, (3, 3), activation='relu'))
model.add(Conv2D(9, (3, 3), activation='relu'))
model.add(Conv2D(9, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

12 CONCLUSION

This paper shows thee base architecture of how surface inspection system can work using neural network based on the autoencoder and the CNN architecture. Through this paper the model can be transformed into various other language such as C and be used in the system to get a better analysis of the classification. The way forward is to experiment this code on real time data to check the autoencoder detection algorithm and to see if it is sufficient enough to meet the requirements of the complex industrial environment.

REFERENCES

[1] Tata Steel
`https://www.tatasteel.com/investors/integrated-reportannual-report/`

[2] Cold Rolled Steel
`https://www.tatasteeleurope.com/en/products/engineering/cold-rolled`

[3] A basic Introduction to Neural Network
`http://pages.cs.wisc.edu/ bolo/shipyard/neural/local.html`

[4] Convolutional Neural Network
`https://en.wikipedia.org/wiki/Convolutionalneuralnetwork`

[5] Surface Inspection System in Tata Steel
`http://www.automationtatasteel.com/brochures/squins.pdf`

[6] Xian Tao , Dapeng Zhang, Wenzhi Ma, Xilong Liu and De Xu. *Automatic Mettalic Surface Defect Detection and Recognition with Convolutionsal Neural Network*. Published: 6 September, 2018.