A Project Report on:

"Classification of Welding Defects in Radiographic Images"

Submitted by:

N Janavi (181EC127)

Chandravaran K (181EC156)

V SEM BTECH (ECE)

Under the guidance of

Prof. Deepu Vijayasenan

Department of ECE, NITK Surathkal

in partial fulfilment for the award of the degree
of
Bachelor of Technology
in
Electronics and Communications
at

National Institute of Technology Karnataka, Surathkal

November 2020

Table of Contents

1) Abstract

This report describes the work that was carried out in trying to classify unlabeled welding defects in radiographic industry images, we start the report with some introduction to the concepts used throughout the project. After explaining some of the related work to this project we move on to explain about the problems we tackled and the analysis of the results which were obtained through the project. We finally end the report with the results obtained after running the images through a pretrained model and classifying them with respective sample label images.

2) Introduction

DATASET and what's in it?
T Radiography is one of the oldest techniques of Non Destructive Testing (NDT) inspection; it is still accepted as necessary for the control of welded joints in many industries such as the nuclear, oil piping, and aeronautical. The radiographic welded joint interpretation is a complex problem requiring expert knowledge because of the heterogeneity and the defect's nature, morphology, position (superficial or internal location), orientation and size. The dataset we received had a total of 3900 radiographic images of welding. Which included images of 30 different types of defects which had to be preprocessed and classified. The images varied in size from 187- 2366 along the high and 1025-6775 along the length, so it was very important to bring the images to equal sizes without loss of generality.

The main objective of this project is to take the raw data of X-rays images from the industry and classify them based on their type of defect. We are going to deal with around 30 types of defects. We received a dataset of radiographic images which were unlabelled and all of them consisting of 3,4 and 5 images stacked vertically. We needed a generic method that had to be incorporated to split these stacked images into its individual parts and the same was decided to be carried out using the Hough Transform and Morphology techniques. The latter yields better results in determining the horizontal lines which were used as a reference. The data images were then split accordingly, as the images were equally spaced.

This was followed by feature extraction, an important step in Image segmentation.Feature extraction is a type of dimensionality reduction where a large number of pixels of the image are efficiently represented in such a way that interesting parts of the image are captured effectively. We decided to go with pre trained CNN models for this task because of time constraints.Transfer learning is the most popular approach in deep learning. In this, we use pre-trained models as the starting point in computer vision.Choosing the optimal pre trained model is important as our problem should be comparable to the problem the model was trained on to get better predictions. We decided to run our dataset with Resnet50 and Inception_v3 models.

3) Related work

Wang at el used an automatic computer-aided identification system to recognize different types of welding defects in radiographic images. To separate defects from the background, image-processing techniques such as background subtraction and histogram thresholding were used. To represent each defect, 12 numeric features were taken out or extracted. 2 well known classifiers which are fuzzy k-nearest neighbors and multi-layer perceptron neural network classifiers were used to classify the extracted feature values. Their performances are tested and compared using the bootstrap method.

Moghaddam, Alireza Azari in their paper had developed their own algorithm for classifying 3 defects in welding images namely Lack of Penetration(LOC), Effect undercut(EUC) and Incomplete fusion(IF) and compared it with widely used classifiers SVM and Fuzzy k-nearest neighbor. It proved to be better at classification than the other two. The started with thresholding for image segmentation and extracted the features of the segmented images to run through their algorithm.The results showed superior accuracy.

4) Methodology

Analysing Data

The first task that was given to us was the analysis of the data. It comprised of 3.9k images which had radiographic images which varied in dimensions, so we found the size of the each image and placed the the sizes into an excel sheet named “image_size_file” that has been submitted along with this report, the following code snip was used:

```
#finding size code
images_D = 'F:\\IP_mini_project\\RT Training Images DGI'
dic = collections.defaultdict(int)
c=0

l=[]

for f in os.listdir(images_D):
  i = cv2.imread(os.path.join(images_D, f))
  d = i.shape
  dic[d] += 1
  l.append([d,f])
  c+=1
```

Code snippet: To find size of all Images

After writing the sizes into the excel sheet, we had observed that in each image the number of X-rays that were stacked ontop of each other was different so we took a look at the height of the images and plotted them into a histogram and we observed 3 bins with which has the most number of images, at this point we thought each large ment the first started with 2 image and second one 3 and so on. To see if we were right we split the bins among yourself to see how each batch of images was, and we found out that each batch was consistent with the number of images stacked ontop of each other.

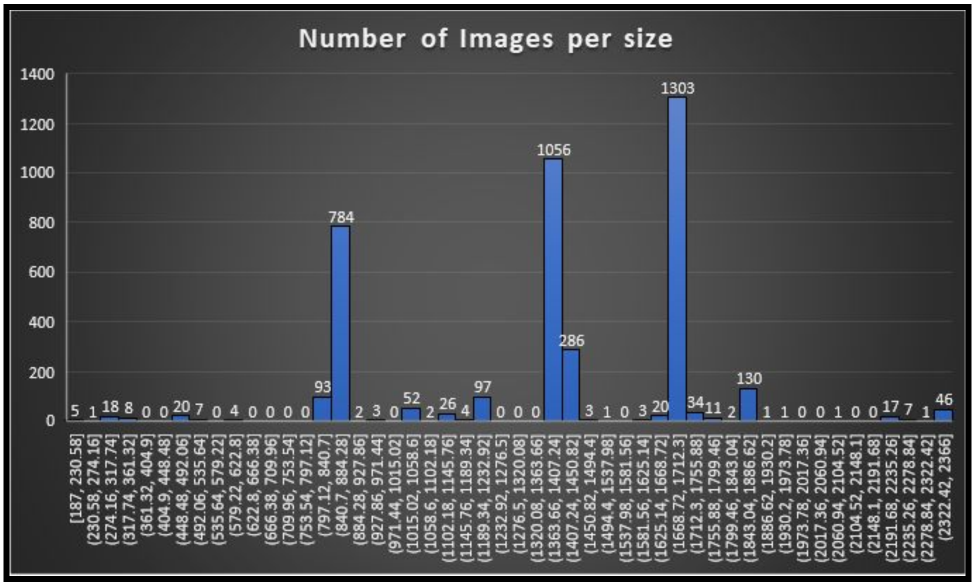


Fig: Plotted bins on Excel

The below code snip is what was used to view each bin and to analyse it

```
def images(c):
   l=[]
   b=0
   if c[0]==840.7 or c[0]==1363.66 or c[0]==1668.72 or c[0]==1407.24 :
      r=21
   else:
      r=5
   for f in os.listdir(images_D):
      i = cv2.imread(os.path.join(images_D, f))
      d = i.shape
      if (c[0]<=d[0]) and (d[0]<=c[1]):
         if(b<r):
            b+=1
            l.append(i)
         else:
            break
   return l
```

Code snippet: To find Analyse Images in each Bin

```
c=[1363.66,1407.24]
a=43.58
l1=images(c)
for i in l1:
   plt.imshow(i)
   plt.title(c)
   plt.show()
c[0]+=a
c[1]+=a
```

Code snippet: To Move to next Bin

The first is a function that was called each time when we started at a reference bin location and went on incrementing the bin limits, and we found images in that range and then appended there values into an list and then displayed them. There is one if condition that is present before the for loop that was used so that we skip the bins in which there were no images present.

Our observations were kept into an excel sheet against the bins and the number of images stacked on top was written down which can be found in “number_x_rays”. There max 1,2 irregularities within the bin which could be neglected, as the data set was huge.

Splitting images

Now The task at hand was the splitting of the images, now the doubt was whether the images were equally spaced throughout the height or were they unequally spaced, we tried out 2 algorithms to find horizontal lines, and as they would run across the image, the horizontal lines found had to be big. The 2 methods used were Hough transform and Morphology For example

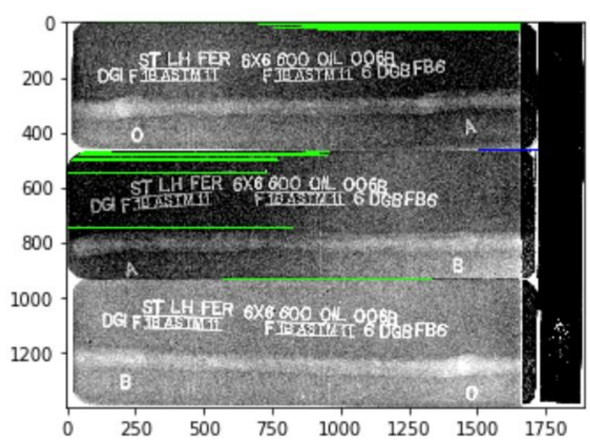


Fig:Image of finding horizontal lines

After this we observed that many of the stacked images were equally spaced, only very few of them were weirdly aligned, this could be removed as we had 3.9K images.

The code used to split the images, initially we coded 2 functions, the first one would find image which was part of the bin store it and pass onto the next function which was splitting the images, but we combined them into one function, as passing the list to another function was taking up too much memory space and becoming slow. We also stored the number of images present in each bin in a list and also the number of images stacked on top of each other from “number_x_rays”, these were passed on to so the function knows were to cut the image.

```
def images(c,index):
    l=[]
    b=0
    for f in os.listdir(images_D):
        if(b<=num_i[index]):
            img = cv2.imread(os.path.join(images_D, f))
            d = img.shape
            if (c[0]<=d[0]) and (d[0]<=c[1]):
                #l.append([i,f])
                d1=int(d[0]/xrays[index])
                dashes=[]
                for i in range(1,xrays[index]):
                    dashes.append(img[i*d1,:,:])
                # check where to split the picture and store that information
                splits = [0]
                for i in range(img.shape[0]):
                    for j in dashes:
                        if np.allclose(img[i,:,:], j):
                            splits.append(i)
                splits.append(img.shape[0])
                # write each cropped picture to your desired directory
                print(f)
                e=len(f)
                f1=f[:e-4]
                for j in range(len(splits)-1):
                    new_img = img[splits[j]:splits[j+1],:]
                    cv2.imwrite(save_file+f1+"_"+str(j)+".jpg", new_img)
                b+=1
```

Code snippet: To Split the Images

A list of the bin limits were also made, and this was used to check if an image was present in the bin and use the data was passed along with it.

Here are some examples of before split and after split pictures

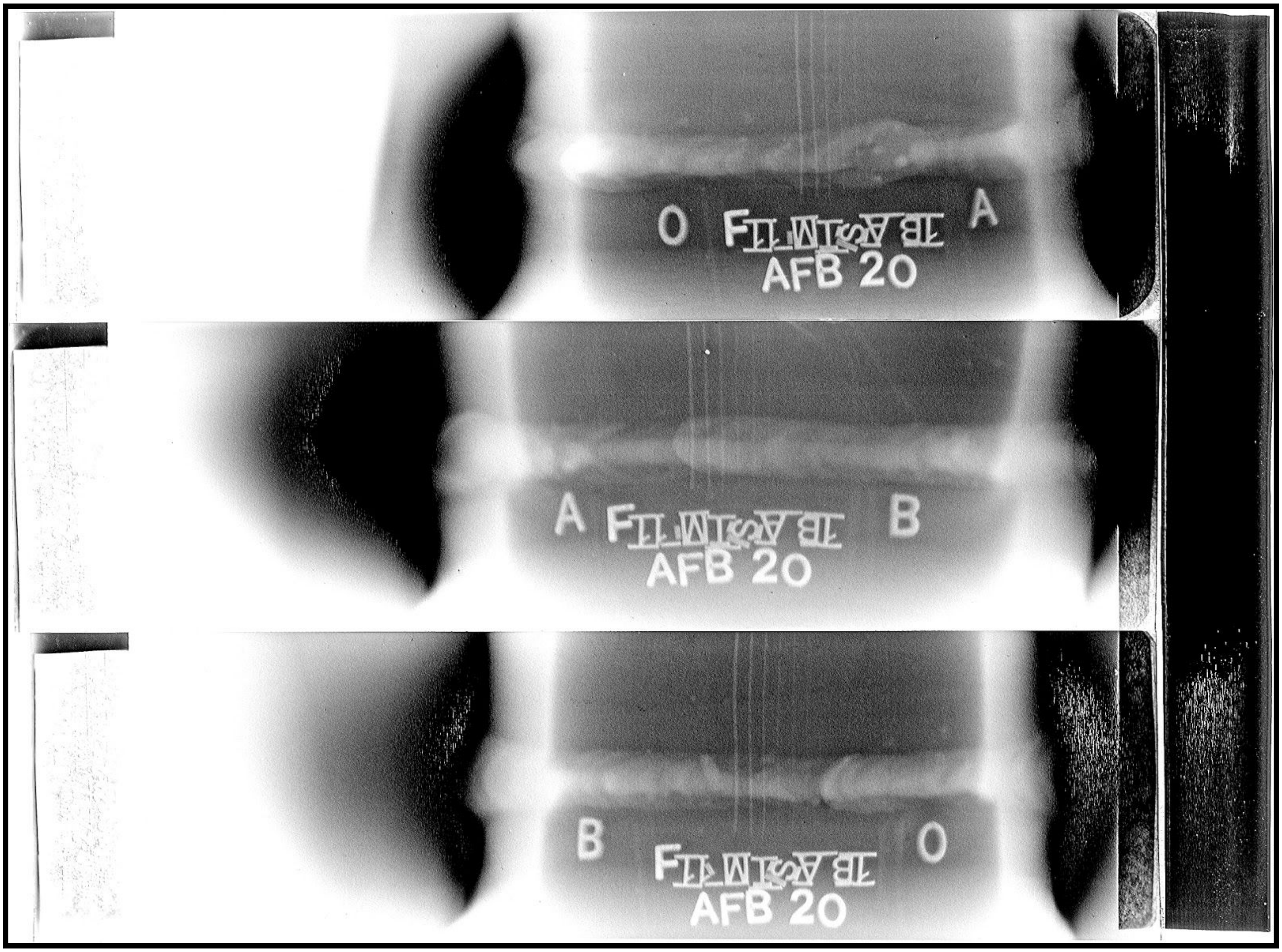


Fig:Image before splitting

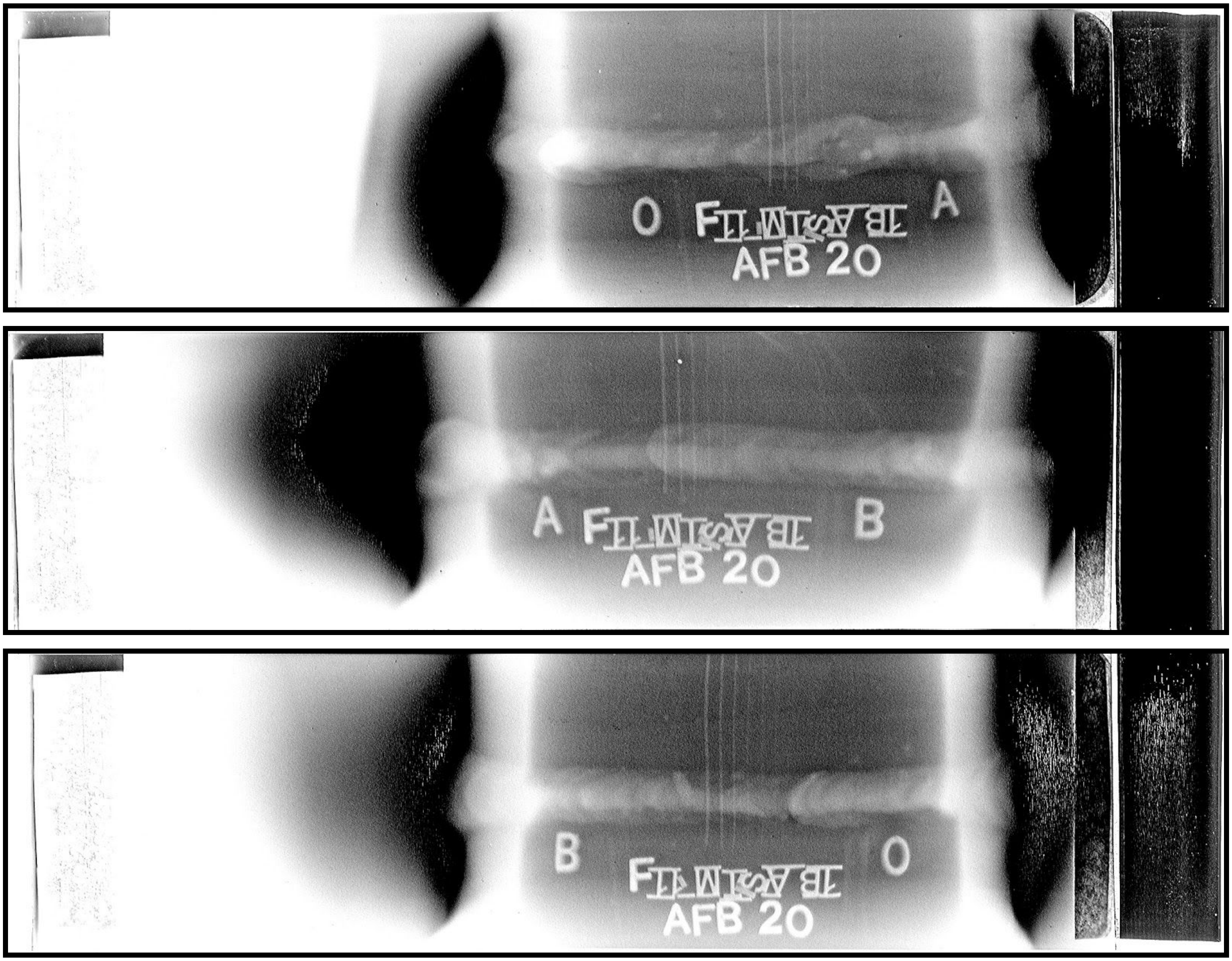


Fig: Images split into its parts

Resizing Images

Ideally the next stage of the project would be to run the images through a CNN and start our classification, the problem we faced here is that the images we had were not labeled data, so we decided until we received the labeled data we would try to classify the images our self and this is where transfer learning starts.

So we looked at all the models that were already present on the tensorflow hub, we needed models that gave us feature vectors, we determined to use 2 pretrained models, the Resnet50 and the Inception_V3.But what we observed was that the images that needed to be used for these models were 224*224 and 299*299 for the ResNet_50 and Inception_V3 respectfully, but the image we had present were about 700*1500 and these were too big to fit into the models. Initially we decided to resize the images directly, which was a very bad idea, as it can be observed in the picture below which is an image that has been resized to 299*299. This clearly was not good enough as we were dealing with images that had small defects and they would be lost.

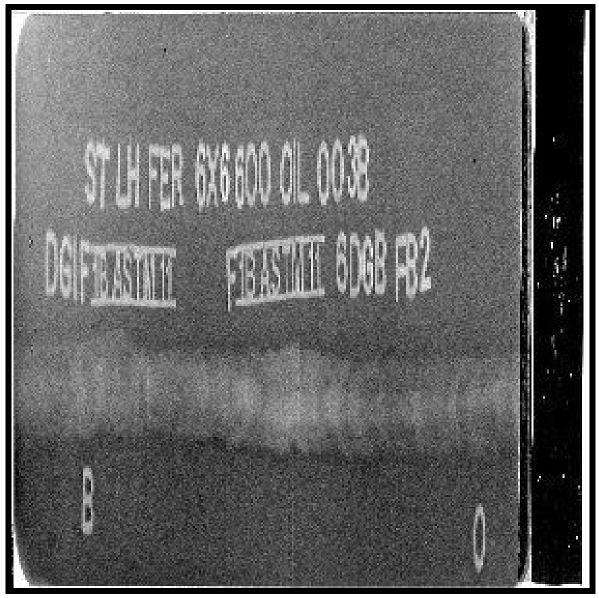


Fig: Resized Image

So what we decided to do was cut the image around the center, as the sample images given to us of the various defects it could be seen the defects were in the middle. We cut the images resized them into 2 different sizes one was 299*299 and the other was 299*500, this was done to see if taking a different size would give us a different result as there will be more information in the larger image. Below you will see the image of a cut image and then resized.

Fig: cut images of dimensions same height x 800/600/500

The images were cut according to their original widths and the height, if the width was too big we cut it into size 800, and it was smaller than 600 and even smaller 500. The code below shows that images were cut to 800, 600, 500 dimensions of width.These were then resized without to dimensions 299*299 and 299*500 for the next step of feature extractions. This method was employed so that the maximum features could be in the limited size.

Fig: Resized_299_299 Image

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
images_D =r"C:\Users\JANAVI N\Downloads\Images_IP_project\100sample_resized"
save_file= r"C:\Users\JANAVI N\Downloads\Images_IP_project\100sample_cut\\"
l=[]
splits=[]
b=0
for f in os.listdir(images_D):
    img = cv2.imread(os.path.join(images_D, f))
   d = img.shape
    print(d)
    h=int(d[0]/2)
    w=int(d[1]/2)      # write each cropped picture to your desired directory
    splits.append(img.shape[1]/2)              #write each cropped picture to your desired directory
    if w==1465:
        new_img = img[:,w-400:w+400]
        cv2.imwrite(save_file+f+"_"+".jpg", new_img)
    if w>300 and w<400:
        new_img = img[:,w-300:w+300]
        cv2.imwrite(save_file+f+"_"+str(j)+".jpg", new_img)
    if w==261:
        new_img = img[:,w-250:w+250]
        cv2.imwrite(save_file+f+"_"+str(j)+".jpg", new_img)
```

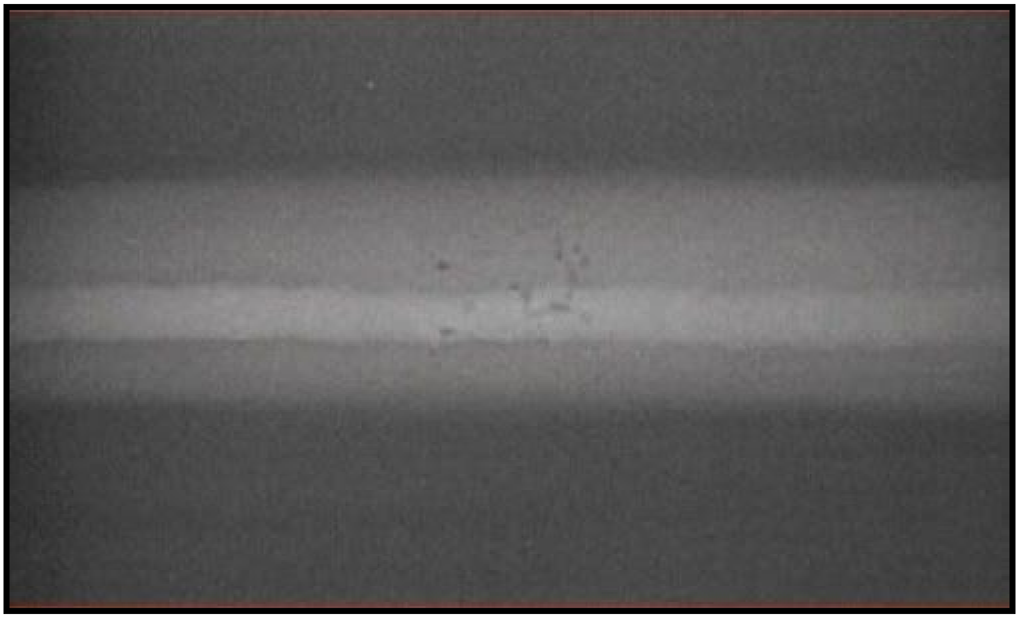


Fig: Resized_299_500 Image

Feature extraction

Initially we ran it through Resnet_50 and the architecture is explained below:

ResNet50 is a variant of the ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations. With a total of 50 layers, you can load a pre-trained version of the network trained on more than a million images called ImageNet database.

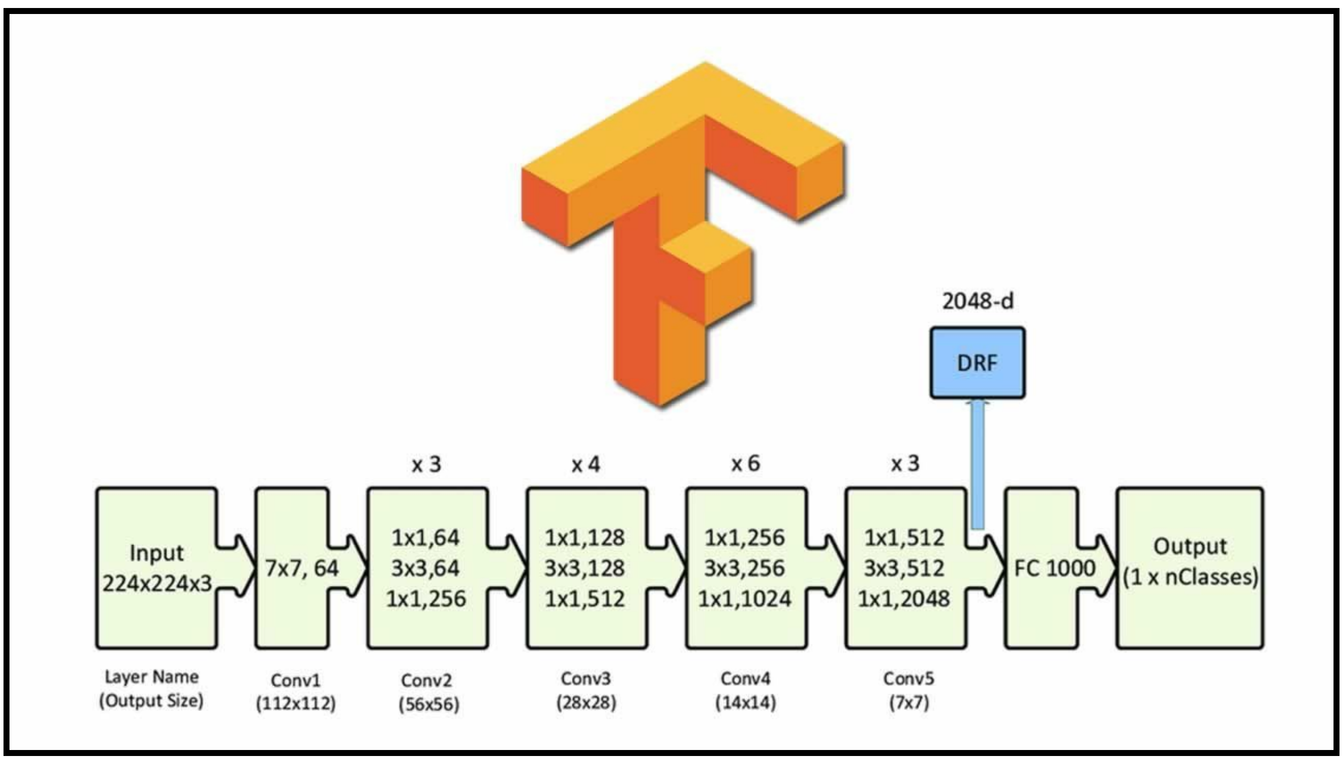


Fig:Resnet50 Architecture

The architecture: Every ResNet architecture performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively.Each 2-layer block is replaced in the 34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet. There are 4 stages in resnet50. The 1st stage has 3 residual blocks ,2nd stage has 4 residual blocks, the 3rd stage six and the fourth has 3 blocks. Each residual block has 3 layers each,As we go on to the next stages the channel width is doubled and the size of the input is reduced by half.

For each residual function F, 3 layers are stacked one over the other. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions. Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).

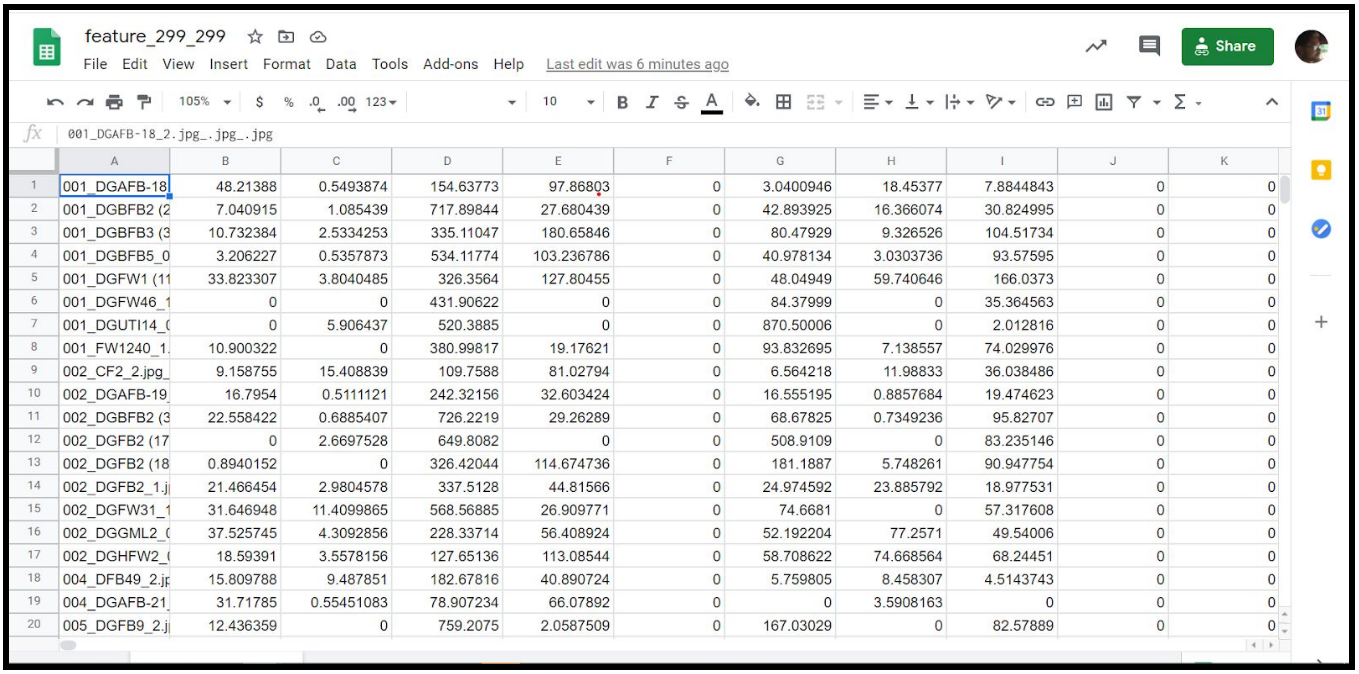
The code used for the feature extraction is as follows:

```
import numpy as np
import os
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_hub as hub
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input

images_D = 'F:\\IP_mini_project\\299x299_60img'
l=[]

url='https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/4'

base_model = hub.KerasLayer(url,input_shape=(299,299,3))

base_model.trainable = False

model = keras.Sequential([base_model])

for f in os.listdir(images_D):
    img = cv2.imread(os.path.join(images_D, f))
    img_data = image.img_to_array(img)
    img_data = np.expand_dims(img_data, axis=0)
    img_data = preprocess_input(img_data)
    feature = model.predict(img_data)
    l.append(feature)
print("completed")
```

Code snippet: To get Features

So the first thing to have to classify an image is having proper classes, so we used the 30 defects that were provided to us as defects along with about 30-40 Non defects taken from the data sets. The feature vectors were found for these and this can be done by taking one image at a time and passing it through the model and predicting the output of the model, as this does not have a final Softmax layer the output will be feature vector these are now stored in an excel sheet called “feature_299_299” .

Below is the picture of the first 10 feature vectors of the first 20 images:



	A	B	C	D	E	F	G	H	I	J	K
1	001_DGAFB-18	48.21388	0.5493874	154.63773	97.86803	0	3.0400946	18.45377	7.8844843	0	0
2	001_DGBFB2 (2	7.040915	1.085439	717.89844	27.680439	0	42.893925	16.366074	30.824995	0	0
3	001_DGBFB3 (3	10.732384	2.5334253	335.11047	180.65846	0	80.47929	9.326526	104.51734	0	0
4	001_DGBFB5_0	3.206227	0.5357873	534.11774	103.236786	0	40.978134	3.0303736	93.57595	0	0
5	001_DGFW1 (11	33.823307	3.8040485	326.3564	127.80455	0	48.04949	59.740646	166.0373	0	0
6	001_DGFW46_1	0	0	431.90622	0	0	84.37999	0	35.364563	0	0
7	001_DGUTI14_(	0	5.906437	520.3885	0	0	870.50006	0	2.012816	0	0
8	001_FW1240_1.	10.900322	0	380.99817	19.17621	0	93.832695	7.138557	74.029976	0	0
9	002_CF2_2.jpg_	9.158755	15.408839	109.7588	81.02794	0	6.564218	11.98833	36.038486	0	0
10	002_DGAFB-19	16.7954	0.5111121	242.32156	32.603424	0	16.555195	0.8857684	19.474623	0	0
11	002_DGBFB2 (3	22.558422	0.6885407	726.2219	29.26289	0	68.67825	0.7349236	95.82707	0	0
12	002_DGFB2 (17	0	2.6697528	649.8082	0	0	508.9109	0	83.235146	0	0
13	002_DGFB2 (18	0.8940152	0	326.42044	114.674736	0	181.1887	5.748261	90.947754	0	0
14	002_DGFB2_1.j	21.466454	2.9804578	337.5128	44.81566	0	24.974592	23.885792	18.977531	0	0
15	002_DGFW31_1	31.646948	11.4099865	568.56885	26.909771	0	74.6681	0	57.317608	0	0
16	002_DGGML2_(	37.525745	4.3092856	228.33714	56.408924	0	52.192204	77.2571	49.54006	0	0
17	002_DGHFW2_(	18.59391	3.5578156	127.65136	113.08544	0	58.708622	74.668564	68.24451	0	0
18	004_DFB49_2.jp	15.809788	9.487851	182.67816	40.890724	0	5.759805	8.458307	4.5143743	0	0
19	004_DGAFB-21	31.71785	0.55451083	78.907234	66.07892	0	0	3.5908163	0	0	0
20	005_DGFB9_2.j	12.436359	0	759.2075	2.0587509	0	167.03029	0	82.57889	0	0

Fig: Feature vectors sample labels 299_299-Resnet50

Next objective would be to find the feature vectors of the images that we cut into the size 299*500 the only parts of the code that change are shown below, similarly these feature vectors are stored in excel sheet name “feature_299_500”

```
images_D = 'F:\\IP_mini_project\\300x500_cut'
base_model = hub.KerasLayer(url,input_shape=(299,500,3))
```

feature_299_500

001_DGAFB-18_2.jpg_.jpg_0.jpg_.jpg

	A	B	C	D	E	F	G	H	I	J	K
1	001_DGAFB-18	18.260565	0.07803211	119.88579	2.4165065	0	6.052761	0.5165123	1.8801043	0	0
2	001_DGBFB2 (2	3.3637366	0.5704044	226.6635	37.43845	0	71.727104	3.2598872	16.282558	0	0
3	001_DGBFB3 (3	8.767302	0.49612004	101.08006	20.46378	0	23.715328	1.1390226	0.23499303	0	0
4	001_DGBFB5_0	0	0.8862525	212.61984	4.9607406	0	13.130216	0	12.541077	0	0
5	001_DGFW1 (11	23.672249	0.3261482	91.20211	70.82673	0	3.5192409	18.129025	0	0	0
6	001_DGFW46_1	0	0	314.02975	0	0	182.52838	0	13.322682	0	0
7	001_DGUTI14_(	0	0	372.15942	0	0	1055.4504	0	10.2549	0	0
8	001_FW1240_1.	10.664259	1.7887144	72.54421	30.739517	0	3.5150707	35.83175	0	0	0
9	002_CF2_2.jpg_	21.028048	5.9590855	36.62054	48.234924	0	3.7366893	4.212622	0	0	0
10	002_DGAFB-19	10.990091	0	185.88242	0	0	49.06854	0	5.691861	0	0
11	002_DGBFB2 (3	3.9375331	0.7529628	491.294	8.613635	0	88.11343	1.537493	7.8036966	0	0
12	002_DGBFB4_2	5.8651977	3.4034874	179.00461	0	0	84.86194	0	47.452976	0	0
13	002_DGFB2 (17	0	0	432.36426	0	0	625.60736	0	9.047043	0	0
14	002_DGFB2 (18	4.4492307	0	229.01411	12.290679	0	227.65088	0	23.81723	0	0
15	002_DGFB2_1.j	15.844808	0.40027267	38.518124	62.61123	0	1.9329132	10.03727	0	0	0
16	002_DGFW2_2.	0	0	111.613525	0	0	366.29776	0	3.9846191	0	0
17	002_DGFW31_1	17.353525	3.461202	94.11156	30.335693	0	83.15044	0.18546502	7.130754	0	0
18	002_DGGML2_(	5.0568547	3.877355	62.73819	91.64959	0	2.654386	85.76422	2.9968066	0	0
19	002_DGHFW2_(	5.45576	0.4088467	88.79514	93.80892	0	4.8389387	69.07946	1.2863514	0	0
20	004_DFB49_2.jp	33.811954	6.554254	71.341805	49.6163	0	4.9680424	24.32254	1.6727467	0	0

Fig: Feature vectors sample labels 299_500-Resnet50

Now we take 100 sample images to test out how good this model is, these 100 images were split into 299*299 and also 299*500 so both data sets were run through the model and their feature vectors were found. To test if the model is good we find the distance of each test image from the label images by using a simple distance formula. Respective feature vectors of the test and sample image are subtracted and then squared and all the squared distances are added up and then divided by 2048 to get an average distance value as the number of feature vectors are 2048. These distance can be found in "feature_distance" for 299*299 and "feature_distance1" for 299*500.

Code to find the distance:

```
disfeatures=[]
for i in range(0,100):
   dis=[]
   for j in range(0,73):
      add=0
      for k in range(0,2047):
         sub = (l2[i][0][k]-l[j][0][k])**2
         add = add + sub
      add=add/2048
      dis.append(add)
   disfeatures.append(dis)
```

Code snippet: To Find Distance Between Two Image's Features

The disfeatures is what was saved into Excel sheet, the first few distances can be viewed in results section.

Now to compare the results we ran the images through another model called inception_v3 and the architecture is explained below:
Inception-v3 is a convolutional neural network that is 48 layers deep.
Inception v3 mainly focuses on burning less computational power by modifying the previous Inception architectures.The architecture of an Inception v3 network is progressively built.

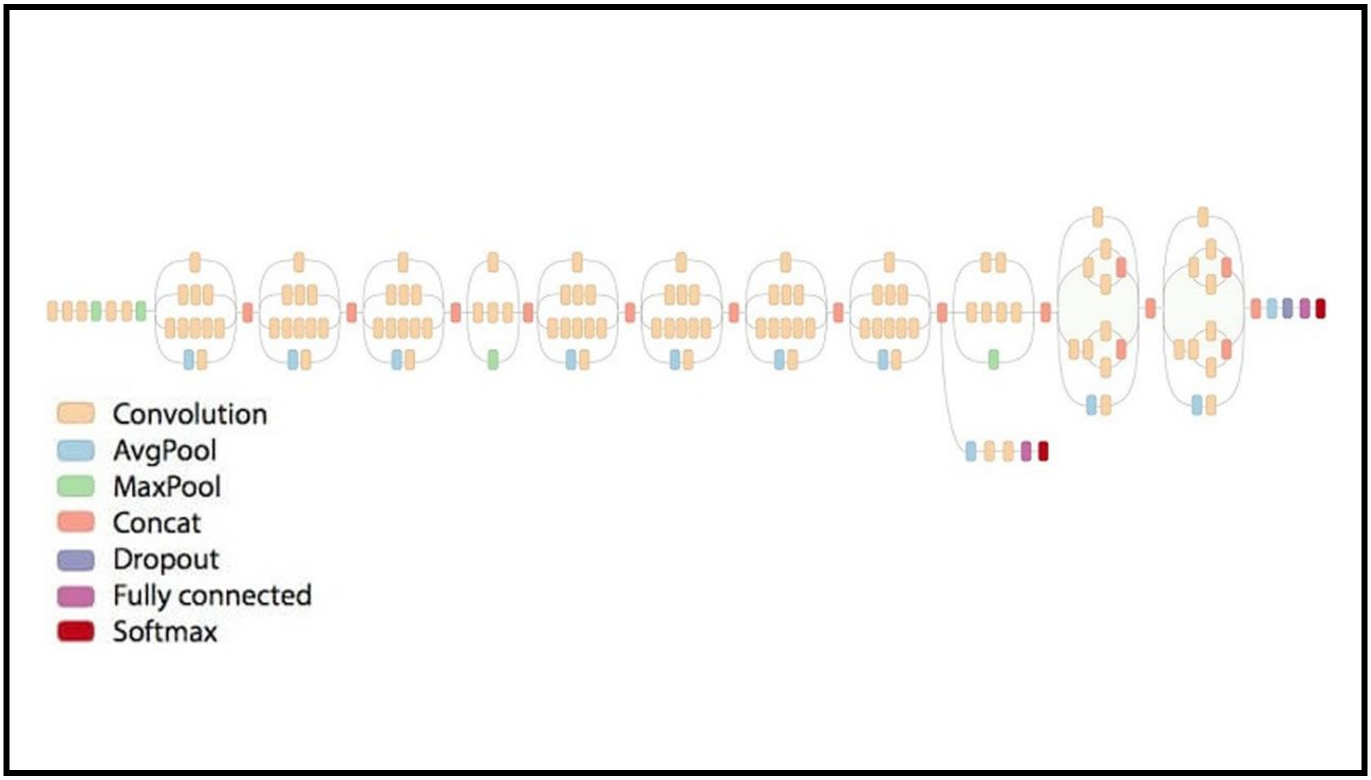


Fig: Inception_v3 Architecture

Inception v3 is a convolutional neural network model and by using GPU configured computer it takes weeks to train from scratch, tensorflow a machine learning framework which provides platform to train the classification layer with imagenet dataset using transfer learning mechanism, which keeps the weights and bias values of the feature extraction layer and removes parameters on classification layer of inception v3[6]. First the input images of size 299x299x3 are fed to the feature extraction layer of CNN. After that feature extraction layer calculates the feature values for each image, feature vectors are 2048 float values for each image. The classification layer of the CNN is trained with these feature vectors. The output labels in the classification layer is equal to the number of image classes on the dataset

The same code as the Resnet50 was used with one small change which is as follows:

```
url='https://tfhub.dev/google/imagenet/inception_v3/feature_vector/4'
```

The url changes which points to the inception_v3 model of tensorflow hub.

The feature vectors for the label data and the test data were taken and stored in excel sheets which are named "feature_v3_299_299" and "feature_v3_299_500" which have the feature vectors of the label data. And in the folders "feature_distance_v3_299_299" and "feature_distance_v3_299_500" have the distance of each test image to a sample image. The results of this analysis will be explained in the next section.

Below are the first 10 feature vectors of the first 20 images:

feature_v3_299_299

001_DGAFB-18_2.jpg_.jpg_.jpg

	A	B	C	D	E	F	G	H	I	J	K
1	001_DGAFB-18	2.845973	0.2568608	8.114507	16.412699	5.783505	1.4573632	3.852632	4.914076	13.013134	3.758657
2	001_DGBFB2 (2	0.25532255	9.907504	20.956127	25.955307	5.2036304	4.423435	7.532301	0.7025992	22.710056	4.7712507
3	001_DGBFB3 (3	1.3043972	9.740449	15.277913	26.133268	7.407022	8.539255	18.122877	0.039738335	21.328897	6.406266
4	001_DGBFB5_0	1.0366489	3.5902724	10.634958	8.894391	3.6741288	0.48015493	6.4394417	2.7806985	16.939981	6.556768
5	001_DGFW1 (11	0.3984795	11.120052	18.492739	26.568768	6.468583	5.183705	9.066516	2.0988898	24.643606	6.213155
6	001_DGFW46_1	4.7514825	13.101976	54.476463	24.62578	11.322186	5.232115	22.714376	6.9922705	33.984436	10.979604
7	001_DGUTI14_(	5.498443	170.8942	4.785547	104.61909	82.63592	0	14.59613	0	76.435425	2.121051
8	001_FW1240_1.	0.46812797	3.225378	34.900475	50.752007	0.9348731	7.330266	30.866566	5.7188187	50.587944	0.00723124
9	002_CF2_2.jpg_	1.0116638	0	105.36509	46.2274	26.181429	3.6792822	43.373848	39.041916	20.611616	33.340828
10	002_DGAFB-19_	2.1828907	0.708984	8.371278	45.48045	4.5771523	1.3292431	8.880152	0.5029392	18.01209	0.98844963
11	002_DGBFB2 (3	0.091975726	8.022797	15.2400055	73.83908	1.7252184	2.2839737	44.067562	0.38603806	52.83357	0.7505045
12	002_DGFB2 (17	1.0559125	69.93492	9.469738	61.98917	54.771168	0	11.998061	1.3246315	45.932224	0.61207294
13	002_DGFB2 (18	0.51676095	12.165291	21.339216	36.346424	6.8950505	3.215859	23.4095	0.778598	36.22133	1.4445678
14	002_DGFB2_1.j	4.8736305	0.55415577	52.657288	27.08926	9.008631	1.408498	17.102362	20.380745	21.588598	13.685191
15	002_DGFW31_1	2.4269407	8.858149	63.44425	56.109695	7.8043966	0	17.517824	4.413475	69.5719	2.9036887
16	002_DGGML2_(	2.7371838	8.2298565	15.357899	28.21141	4.582877	4.3004494	11.313791	1.9129093	16.4798	3.6060076
17	002_DGHFW2_(	1.9339161	1.8336142	21.66594	27.333942	4.0848722	2.0409164	6.460322	6.6417823	19.153343	11.264578
18	004_DFB49_2.jp	1.8631505	0.5402077	20.105305	39.847256	10.149728	0.18579957	28.883612	9.592332	50.35773	5.4405074
19	004_DGAFB-21_	2.1150997	2.798519	3.6201239	13.187537	4.680111	1.1271769	3.262154	0.92975706	4.942541	5.0955315
20	005_DGFB9_2.j	5.371175	8.938274	34.27765	123.03831	28.964449	9.018929	61.733406	0.22474393	73.81221	5.299463

Fig: Feature vectors sample labels 299_299-Inception_V3

feature_v3_299_500

001_DGAFB-18_2.jpg_.jpg_0.jpg_.jpg

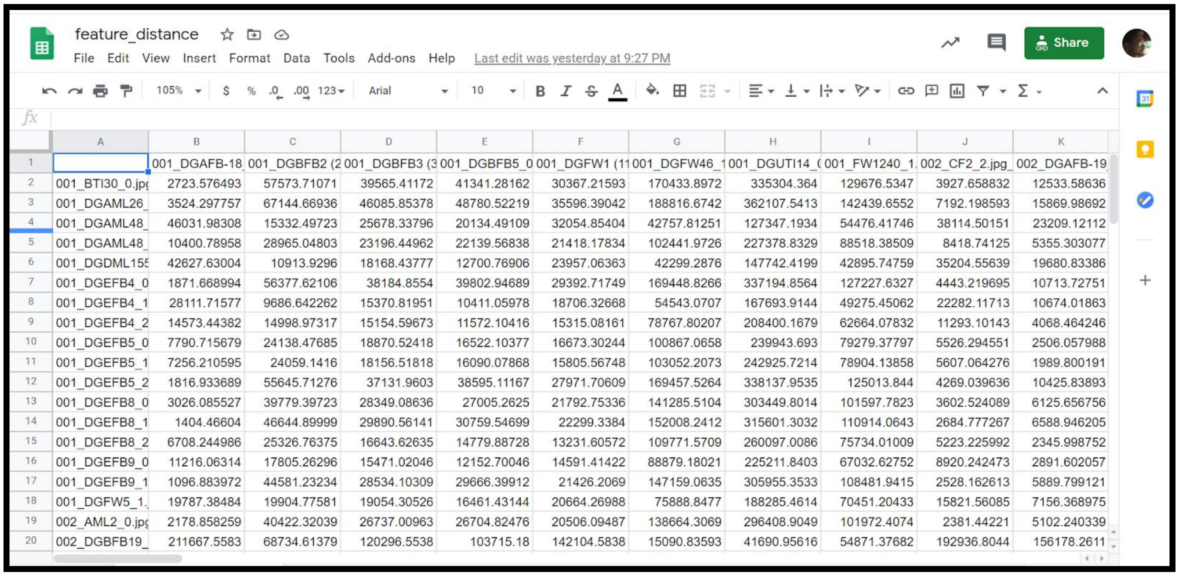
	A	B	C	D	E	F	G	H	I	J	K
1	001_DGAFB-18	0.05667034	1.7601606	3.5578668	7.5221896	3.8921754	0.067929626	8.272646	5.7809305	13.0184355	3.1433334
2	001_DGBFB2 (2	3.744286	6.641848	11.069436	19.56487	7.3012843	1.1519511	13.656314	0.54959124	10.432066	6.52535
3	001_DGBFB3 (3	2.3203206	2.7349553	17.566692	14.654022	6.7827783	3.4475427	4.0266848	0.34087136	24.905361	8.7308855
4	001_DGBFB5_0	1.6743572	8.3495245	5.5273275	16.37742	9.774017	0.41601196	1.560455	0.28703278	16.86357	3.5373435
5	001_DGFW1 (11	1.6780571	1.748295	7.1245313	17.585768	7.694409	7.5094733	3.1766498	1.0593128	17.609213	2.0038605
6	001_DGFW46_1	3.7602046	5.0019	68.11996	39.35035	30.337507	9.696179	19.950382	1.2848399	25.014433	58.894676
7	001_DGUTI14_(	1.5079025	146.41248	9.726004	34.80285	61.25879	0	37.78881	0	81.465836	2.652559
8	001_FW1240_1.	5.1727686	2.8301568	32.66437	14.189798	9.267847	2.3057325	24.911154	14.860421	7.0725017	11.172247
9	002_CF2_2.jpg_	1.3613044	1.3491493	41.49888	23.291397	14.375608	0.9608272	24.061811	14.04141	15.437032	5.0239806
10	002_DGAFB-19_	6.8687553	2.5858548	22.66031	30.73788	14.095843	0.85321367	6.1139255	2.2282035	24.844482	10.450633
11	002_DGBFB2 (3	3.321362	16.367199	12.374646	121.11353	15.678102	0.689935	29.851215	0	69.82162	2.0149627
12	002_DGBFB4_2	4.23621	19.275677	12.051004	20.704714	7.6451426	0.721657	8.759604	0.42780545	27.791733	0.5730408
13	002_DGFB2 (17	0.5916881	109.08356	15.218079	57.43023	63.030605	0	21.295488	0.56597227	45.716076	0.4204981
14	002_DGFB2 (18	0.7175949	9.222769	21.244368	21.66968	9.907439	4.115742	9.587275	4.7645946	18.47524	2.6367316
15	002_DGFB2_1.j	6.1177664	0.8791596	62.48478	16.569744	13.307577	2.2296307	65.86833	21.327478	10.984601	4.51309
16	002_DGFW2_2.	0.57032657	47.573444	11.057467	24.779383	18.721676	0.0143484175	20.591116	0.09725886	25.241222	3.0679345
17	002_DGFW31_1	0.0786866	7.292085	16.608646	23.897997	6.1537213	3.1566694	13.869635	0.50741845	27.203144	6.8704104
18	002_DGGML2_(	1.4477504	1.8145134	3.3907416	17.996233	11.113953	3.1934192	5.441756	0.038801372	8.234972	1.2990123
19	002_DGHFW2_(	2.9492648	7.467796	11.354314	10.8424635	12.459427	1.6445583	2.8394578	0.48633903	7.6598234	6.3666453
20	004_DFB49_2.jp	1.5438136	0.0255679	38.03858	23.435331	2.516602	0	15.512159	12.788872	32.058846	3.4641433

Fig: Feature vectors sample labels 299_500-Inception_V3

5) Results

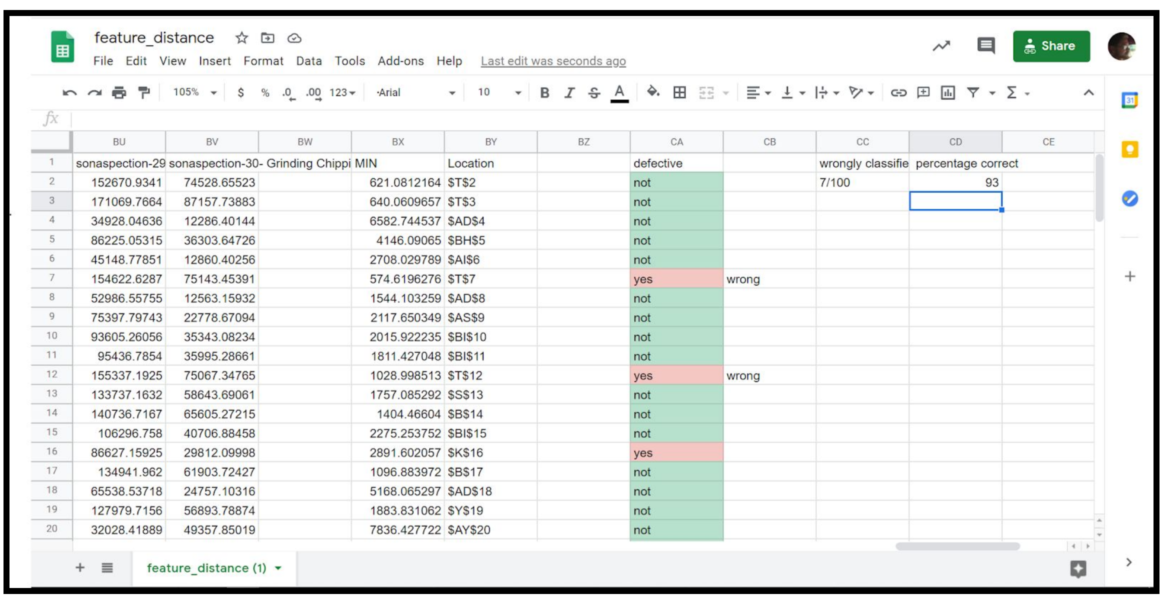
Before I start comparing the results I would like to make an observation, we were to compare the feature vectors of Resnet50 and InceptionV3 we can see that the values in Resnet50 are higher compared to InceptionV3, but it can also be observed the number of zero column have increased, Resnet50 gives us 75458, 92139 while InceptionV3 4885, 4564 what can be deduced is that Inception_V3 is picking up more features then Resnet50, but we would like to have a model which gets us the required features.

Below is an example of the distance excel sheet:



	A	B	C	D	E	F	G	H	I	J	K
1		001_DGAFB-18	001_DGBFB2 (2	001_DGBFB3 (3	001_DGBFB5_0	001_DGFW1 (11	001_DGFW46_1	001_DGUTI14_(	001_FW1240_1.	002_CF2_2.jpg_	002_DGAFB-19
2	001_BTI30_0.jpg	2723.576493	57573.71071	39565.41172	41341.28162	30367.21593	170433.8972	335304.364	129676.5347	3927.658832	12533.58636
3	001_DGAML26_	3524.297757	67144.66936	46085.85378	48780.52219	35596.39042	188816.6742	362107.5413	142439.6552	7192.198593	15869.98692
4	001_DGAML48_	46031.98308	15332.49723	25678.33796	20134.49109	32054.85404	42757.81251	127347.1934	54476.41746	38114.50151	23209.12112
5	001_DGAML48_	10400.78958	28965.04803	23196.44962	22139.56838	21418.17834	102441.9726	227378.8329	88518.38509	8418.74125	5355.303077
6	001_DGDML155	42627.63004	10913.9296	18168.43777	12700.76906	23957.06363	42299.2876	147742.4199	42895.74759	35204.55639	19680.83386
7	001_DGEFB4_0	1871.668994	56377.62106	38184.8554	39802.94689	29392.71749	169448.8266	337194.8564	127227.6327	4443.219695	10713.72751
8	001_DGEFB4_1	28111.71577	9686.642262	15370.81951	10411.05978	18706.32668	54543.0707	167693.9144	49275.45062	22282.11713	10674.01863
9	001_DGEFB4_2	14573.44382	14998.97317	15154.59673	11572.10416	15315.08161	78767.80207	208400.1679	62664.07832	11293.10143	4068.464246
10	001_DGEFB5_0	7790.715679	24138.47685	18870.52418	16522.10377	16673.30244	100867.0658	239943.693	79279.37797	5526.294551	2506.057988
11	001_DGEFB5_1	7256.210595	24059.1416	18156.51818	16090.07868	15805.56748	103052.2073	242925.7214	78904.13858	5607.064276	1989.800191
12	001_DGEFB5_2	1816.933689	55645.71276	37131.9603	38595.11167	27971.70609	169457.5264	338137.9535	125013.844	4269.039636	10425.83893
13	001_DGEFB8_0	3026.085527	39779.39723	28349.08636	27005.2625	21792.75336	141285.5104	303449.8014	101597.7823	3602.524089	6125.656756
14	001_DGEFB8_1	1404.46604	46644.89999	29890.56141	30759.54699	22299.3384	152008.2412	315601.3032	110914.0643	2684.777267	6588.946205
15	001_DGEFB8_2	6708.244986	25326.76375	16643.62635	14779.88728	13231.60572	109771.5709	260097.0086	75734.01009	5223.225992	2345.998752
16	001_DGEFB9_0	11216.06314	17805.26296	15471.02046	12152.70046	14591.41422	88879.18021	225211.8403	67032.62752	8920.242473	2891.602057
17	001_DGEFB9_1	1096.883972	44581.23234	28534.10309	29666.39912	21426.2069	147159.0635	305955.3533	108481.9415	2528.162613	5889.799121
18	001_DGFW5_1.	19787.38484	19904.77581	19054.30526	16461.43144	20664.26988	75888.8477	188285.4614	70451.20433	15821.56085	7156.368975
19	002_AML2_0.jpg	2178.858259	40422.32039	26737.00963	26704.82476	20506.09487	138664.3069	296408.9049	101972.4074	2381.44221	5102.240339
20	002_DGBFB19_	211667.5583	68734.61379	120296.5538	103715.18	142104.5838	15090.83593	41690.95616	54871.37682	192936.8044	156178.2611

Fig: The first few distances from the sample label images



	BU	BV	BW	BX	BY	BZ	CA	CB	CC	CD	CE
1	sonaspection-29	sonaspection-30-	Grinding Chippi	MIN	Location		defective		wrongly classifie	percentage correct	
2	152670.9341	74528.65523		621.0812164	T2		not		7/100	93	
3	171069.7664	87157.73883		640.0609657	T3		not				
4	34928.04636	12286.40144		6582.744537	AD4		not				
5	86225.05315	36303.64726		4146.09065	BH5		not				
6	45148.77851	12860.40256		2708.029789	AI6		not				
7	154622.6287	75143.45391		574.6196276	T7		yes	wrong			
8	52986.55755	12563.15932		1544.103259	AD8		not				
9	75397.79743	22778.67094		2117.650349	AS9		not				
10	93605.26056	35343.08234		2015.922235	BI10		not				
11	95436.7854	35995.28661		1811.427048	BI11		not				
12	155337.1925	75067.34765		1028.998513	T12		yes	wrong			
13	133737.1632	58643.69061		1757.085292	S13		not				
14	140736.7167	65605.27215		1404.46604	B14		not				
15	106296.758	40706.88458		2275.253752	BI15		not				
16	86627.15925	29812.09998		2891.602057	K16		yes				
17	134941.962	61903.72427		1096.883972	B17		not				
18	65538.53718	24757.10316		5168.065297	AD18		not				
19	127979.7156	56893.78874		1883.831062	Y19		not				
20	32028.41889	49357.85019		7836.427722	AY20		not				

Fig: The analysis of the distances from the sample label images

Thus you can see that we have first determined if the image is defective or not and the ones that were wrongly classified are written wrong on the side. The above images are the excel sheet screen shots of the first few distances and then the analysis we have done by finding the min and the location of the min and determining if classified properly.

299*299_Resnet50	Predicted Non-Defective	Predicted Defective
Actual Non-Defective	79	0
Actual Defective	7	14

Table:Confusion Matrix_299_299_Resnet50

299*500_Resnet50	Predicted Non-Defective	Predicted Defective
Actual Non-Defective	76	0
Actual Defective	6	18

Table:Confusion Matrix_299_500_Resnet50

299*299_Inception_v3	Predicted Non-Defective	Predicted Defective
Actual Non-Defective	79	0
Actual Defective	8	13

Table:Confusion Matrix_299_299_Inception_V3

299*500_Inception_v3	Predicted Non-Defective	Predicted Defective
Actual Non-Defective	76	0
Actual Defective	6	18

Table:Confusion Matrix_299_500_Inception_V3

	299*299_resnet50	299*500_resnet50	299*299_inception_v3	299*500_inception_v3
#Defective samples	21	24	21	24
#Non defective	79	76	79	76
#Wrongly classified	7	6	8	6
Accuracy %	93	94	92	94

Table:Final Results Table

Those we can observe that the Resnet50 had a slightly better classification but looking at the feature vectores it would be better to go ahead with Resnet50 as we use more images the variance will increase and would require models that would enhance features better.

6) Conclusions and Future works

We were able to compare the results obtained using both the models and infer resnet50 as a better classifier. Minimum defects were wrongly classified in the resnet model. The feature extraction could have been implemented better if we were successful in removing the texts on the radiographic images, which otherwise could be a major problem and affect our feature vector results. This is because the texts might confuse the classifier for defects and hinder the results. To avoid this we can come up with a solution to remove these texts and other background noise in the images.mOne solution could be background subtraction using histogram thresholding and another could be the selection of the region of interest (ROI) which is the reduced zone of the image that will be processed. This will allow the operator to process the useful parts of the image and thus

reduce computation complexity. We can then focus on the segmentation of the defects caused only by the welding process. The defects outside the weld bead are mainly due to manufacturing. For this reason we chose to limit the ROI to the weld bead.

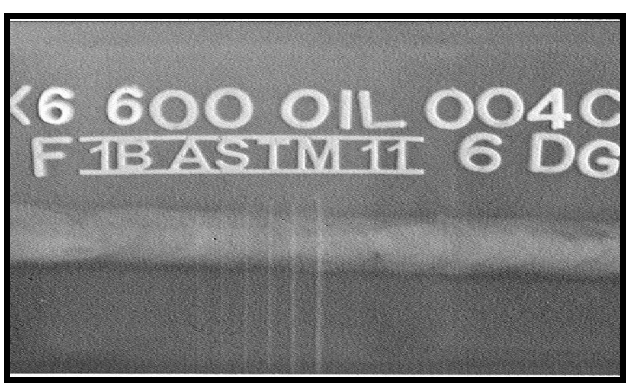


Fig: Example of the Text being more Prominent than the Weld Seam

Further taking the project to the next step we can apply our own classifier algorithm once the images are segmented. The rough outline of the classifier algorithm for detecting 3 defects i.e., Lack of Penetration(LOC), Incomplete Fusion(IF), and effect of undercut(EUC). Is given below:

Input: segmented defective weld images
Output : LOP(lack of penetration),IF(incomplete fusion), or EUC(Effective undercut)

1. Find interest area of gray scale image
2. Calculate Start point,Maximum Point and end point for each profile line of images and store these in arrays Sta,Ste,Sx.
3. Calculate mean of these arrays.
4. Binarize the gray scale images
5. Find the boundaries of the defective portions in the image.(Bn number of boundaries).
6. Detect maximum length oriented in the direction of weld seam for each boundary.
7. Calculate the ratio of maximum length oriented in the direction of weld seam to the maximum length the vertical direction of weld seam.
8. If the ratio is higher than threshold:
    a. Then defect is lengthy shape
        i. If located near mean of Sx output: “LOP” and stop.
        ii. If located near mean of Sta or Ste output: “EUC” and stop.
        iii. Output: “IF”

9. End of algorithm.

We have been successful in detecting the text in the image, but were not able to remove and apply a constant colour in its place below are 2 images

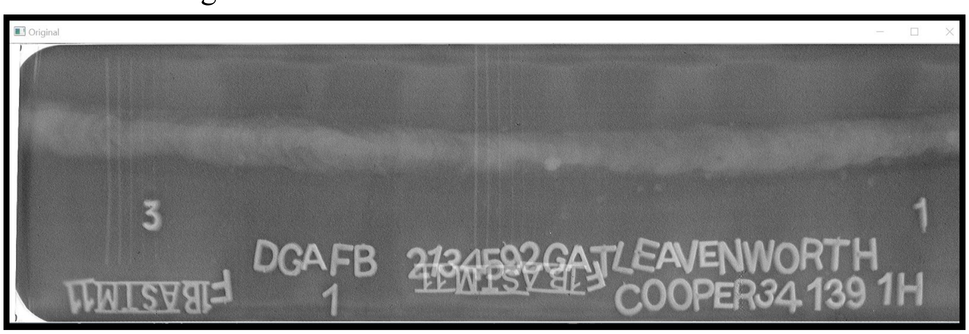


Fig: Original Image

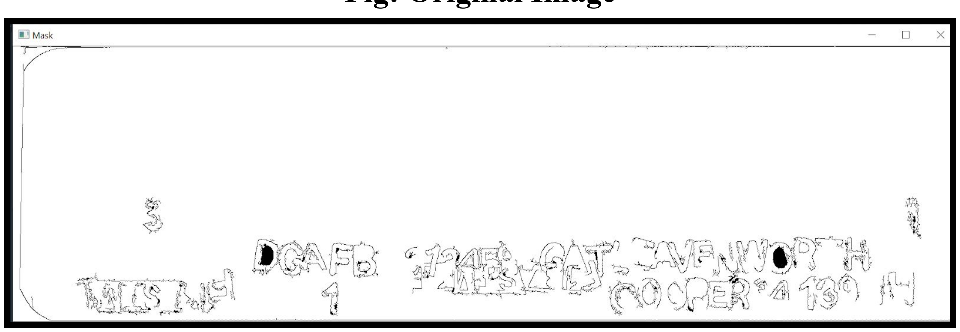


Fig: Letter Detected Mask

7) References

1. *Tensorflow hub from google*

2. *Keras from google*

3. *Imagenet models- ResNet50,Inception_v3*

4. Moghaddam, Alireza Azari (2015). *[IEEE 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI) - Tehran, Iran (2015.11.5-2015.11.6)] 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI) - Image processing techniques for classification of linear welding defects.*

5. *Saba MADANİ 1 , Mortaza AZİZİ -Detection of Weld Defects in Radiography Films Using Image Processing Cumhuriyet University Faculty of Science Science Journal (CSJ), Vol. 36, No: 3 Special Issue (2015).*