Mondragon Unibertsitatea
Goi Eskola Politeknikoa
Escuela Politécnica Superior

WELD IMAGE INSPECTION

Deep Learning

Masters in Robotics and Control

Raj Tushar Khatri
Manisha Vijay Sampat

TABLE OF CONTENTS

1. INTRODUCTION

The task deals with inspection of welds in a company. The major idea is to detect the pores in the welds to identify the good and the bad welds. Welding inspection is a difficult process as one must be an expert to understand the integrities of a good and a bad weld. However, with the growing need for and importance of Machine learning for such tasks, companies are adopting means of Deep learning algorithms to eliminate the manual inspection of welds and adopt more machine-driven and accurate methods of deep learning. In this task, we propose carrying out the weld inspection using the basic deep learning technology like Convolutional neural network (CNN). To further advance and adapt more efficiently, the UNET architecture and auto-encoders have also been developed to segment and detect anomalies in the weld images. As shown in Figure 1 and Figure 2 below, we try to segment the images as one which is good and one defective with weld pores (Figure 2).

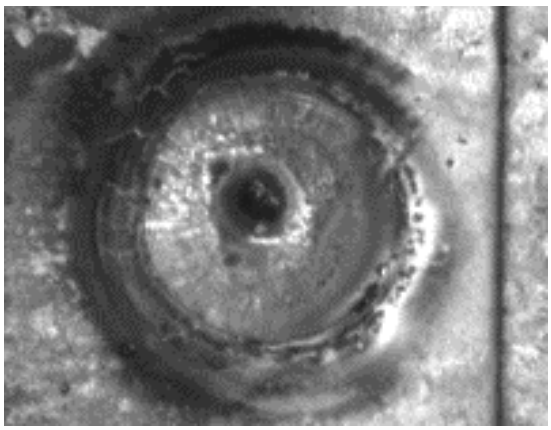

Figure 1: Good weld

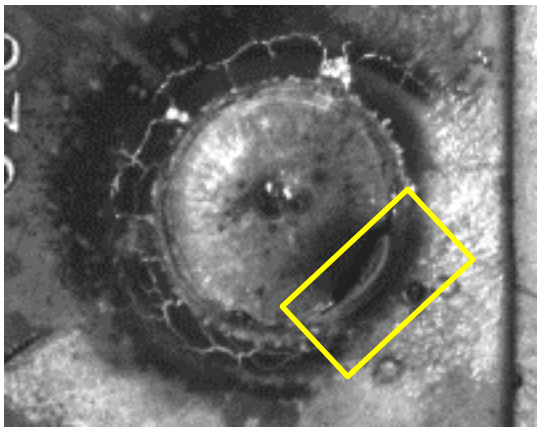

Figure 2: Defective weld

The dataset used to generate and test the model as an offline approach consists of 117 images containing both good and bad images (ones with the pores). Each bad image with a defective weld consists of a corresponding image highlighting the region of interest of that image which highlights the area in which the defective pore is available. The next section describes the approach taken to solve this desired application.

2. APPROACH

To carry out image segmentation and anomaly detection in the best possible way, the approach adopted was to first segment the data into train and test to train and test the model offline. Once the data set is generated, a basic CNN model would be trained using sliding window of 64*64 for inspecting the images. The model would be tested with the validation data and noting the accuracy of prediction. In addition to the basic CNN model, the similar task of segmentation is carried out using the UNET architecture, where the agenda would be to first understand the UNET architecture and apply it for segmenting using the 64*64 sliding window and whole images.
The second part of the task would deal with anomaly detection using the autoencoder and

UNET architecture. The anomaly in the autoencoder is detected based on the reconstruction error. Each of these steps are explained in detail in their respective upcoming sections.

3. DATA SET GENERATION

As a primary foundation to training and testing the model for desired purposes, it is important to bifurcate data points for training and testing in a desired proportion suited best for the application. The data generation in this project has been done in two ways differently for the basic CNN and autoencoder and separately for the UNET architecture both described below in detail.

i) *Train and Validation generator for CNN and autoencoder*

The original images are firstly divided into directories of 'train' and 'validation', with 'good' and 'bad' images in them respectively. The images stored in these directories are resized to 64x64 size images from their original size 401x327 pixels. As a result, the data is bifurcated into directories as shown in Figure 3.

```
print('Train::')
print(len(Train_G),len(Train_B))

print('')
print('Validation::')
print(len(Val_G),len(Val_B))
```

```
Train::
569 1252

Validation::
61 132
```

Figure 3: Train and validation data directories

Further, in order to train the CNN model, these images are further rescaled by transforming each pixel value from 0-255 to 0-1 to have even representation and updating in weights for the training model. This is done using the ImageDataGenerator. We then generate 'train_generator' and 'validational_generator' which contains the 64x64 images that are rescaled using the method above and is seen as shown in Figure 4.

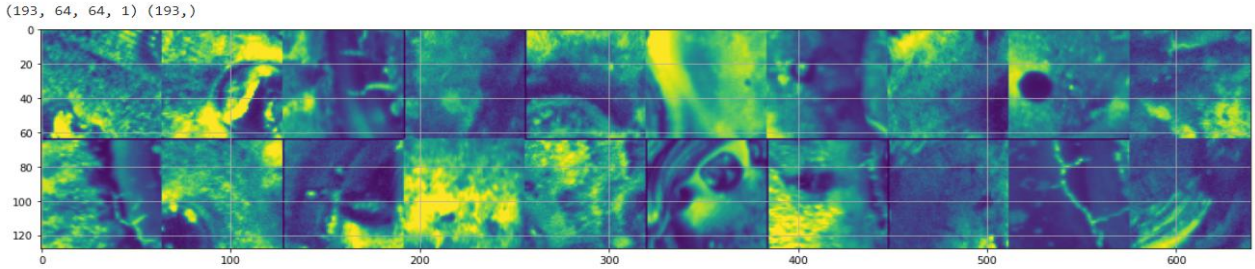

Figure 4: 64x64 size validation dataset

From these train and validation generators which are a list of the images an array is created [number_of_images , image_width, image_height , no_of_channels] named as x_train and x_train which are finally used to train and test the CNN model. This conversion is done using batches of images. In final, the train data is created with size as seen in Figure 5 and a similar approach is done for the test images.

```
x_train.shape

(1821, 64, 64, 1)

y_train.shape

(1821,)
```

Figure 5: Train dataset

ii) *UNET Architecture*

We have used the concept of Image Stacking. Image stacks are composed of a number of related images opened on top of each other in a single window. Stacks can represent different channels from the same image, z-slices captured on the microscope or time series. When channel, Z-series or time information is recorded the images are usually opened as stacks automatically when bio-formats importer is used to open the files. We generate a stack by ourselves from related BMP images. For this purpose, we used the help of ImageJ tool for stacking the images and its Region of Interest (ROI).

In Figure 6, we see how to make images to stack using ImageJ. Load all the images that we want to stack and then go the path of Stacks-Images to Stack, then there will be the number of images to be stacked, if it is RGB or not and other parameters to edit.

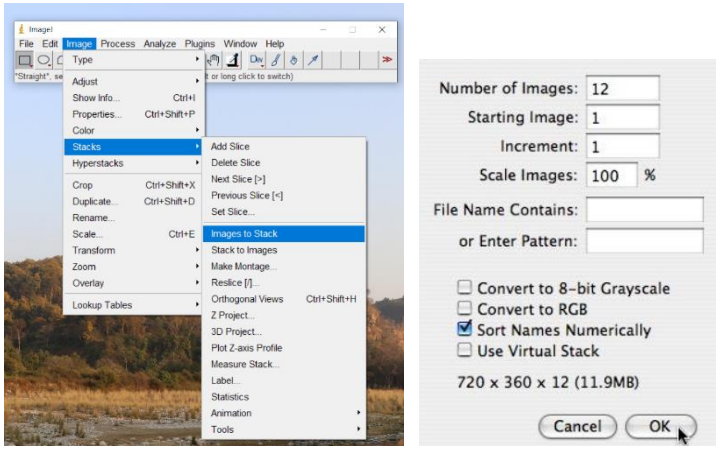

Figure 6: ImageJ Stacking

After the same methods are done for both original weld images and its ROI, the stack files in TIFF format will be visible like the below Figure 7. Each stack has a number of slices, in our case it is 33 slices in 1 stack. For each image, there is a mask image in the stack at the same location.

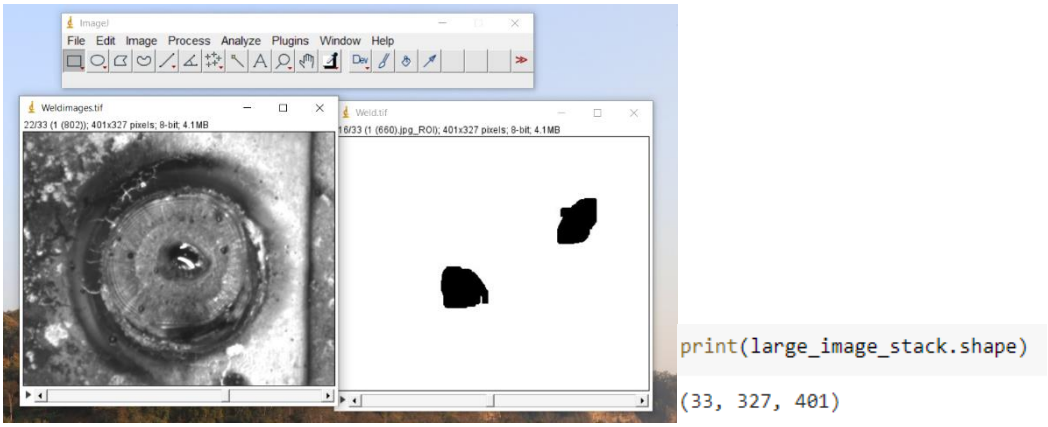

Figure 7: ImageJ Weld and Mask Stack Shapes

For the next processing of the images, we have used a library of python known as patchify. patchify can split images into small overlap able patches by given patch cell size, and merge patches into original image. In our case the images were patched to 64 x 64 patches with steps of 64, that means there will be no overlap between those patches as it is shown in the next Figure 8. It is similar to doing the image resizes like the train and validation generators in previous case. To install this library, we used “pip install patchify”

```
#Step of 64 for each 64 by 64 patch,here it means no overlap
patches_img = patchify(large_image, (64, 64), step=64)
```

Figure 8: Patchify Batch for image

Further, in order to train the UNET model, these images are further rescaled by transforming each pixel value from 0-255 to 0-1 to have even representation and updating in weights for the training model. Another constrained in our case is the UNET model requires a 3-channel input of the images but only 1 Channel for the ROI/Masks, so in order to be compatible, we decided to make the grayscale image to 3 channels by the following “np.stack” command in the below Figure 9.

```
#Convert grey image to 3 channels by copying channel 3 times.
#We do this as our unet model expects 3 channel input.
images = np.stack((images,)*3, axis=-1)
```

Figure 9: 3-Channel Image conversion

Hereby we finally get the images as per the requirement of our model as images and its equivalent mask as shown in the next Figure 10, which will help us to train the model perfectly.

```
print(images.shape)
print(masks.shape)
```

```
(990, 64, 64, 3)
(990, 64, 64, 1)
```

Figure 10: UNET Train dataset

In order to test if the dataset is generated in the desired fashion with its mask, it was checked by displaying images and the corresponding mask with the region of interest and check if they are as desired and the results are shown in Figure 11.

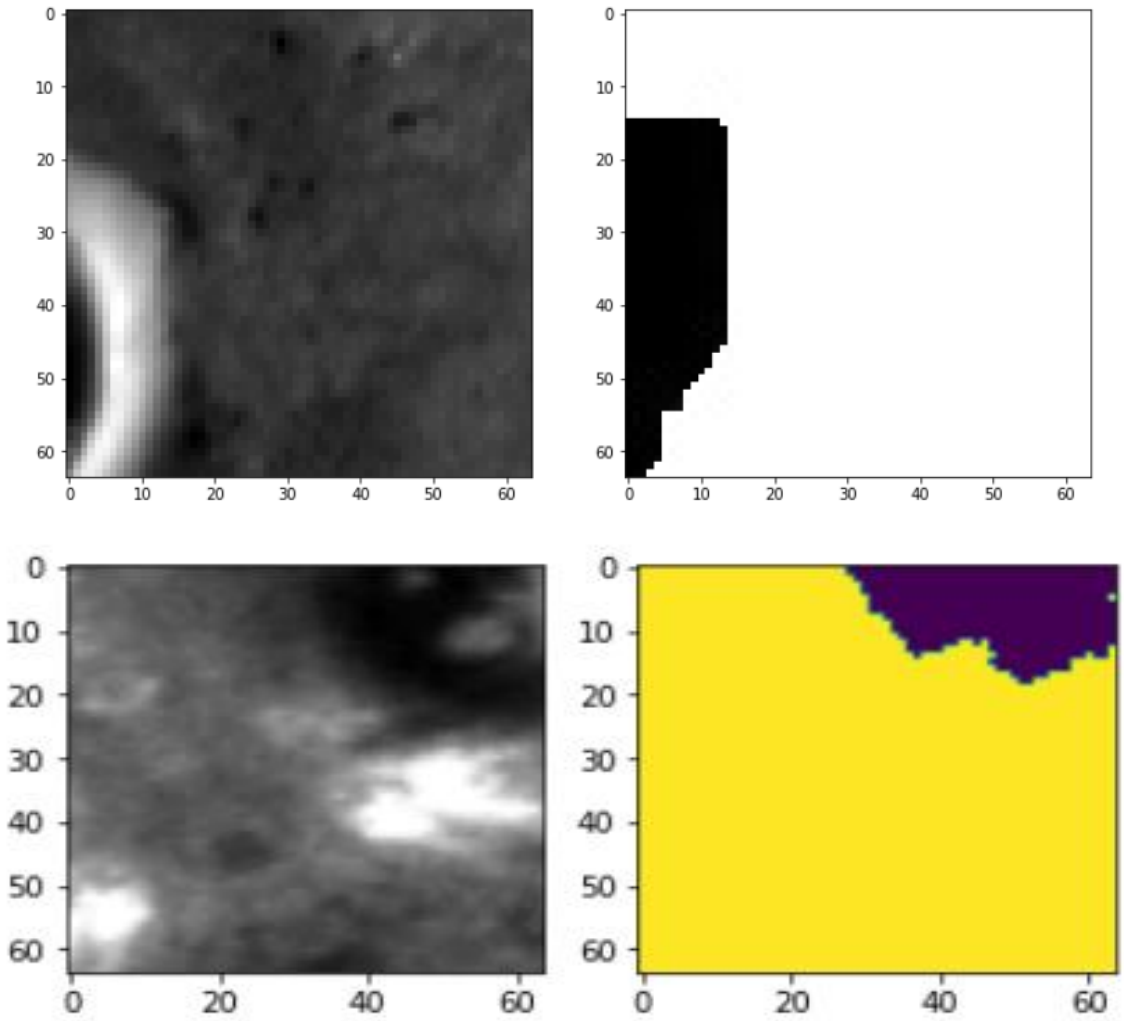

Figure 11: UNET Train dataset images

4. IMAGE SEGMENTATION USING CONVOLUTIONAL NEURAL NETWORK (CNN)

Once the dataset is created, a segmentation is carried out using a CNN network. The CNN network defined for the task consist of convolution, batch normalization and max pooling layers with the output in one class, i.e either good or defective weld. Furthermore, a flatten, dense and dropout layer is used. The model is trained with the train dataset and validated on the test dataset with 100 epochs in the model training and the result of the training is shown in Figure 12.

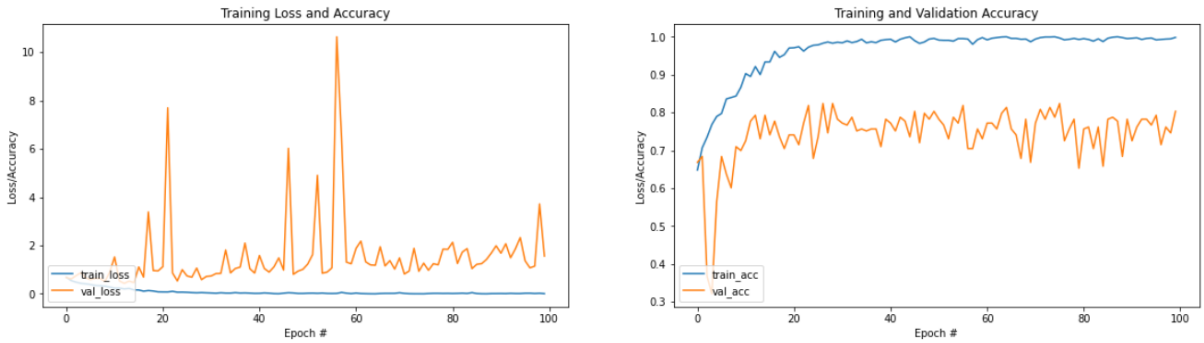

Figure 12: Training and validation accuracy history of basic CNN model

From the training history, it can be seen that the model does not perform very well in the test with the validation data, which suggests that there could be some overfitting in the model which means it performs well in the data it is trained with but tends to underperform with the unknown validation data. In order to test this, we then make some predictions in the test data. First a prediction is done on the test data and the result is shown in Figure 13 and 14. Though the one in Figure 13 can predict the defect correctly with a 85.79% confidence, the one in Figure 14 predicts it right, though the confidence of prediction is nil, therefore it would be right to say that the model is possibly overfitted.

```
Real = 0.0 :: Pred. = 0 Conf. 85.79% // second  0 %85.79
```

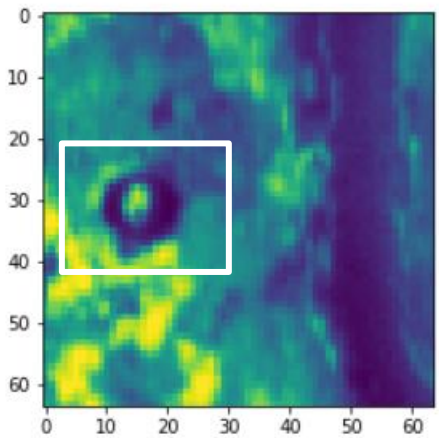

Figure 13: Prediction of model on test images

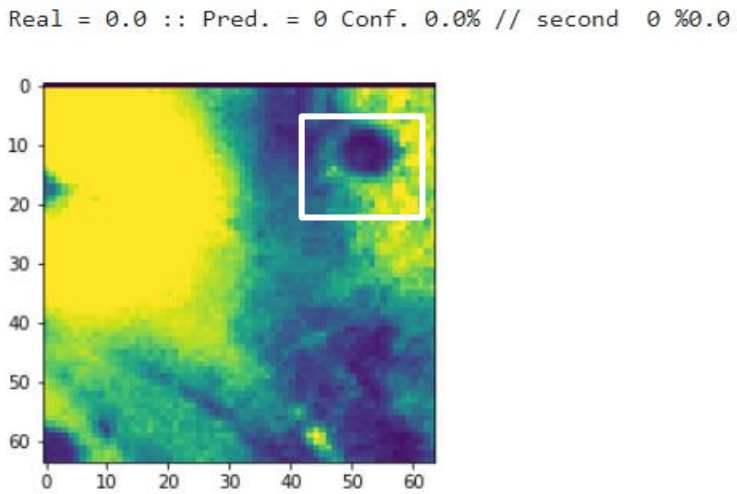

Figure 14: Prediction of model on test images

To further test the model, predictions were made using the model against full size test images and not 64x64 images. The performance of the model on the same was not so impressive either.

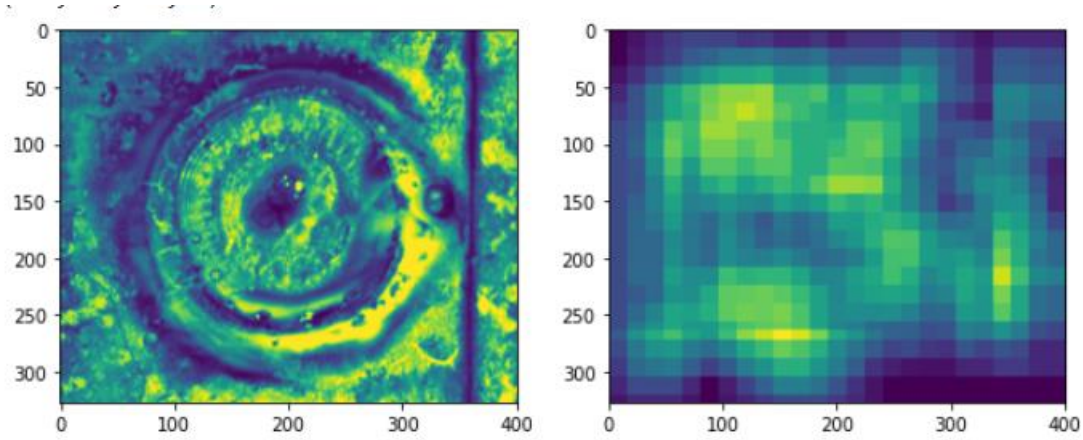

Figure 15: Image segmentation using CNN

It can therefore be concluded that the model is not able to predict the segmentation accurately on full size images as seen in Figure 15. One possible reason of the same could be the amount of dataset it is trained with, which could be less and that the model is learning the noise around the image. Considering the application, it is highly difficult to understand the difference between a good and a bad weld and therefore the CNN could be used for a very primary analysis in the application and rather efficient methods are described in later sections.

5. IMAGE SEGMENTATION USING UNET ARCHITECTURE

Once the dataset has been generated and manipulated as per the requirements, we start with the UNET Model. In our case, we take the help of segmentation models from python library. Segmentation models are python library with Neural Networks for Image

Segmentation based on Keras (TensorFlow) framework using the “pip3 install -U segmentation-models". The main features of this library are that it has a High-level API with just two lines to code, we can create a neural network. We can do binary as well as multi class segmentation. It has 25 available backbones for each of the architecture. All backbones have pre-trained weights for faster and better convergence.[1]

```
!pip3 install -U segmentation-models


%env SM_FRAMEWORK=tf.keras
import segmentation_models as sm
```

Figure 16: Segmentation Models install

The backbone that we decided to choose is ResNet34. Resnet34 is a 34-layer convolutional neural network that can be utilized as a state-of-the-art image classification model. However, it is different from traditional neural networks in the sense that it takes residuals from each layer and uses them in the subsequent connected layers (similar to residual neural networks (RNN) used for text prediction). [2]

```
BACKBONE = 'resnet34'
preprocess_input1 = sm.get_preprocessing(BACKBONE)
```

Figure 17: Resnet34 implementation

ResNet34 is a model that has been pre-trained on the ImageNet dataset--a dataset that has 100,000+ images across 200 different classes. The ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories with a typical category, such as "balloon" or "strawberry", consisting of several hundred images.

There were two possible ways to utilize and use the UNET model. One was to use it from scratch with zero weights in each of the layers, however teaching the model from a zero would be inefficient and require more time for accurate predictions. The other was to use pretrained weights in already available models which are proven to show more performance and accuracy [3], hence, we used the UNET model with pre trained weights.

A UNet model with ResNet encoder (ResNet-UNet) was constructed for the deep learning image segmentation. The architecture of the ResNet-UNet follows traditional UNet’s structure, composing of encoder (down-sampling) and decoder (up-sampling) portions. The key difference between the ResNet-UNet and the UNet is that the down-sampling

encoder adopts the ResNet-34 model, which is widely applied in image classification area and benefits from the advanced deep residual learning method. The ResNet-34 used here consists sequentially of firstly a 7×7 convolutional layer (CL), a max pooling layer, and the following 16 residual blocks. Each residual block contains two 3×3 convolutional layers with ReLu and a batch normalization identity shortcut connection (Figure 18). Therefore, the ResNet down-sampling portion totally consists of 34 layers. To match the image size changes [32, 64, 128, 256] in the up-sampling portion, the corresponding 4 feature maps from the down-sampling structure are: (1) output from the 7×7 CL (feature map size 256×256); (2) output from the first 3 residual blocks (feature map size 128×128); (3) output from the second 6 residual blocks (feature map size 64×64); (4) output from the third 4 residual blocks (feature map size 32×32). The up-sampling portion starts from the 512-channel 16×16 feature map. It is convoluted by a 2×2 transposed convolution with up-sampling factor 2 (stride=2). The up-sampled output (128-channel 32×32 feature map) is concatenated with 1×1 convolutional output (128-channel) of the corresponding feature map from the down-sampling counterpart. The procedure is repeated until the output reaches image size 256×256. Then, the final layer transposes the 256-channel 256×256 feature map to a 1-channel 512×512 feature map, giving a probability map of the segmented objects [4]

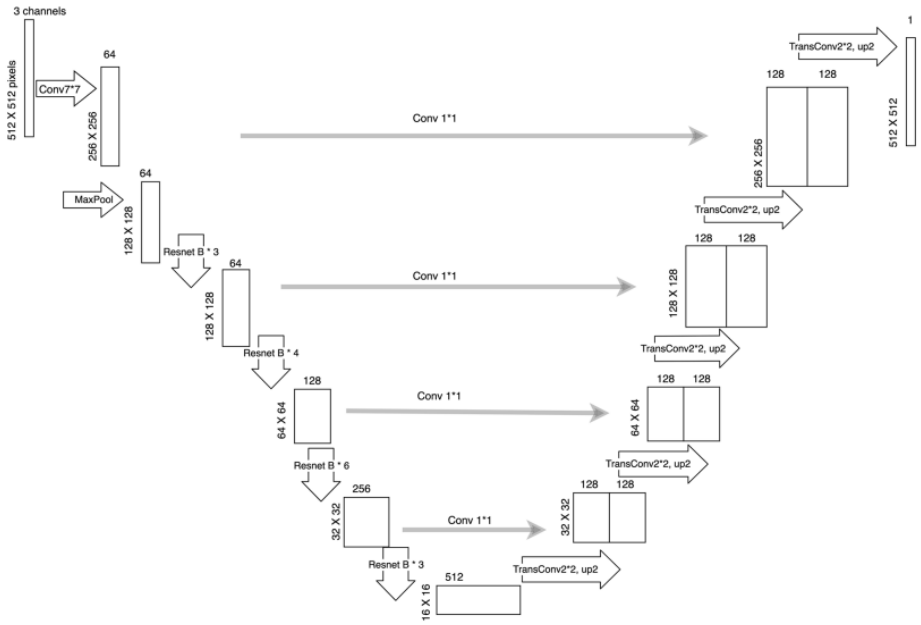

Figure 18: UNET Model with ResNet34 Block

```
model = sm.Unet(BACKBONE, encoder_weights='imagenet')
model.compile('Adam', loss=sm.losses.bce_jaccard_loss, metrics=[sm.metrics.iou_score])
print(model.summary())
```

Figure 19 : Implementation of UNet Model

The above-described model was trained and the performance of the training history was checked as shown in Figure 20.

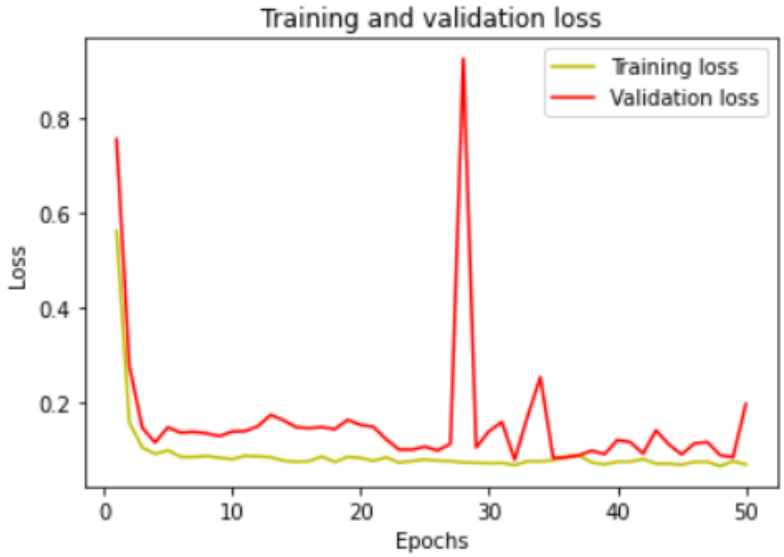

Figure 20: Training and validation history for UNET Model

From the training history it can be understood that by the end of the run of 50 Epochs, the training and validation loss converge which depict that the model is not overfitted and it would perform well with both the training and test data.
Another parameter tested was also the Intersection over union (IOU) score. Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. Computing the Intersection over Union is as simple as dividing the area of overlap between the bounding boxes by the area of union. Higher the IOC score, means that the features of interested are detected.

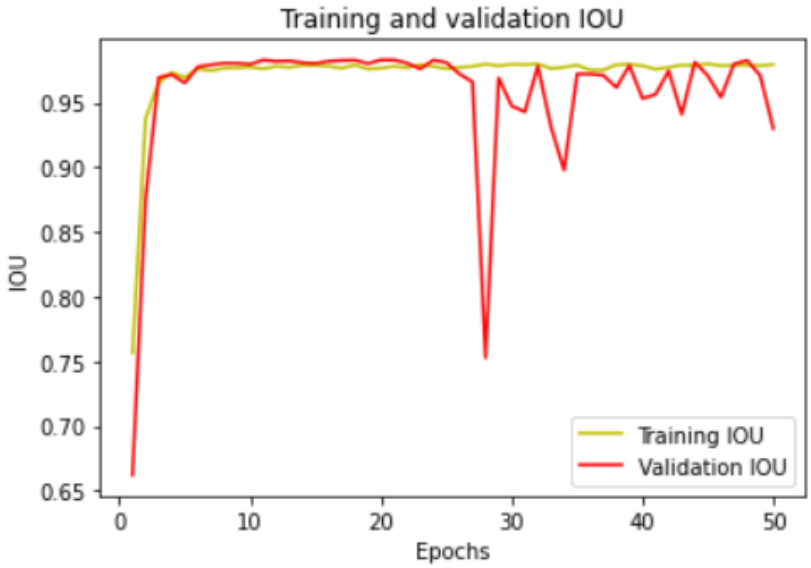

Figure 21: Training and validation history for UNET Model

In final, the model is tested by using the test data set and the predicts of segmentation are made using the UNET model as shown in Figure 22.

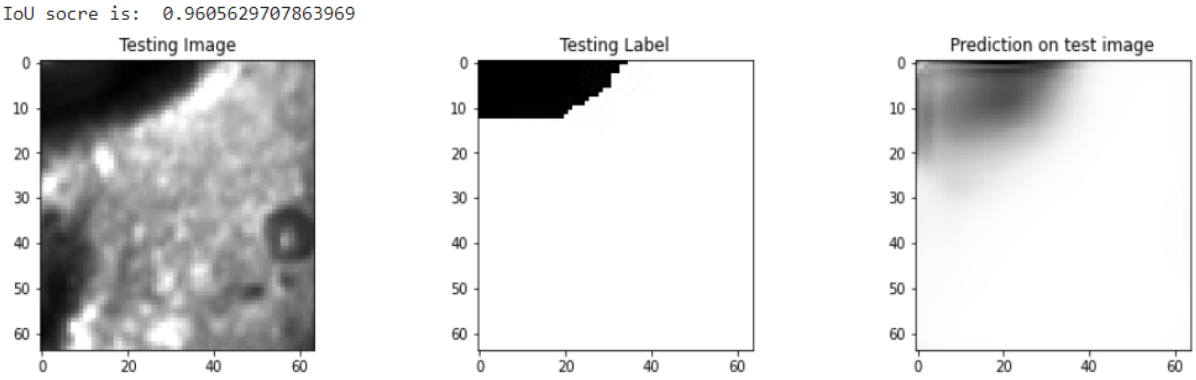

Figure 22: Training and validation history for UNET Model

It can therefore be said that the UNET model works accurately in segmenting the good and bad images.

6. ANOMALY DETECTION USING AUTOENCODER

The basic approach to detect anomaly is the implementation of autoencoders. Autoencoders are unsupervised learning models that try to minimize reconstruction error as part of its training. Anomalies are detected by checking the magnitude of the reconstruction loss. In order to train the autoencoder, the train and test data as described in Section 3(i) is used.

The architecture of the convolution autoencoder defines the construction of the encoder which takes the original input image and converts it into the latent space which is then fed to the decoder for reconstruction of the original image. The encoder architecture implemented consists of filters of a 2D convolution layer along with the activation function LeakyReLU and the output is flattened in order to construct the vector in latent space. The decoder then takes the latent space input and filters it in the reverse order using a transpose of the convolution layer. The output of the decoder is the reconstructed image which resembles the input image. In case of finding anomalies, the error of reconstruction between the input image and reconstructed image at the output is calculated to decide if it is an anomaly or not.Taking the autoencoder architecture described above, an autoencoder model is trained for 25 epochs and Figure 23 describes the training history.

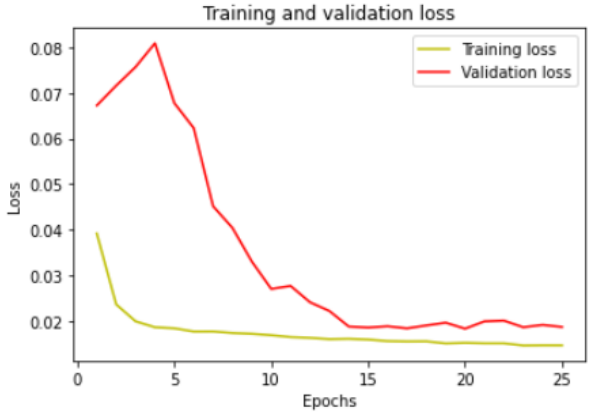

Figure 23: Autoencoder training history

From the above training history, it can be understood that the loss in the training and validation process almost converges by the end of 25 epochs and the model could be fitted and trained correctly. However, this could be understood only when we see and test the reconstruction of input images. In order to test this, the model was fed with the validation images and the reconstruction was observed as shown in Figure 24

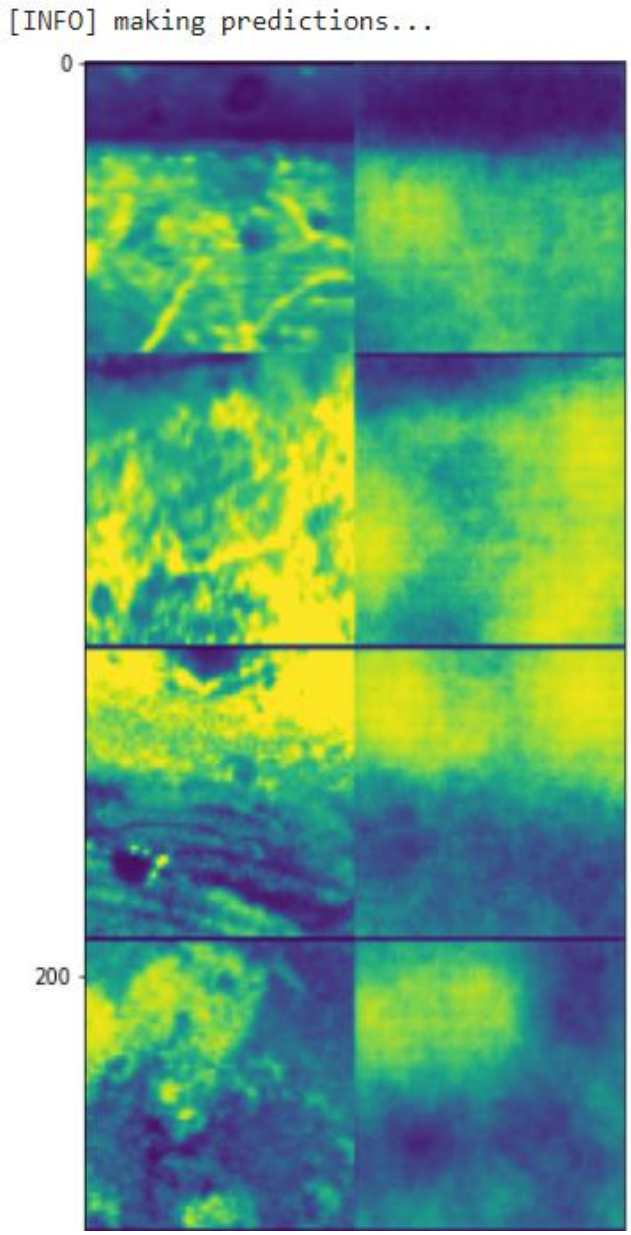

Figure 24: Reconstruction of the input images and prediction

It can be seen from the reconstructed images that the autoencoder model does well in reconstructing the input images. Now, to identify if the input image is a good image or an anomaly, the reconstruction for all the images is calculated and the distribution of the reconstruction errors are viewed as seen in Figure 25

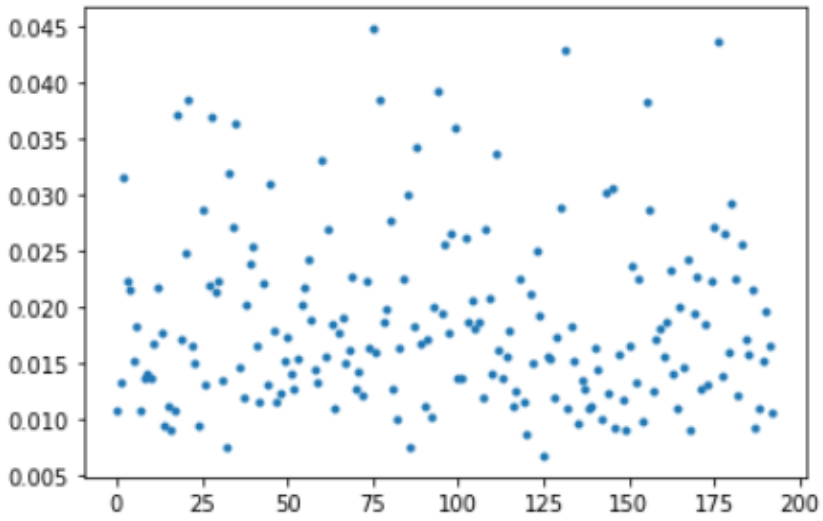

Figure 25: Reconstruction errors

To further segment the anomalies, two methods could be adopted, first giving a manual threshold for the reconstruction error, secondly doing it automatically by defining a certain percentage of the distribution around the mean is good images and the rest are anomalies, and the manual thresholding has been used.

With the manual approach, a threshold of 0.025 was specified, where values of reconstruction error were anomalies. With this approach about 35 anomalies were found as seen in Figure 26

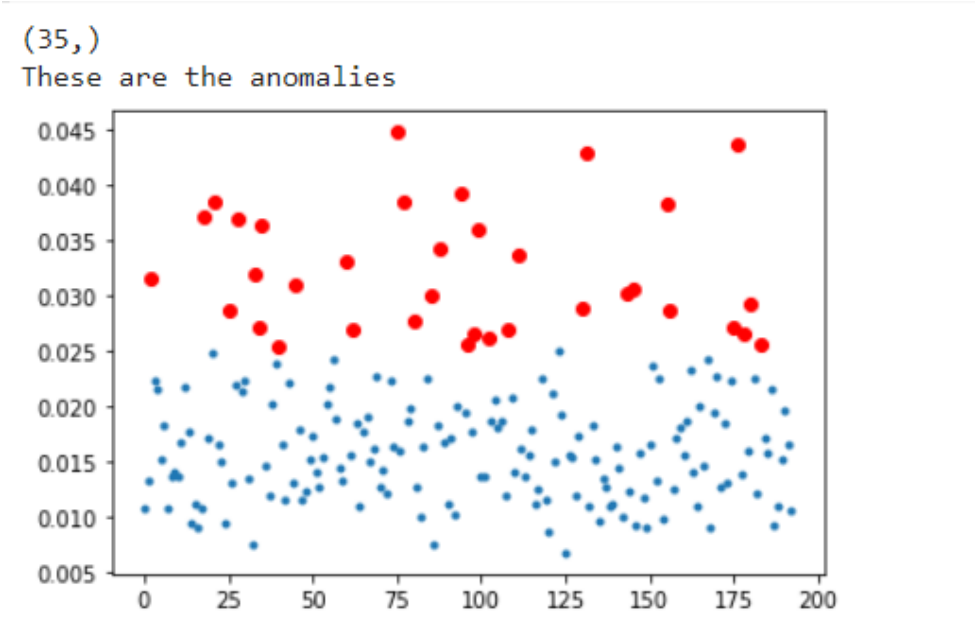

Figure 26: Anomalies identification using manual thresholding

To identify, if the anomalies were correctly detected using this method, the image corresponding to these anomalies were checked and the results are in Figure 27.

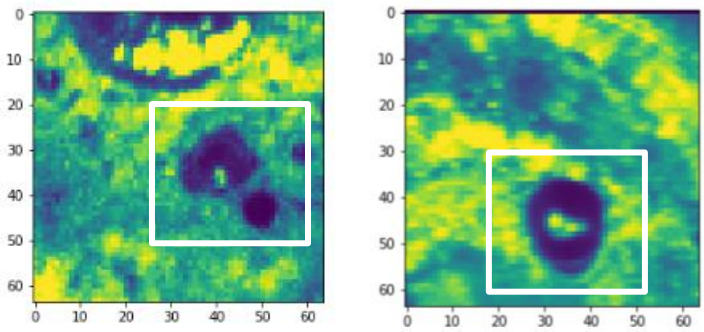

Figure 27: Result of the detected anomalies

From all the anomalies checked, almost all of the detected anomalies were identified truly as defective welds and hence it can be said that the model worked well in detecting the anomalies in the validation dataset.

7. DETECTION OF WELD PORES USING FULL SIZE IMAGE IN UNET ARCHITECTURE

In this part, we analyze the UNET architecture for full image which has been resized to 256x256 pixels. In this section, we also take support from the resnet34 architecture and ImageNet as the pre-trained weights.

We analyze the location of the weld pores, in this method. Here we can detect the anomaly in the weld bead which in our case is the weld pore/holes. The below figures 28 and 29 show that the UNet model is able to learn about the defects in the weld images. The graphs converge and remain in a steady state. The validation data is also able to get a good IOU score

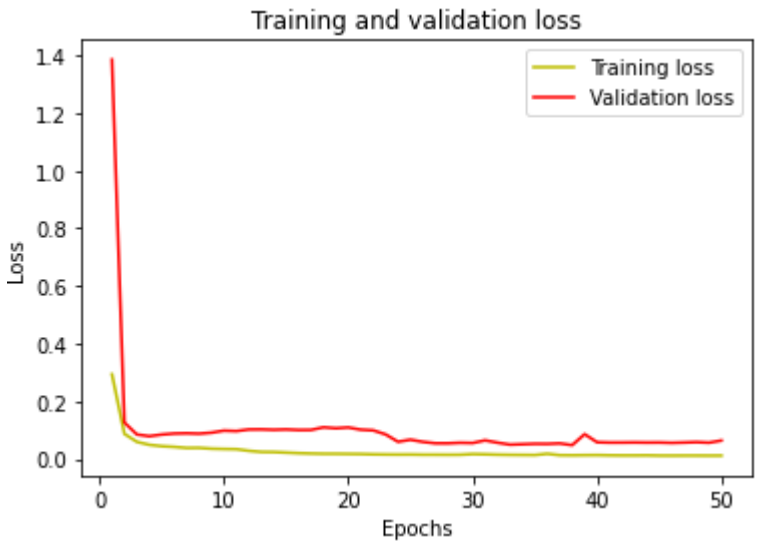

Figure 28: Training and validation history for Full Image UNET Model

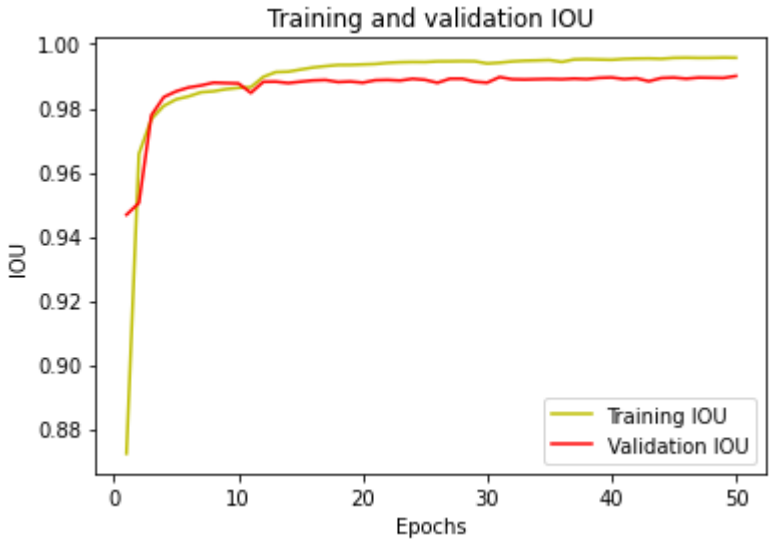

Figure 29: Training and validation history of IOU for Full Image UNET Model

The below figures in Figure 30. show that the UNet model is able to identify the welding defects and their appropriate position. Though the size of the weld defects is decreased in the predicted images, it is quite accurately able to detect it with precision.

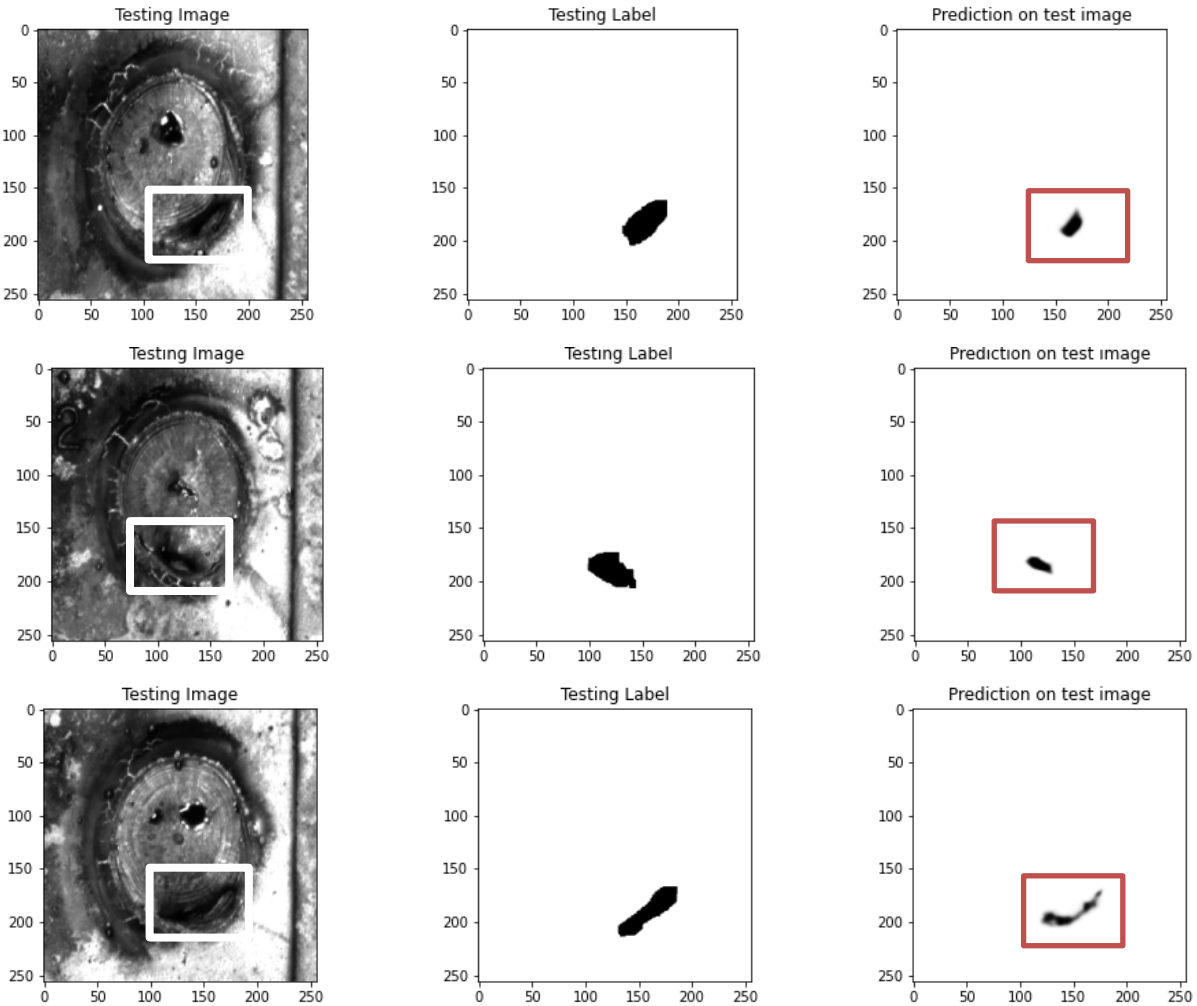

Figure 30: Weld Defects Detected from the image using UNET model

8. CONCLUSION

In conclusion, for a complex problem like inspection of welds image segmentation is a difficult task and so is anomaly detection due to the nature of the problem. In addition, this task dealt with a dataset of a few thousands of 64x64 size images and about 171 full size images, which could be less in training and testing a full-size operational model and could be one reasons of overfitting in the basic convolutional neural network-based model for image segmentation. On the contrary, the segmentation using the UNET for the same number of images proves to be highly efficient and accurate in the prediction.
In the case of anomaly detection, the autoencoder architecture proves to be highly accurate in finding the images with higher reconstruction error that are called anomalies and as seen, it predicts anomalies with utmost accuracy. UNET architecture, in similar lines, is highly accurate in detecting anomalies and abnormalities. The use of pre-trained weighted models like Resnet 34 has therefore been proven to be of great use in helping the model learn and predict better.

9. REFERENCES

[1] Segmentation Models' documentation, version 0.1.2;
https://segmentation-models.readthedocs.io/en/latest/tutorial.html

[2] Kaggle datasets , ResNet 34 Pre-trained Model for PyTorch;
https://www.kaggle.com/datasets/pytorch/resnet34

[3] TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation, Iglovikov, Vladimi, Shvets, Alexey, 2018/01/17
[4] Deep learning segmentation of hyperautofuorescent fleck lesions in Stargardt disease - Jason Charng, Di Xiao, Maryam Mehdizadeh], Mary S. Attia, Sukanya Arunachalam , Tina M. Lamey, Jennifer A. Thompson,Terri L. McLaren, John N. De Roach, David A. Mackey' Shaun Frost .& Fred K. Chen
[5] Deep Learning-Based Weld Spot Segmentation Using Modified UNet with Various Convolutional Blocks - Oskar Natan , Diyah Utami Kusumaning Putri - Universitas Gadjah Mada , December 2021.