STEEL DEFECT DETECTION

Introduction:

Steel is one of the most important building materials of modern times. Steel buildings are resistant to natural and man-made wear which has made the material ubiquitous around the world. Serverstal, is a Russian Steel producing company, looking for machine learning approach to improve automation, increase efficiency, and maintain high quality in their production. The production process of flat sheet steel is especially delicate. From heating and rolling, to drying and cutting, several machines touch flat steel by the time it's ready to ship. This project will help engineers improve the algorithm by localizing and classifying surface defects on a steel sheet.

Problem Statement:

The objective of this project is to predict the location and type of defects found in steel manufacturing using the images provided. The images are named with a unique ImageId, and our task is to segment each image and classify the defects in the test set.

Dataset:

Data is taken from the Kaggle challenge, link: https://www.kaggle.com/c/severstal-steel-defect-detection/data. which is around size ~2GB and consists of 18076 images. There are 12568 images in the train set. 5506 images in the test set of the training 12568 images, only 7095 images have defects, rest have no defects. Each image is of size 1600*256. The location of the defects we given as encoded pixels. We have to decode them using a mask function to get the defect location.

Data Description:

Each Image may have no defects, a defect of a single class, or defects of multiple classes (ClassId = [1, 2, 3, 4]). Given a image, our task is to classify the defect and locate the segmentation of the defect. For each image you must segment the defects if it belongs to each of the class (ClassId = [1, 2, 3, 4]).

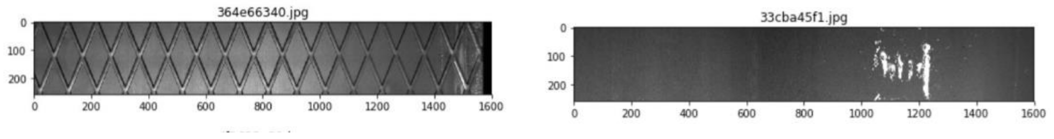

Fig 1: left image with no defect and right images with defects

Since all images are in the grey scale just by visually can not identified defects in them. Hence Using Machine learning and Keras Application like CNN, Resent they are classified and Where exactly defect is present is seen by Unet model.

Task Performed:

We have performed 4 tasks in this project for classification and object detection:

1. **Exploratory Data Analysis**: Data cleaning finding the defects types finding the

2. **Binary classification problem**: Identify whether an input images has a defect or no defects.
3. **Multi-class Classification problem**: Identify the number of class type of defects an image contains, each image can have up to 4 types of defects corresponding to classes [1,2,3,4]
4. **Object Detection**: For each defect type, identify regions of the image that has this defect. This is an image segmentation problem, where each pixel is labeled as having no defect or a defect of a particular type.

Exploratory Data Analysis:

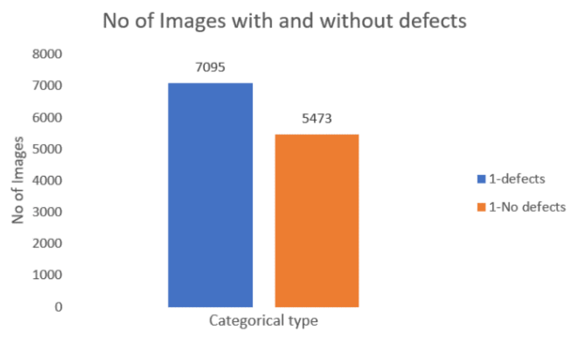

Fig 2: Binary classification

We performed EDA, using bar chart we classified images which has defect and with no defects. From the whole dataset 5473 images with no defects were labeled as **0** and 7095 images with defects were labeled as **1.** The number of images with and without defects are proportionate and the data seems quite balanced. From the fig 1, we concluded that data is balanced for doing binary classification.

Now we will plot the bar graph to see class type of defects each image has, there are 4 types of defects **Type1** defect which conation small seed like scratch or crack on the steel sheet. **Type2** defect which has thin vertical crack on the images. **Type3** multiple vertical line cracks on sheet and Type4 has medium area size defects in images, In the segmentation plot we can see it more clearly the defects.

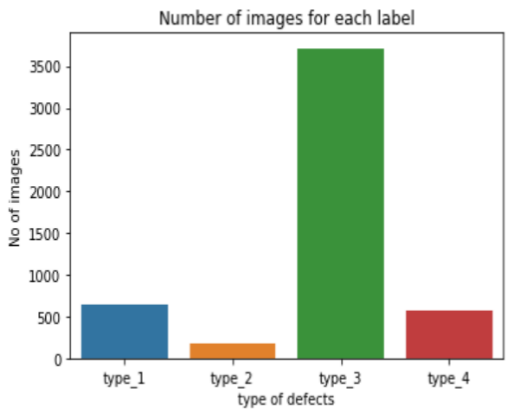

Fig 3: Multiclass classification

Categories of defects – Type1, Type2, Type3, Type4. From that Lot of images with defects belong to Type3. Very few samples for the type2 defects data is very imbalance here. So, to balance the data we tried data augmentation has performed for data augmentation resampling techniques also used and also tried SMOTE method to balance the data. Batch normalization had performed best compared to other methods.

EDA was also performed to check whether there are multilabel image, some of the images had more than one type of defects in them. Use bar graph these images were also classified.

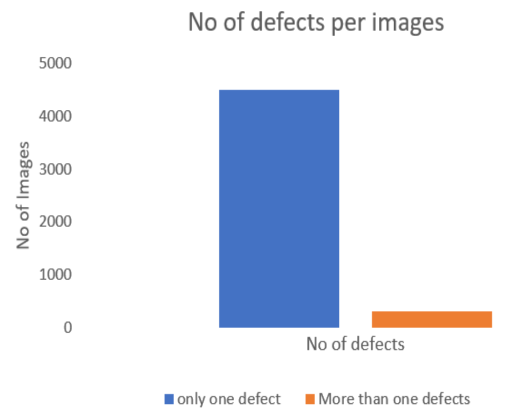

Fig 4: Multilabel classification

These images were classified as multilabel, since they were very few in number. We removed from the data. Now our data is resized and ready to feed into neural network for binary classification and multi class classification model.

EDA conclusion:

The dataset is imbalanced thus we will use stratified sampling for splitting the dataset into train and validation datasets. This is a multi-label image segmentation problem. As there are around 50% of images with no defects, it is equally important to identify images with no defects. Convert masks to EncodedPixels and filter them as per classification probabilities.

Segmentation:

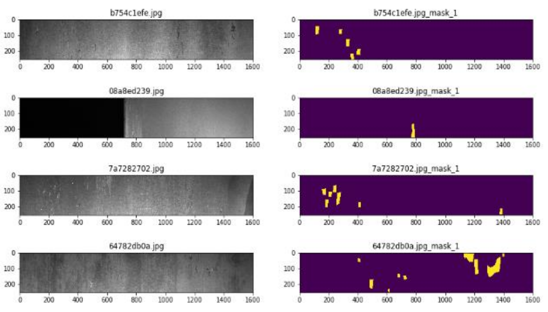

Fig 5a: Type 1 defects

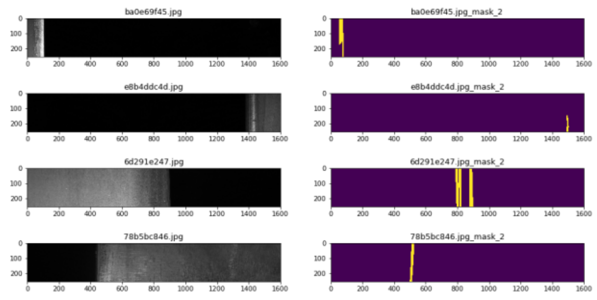

Fig 5b: Type 2 defects

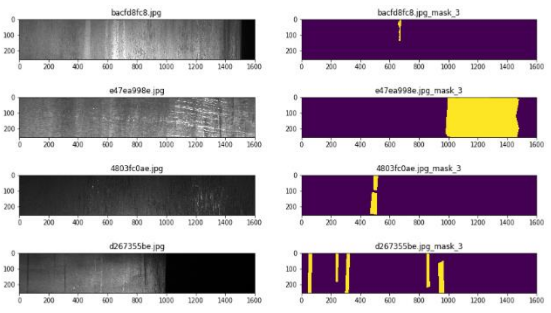

Fig 5c: Type 3 defects

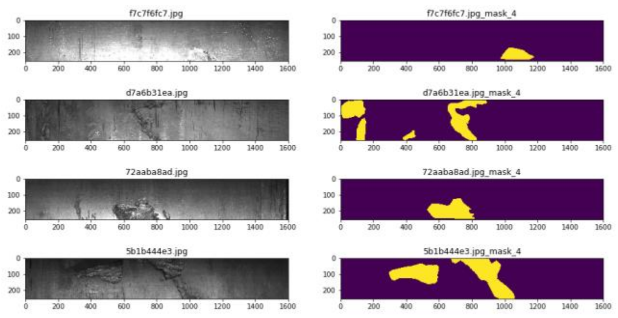

Fig 5d: Type 4 defects

With the help of image segmentation, we can partition the image into multiple segments. This will make it easy for the computer to learn from patterns in these multiple segments. Here, pixel belonging damage or crack on the steel sheet is colored yellow. The regional profile on the masks of defect containing steel surfaces can be seen to be distinguishable among different classes. Though defect type 1 can be seen to have multiple small size regions and defect type 4 images have multiple regions of medium size. Defect type 3 images can be seen to also contain multiple regions of medium size. While defect type 2 and type 3 images can be seen to share some regional characteristics. This method will help for generating predictions

Binary Classification:

To do the Binary Classification, Convolutional Neural Network (CNN) model has been used, as it outperforms all other models in context of images. The architecture of CNN model has been presented in Fig 6. The CNN model starts with 3x3 convolution, with 36 filters in the first layer followed by 2x2 Max Pooling layer, reducing the image size by half. The image with size of 256 is reduced to size of 128 by Max Pooling layer. In the second layer, again there is convolution layer with 3x3 kernel, with 64 filters and so on. At the end of four layers of convolution and max pooling, the final layer is flattened, and feed to the dense layer. The model summary in details has been presented in Figure 4, showing the number of parameters for each layer.

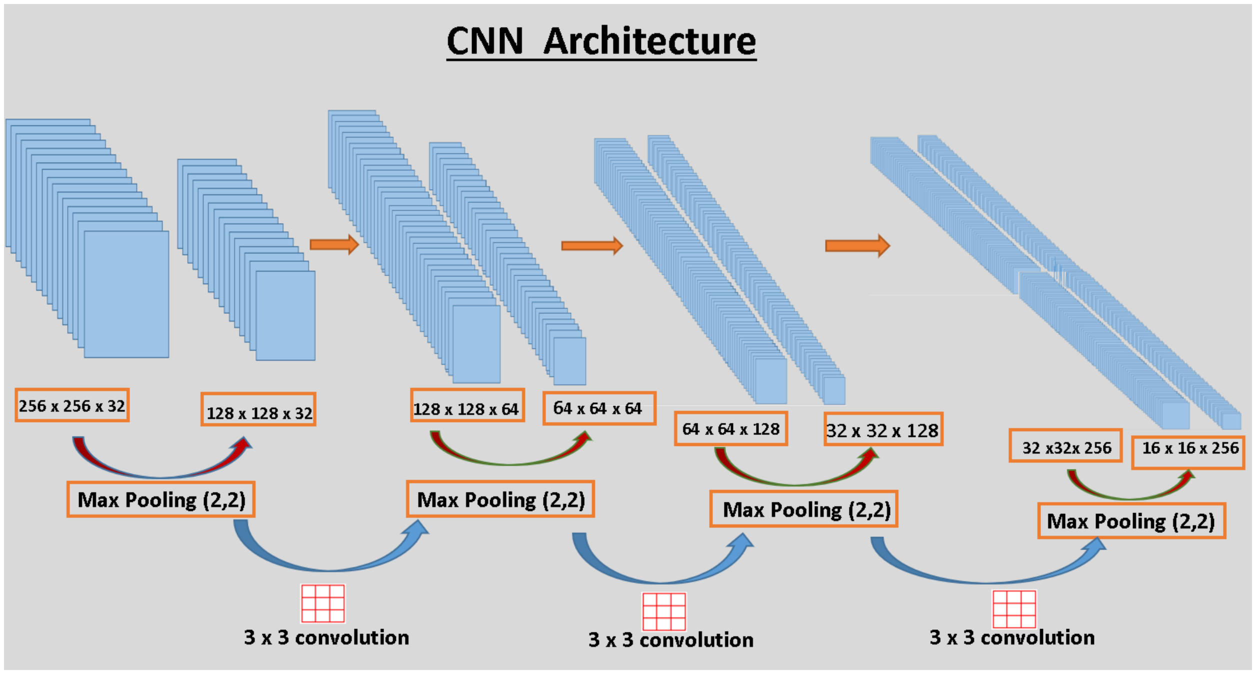

Fig 6: CNN Model architecture

Although the model used four convolution layers and one dense layer, it gives quite good accuracy. In the figure 7a, the loss keeps on decreasing with the increase in number of epochs. The validation accuracy keeps on increasing up to ten epochs, then starts decreasing. The training accuracy keeps on increasing, which was expected with the increase in the number of epochs. The overfitting can be seen here after ten epochs; however, a very good accuracy has been achieved. The validation accuracy of 85% has been achieved. The model can be said to generalizing well.

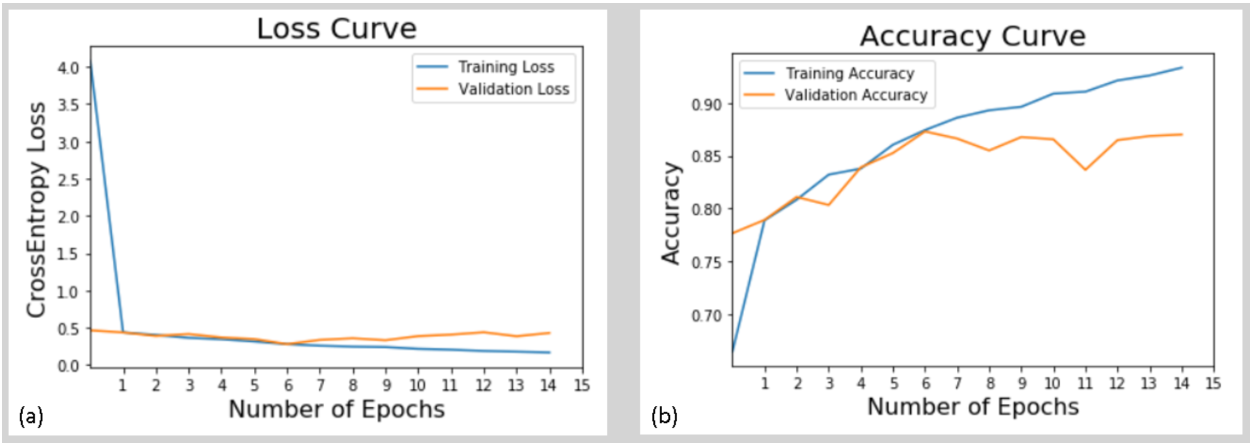

Fig7a: Loss Curve **Fig7b Accuracy Curve**

Confusion Matrix

	True No-Defect	True Defect
Predicted No-Defect	1039	142
Predicted Defect	181	1152

Fig 8: Confusion matrix for test label

The confusion matrix obtained by the model on test-data. as it is able to identify most of the examples correctly. However, there are some wrong prediction, but they are quite low in number. For these types of problems of quality insurance, the problem comes when a defected steel is classified as non-defect steel and is sent to the market. In model, it's identifying 142 defect cases as non-defect cases. It's around six percent of total cases used to create the confusion matrix.

Precision-Recall Matrix

	Precision	Recall
No-Defect	0.85	0.88
Defect	0.89	0.86

Fig 9: Precision-Recall Matrix

The precision recall matrix. The prediction precision is quite high around 85% for non-defect examples, and 89% for defect examples. But recall is of more interest, as we wants to see how accurately, it's able to recall the steel type as defect or non-defect. The recall percentage for non-defect and defect cases are 88% and 86% respectively.

The CNN model is generalizing quite well. It's able to identify the defect steel with a very good accuracy. However, the performance could be increased with regularization and more tuning the model. For the time being, the model performance is quite well.

Multi Class Classification:

In this section we built a multi-class classifier to classify the type steel defect shown on an image. To mitigate the class-imbalance, during training we will augment the images and for every batch augmented we will over sample the minority classes to have better representation. To evaluate the performance of the model, we expect the precision, recall and f1 scores to be similar in training, validation and testing datasets. Class number 3 is approximately 75 percent of the whole dataset. In other words, just by predicting class 3 the model can achieve an accuracy of 75 percent. Therefore, we need the model to generalize among each class as much as possible.

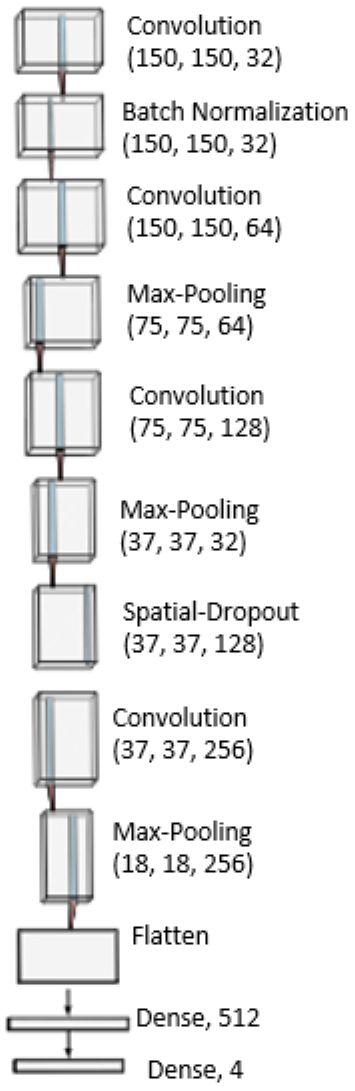
Convolution (150, 150, 32)
Batch Normalization (150, 150, 32)
Convolution (150, 150, 64)
Max-Pooling (75, 75, 64)
Convolution (75, 75, 128)
Max-Pooling (37, 37, 32)
Spatial-Dropout (37, 37, 128)
Convolution (37, 37, 256)
Max-Pooling (18, 18, 256)
Flatten
Dense, 512
Dense, 4

Fig 10: Multi-Class Classifier Architecture

The images were preprocessed into a shape of 64 by 64 to a gray scale, as a way to reduce the number of features. To reduce the memory footprint, to train the model we will use an image generator from Keras. As mentioned above, an additional piece of code was added to balance the image generated during training. To augment the training data, the image generator will flip the image horizontally and vertically, and also tweak the brightness from a range of 02. to 0.8. To ensure reproducibility, the random number generator was seeded at 42.

To reduce overfitting due to the class imbalance, the model was created with several dropouts. After the first convolutional layer we used a batch normalization layer which improved the learning of the algorithm. The following layers are two blocks of convolution, 10 percent spatial dropout and max-pooling layer, then another convolution and a max-pool layer. The classifier has a ten percent dropout with 512 neurons in the first dense layer and 4 neurons in the output layer. This model architecture gives a total of 5,109,060 trainable parameters. Due the image data generator will act randomly, to enhance the learning and avoid learning cycles the optimizer used is stochastic gradient descent. The loss function is categorical entropy and the target metrics is accuracy. The activations functions for the convolutional and the first layer was Relu and SoftMax for the output layer.

Training convolutional models takes time and sometimes after a few epochs it can start overfitting. One strategy we used was to record the best weights after every epoch based on the accuracy of the validation. Additionally, we added early stopping and reduce plateau to monitor the model's learning and stop the process when the model has already learned enough.

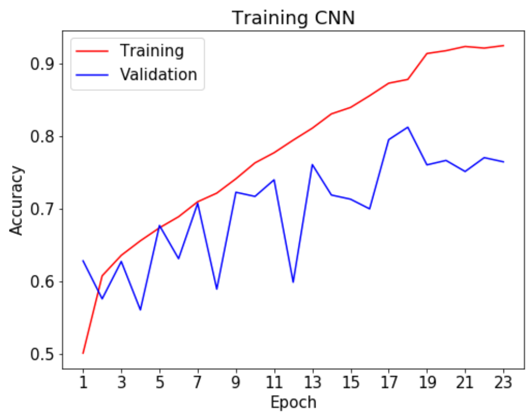

Fig 11: Accuracy per Epoch during training

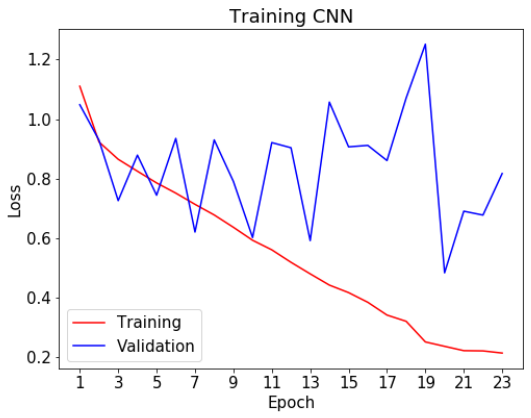

Fig 11b: Loss per Epoch during training

Initially for every epoch the accuracy of the validation increased while its loss was not stable. After 20 epochs the accuracy of the training and validation plateaus out but the loss in the validation does not show a significant change as in the training set. This trend in training and the confusion matrix and classification report of the trained model shows the model does generalize for class 1, 2 and 3 but it enables to properly classify class 4

Training Classification Report	Precision (%)	Recall (%)	F1-Score (%)
1	68	70	69
2	80	70	75
3	89	91	90
4	47	37	41
Accuracy			83
Macro Avg	71	67	69
Weighted Avg	82	83	83

Fig 12a: Classification Report for Training Data

Validation Classification Report	Precision (%)	Recall (%)	F1-Score (%)
1	43	47	45
2	50	47	49
3	84	85	84
4	12	10	11
Accuracy			74
Macro Avg	47	47	47
Weighted Avg	73	74	73

Fig 12b: Classification Report for Validation Data

Testing Classification Report	Precision (%)	Recall (%)	F1-Score (%)
1	54	49	51
2	48	63	55
3	84	88	86
4	23	13	17
Accuracy			76
Macro Avg	52	53	52
Weighted Avg	74	76	75

Fig 12c: Classification Report for Testing Data

Training convolutional neural networks with a small unbalanced dataset is not an easy task. After implementing several strategies to build the model, due to the given dataset, the model overfits. Ideally, a larger dataset with more class representation would do the work. Another option would be to try and drop class 4, this might decrease the overfit and improve the validation scores.

Defect Detection

We perform the task of defect detection through segmentation i.e., we label every pixel of the input as no defect or defect of a specific type. To perform image segmentation, we use the U-net architecture. This architecture was first proposed for biomedical image segmentation. The main idea is to supplement a usual contracting network by successive layers, where pooling operations are replaced by up-sampling operators. Hence these layers increase the resolution of the output. It is also a popular architecture in a class of DNNs called Fully-Convolutional Nets (FCNs).

The FCNs or U-net comprises of 2 parts:

- The first part is similar to image classification CNNs, that have a sequence of CONV/Relu/MaxPool layers
- In the second part, instead of classification layers, transpose convolution is performed to gradually upsize the features and get the final segment map for each class. Transpose convolution also uses feature from the first part of the model when up-sampling the image in order to preserve the resolution.

This gives the U-shape to the architecture. Model Architecture specifics

The U-net model used for this comprises of 33 layers of which, 23 are convolution (Conv2D) layers, 5 are Maxpool layers and 5 are transpose convolution (Conv2DTranspose) layers. The total number of trainable parameters int the network is 600,612.

The U-net model used for this project is shown below.

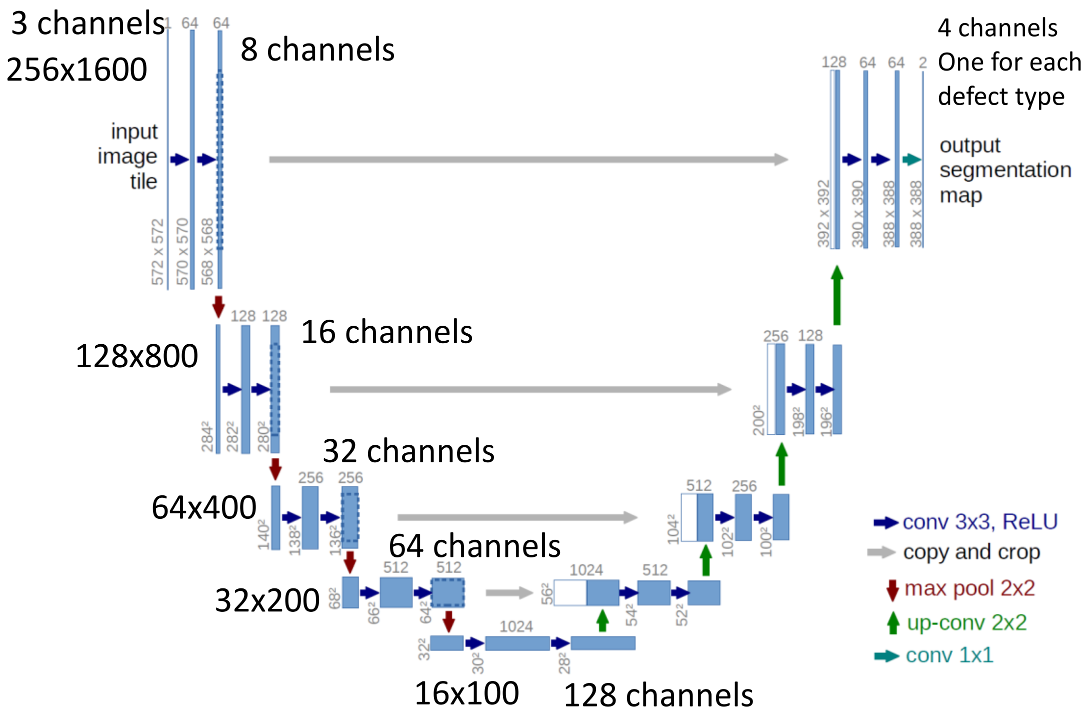

Fig13: U-net architecture

We pass an image of size 256*1600 with 3 channels as the input to the model. The annotations on the side shows how the image size decreases and the number channels increases as we pass through the maxpool layers. In the second half of the network, the image is up-sampled by using transpose convolution and we finally get an output segmentation map for each image with 4 channels, one for each defect type.

Results:

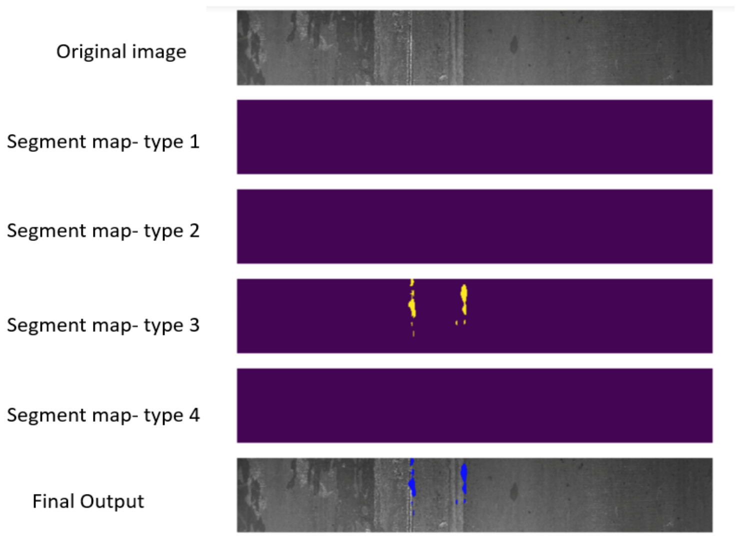

Fig14: Segmentation maps

The output segment maps from the model for a particular image is shown below.

From the segment maps we can understand that, this image has a defect of type 3. The defect region is masked using a masking function based on the location of defect given to us. The final output is formed by super-imposing the 4 segment maps on top of the original image and color coding the defects based on its defect type.

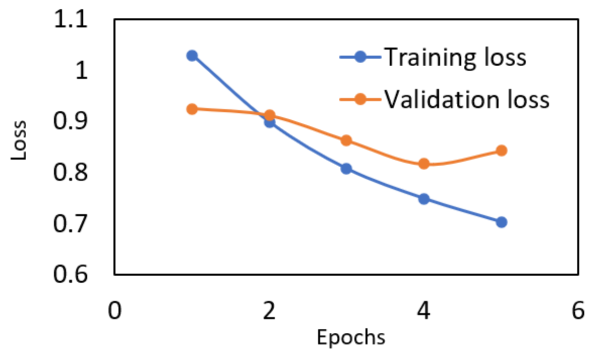

Fig15: Training and Validation Loss

The training and validation loss of the model is showing below as a graph. The training loss and validation loss decreases as the number of epochs increase. The graph shows the values for the losses for 5 epochs**.**

Our model was able to clearly distinguish between images with no defects and images with defects. For the images with defects, the model was able to detect the defect regions well, but it classified the defect as defect of type 3.

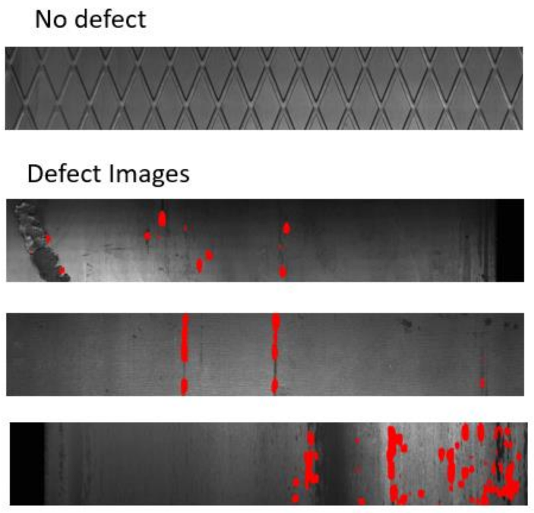

Fig16: Prediction on test images

This is mainly because of the class imbalance in the dataset. color coded the defect classes as red for type 3, green for type 4, blue for type 1 and yellow for type 2. Few output samples from the predictions on the test dataset is shown below.

Since the model was taking very long time to train, we could not try training the U-net model with the augmented images. Instead, we used a subset of the data with more balanced images in each class and trained the u-net model. The model little better results but again the model was biased towards the defect class with a greater number of images.

Future Work:

We can still improve the model: As there is less similarity between train test data, we can use adversarial Validations to improve performance. Use other data Augmentation techniques for both train and test Augmentation.

Conclusion:

Thus, we were able to classify and localize defects on steel sheets using deep learning techniques. Further, we would like to do more data augmentation and fine tune our models to get better accuracy.

References:

[1]. A Steel Surface Defect Recognition Algorithm Based on Improved Deep Learning Network Model Using Feature Visualization and Quality Evaluation Guan, Shengqi & Lei, Ming & Lu, Hao. (2020). A steel surface defect recognition algorithm based on improved deep learning network model using feature visualization and quality evaluation. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2979755.

[2]. U-Net: Convolutional Networks for Biomedical Image Segmentation. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

[3]. Real-time Detection of Steel Strip Surface Defects Based on Improved YOLO Detection Network. Li, Jiangyun, et al. "Real-time detection of steel strip surface defects based on improved yolo detection network." IFAC-PapersOnLine 51.21 (2018): 76-81.