APS360 Final Project Report

Detection, Classification, and Segmentation of Steel Surface Defects

Penalty: 0

Name	Student Number
Farhan Wadia	1003012606
Zhipu Zhang	1002653979
Wenju Zhang	1002285252
Hiranya Maharaja	1004241037

Table of Contents

Introduction

This project's goal is to develop a system to detect, classify, and segment steel defects within an image. Due to how commonplace and essential steel is for maintaining structural integrity, it is imperative that steel is defect-free; damaged steel can lead to failures that endanger lives. Improving steel defect detection (SDD) can be applied in numerous industries, but it remains a challenge because defects never look exactly alike.

Deep learning (DL) is promising for SDD because of data quantity, analysis speed, and potential difficulties in data access. In many scenarios, an imaging system is a prerequisite to obtaining data on defective steel because of safety issues facing human operators, meaning the data comes in a form conducive to applying DL. DL methods are faster than manual analysis, and don't get fatigued.

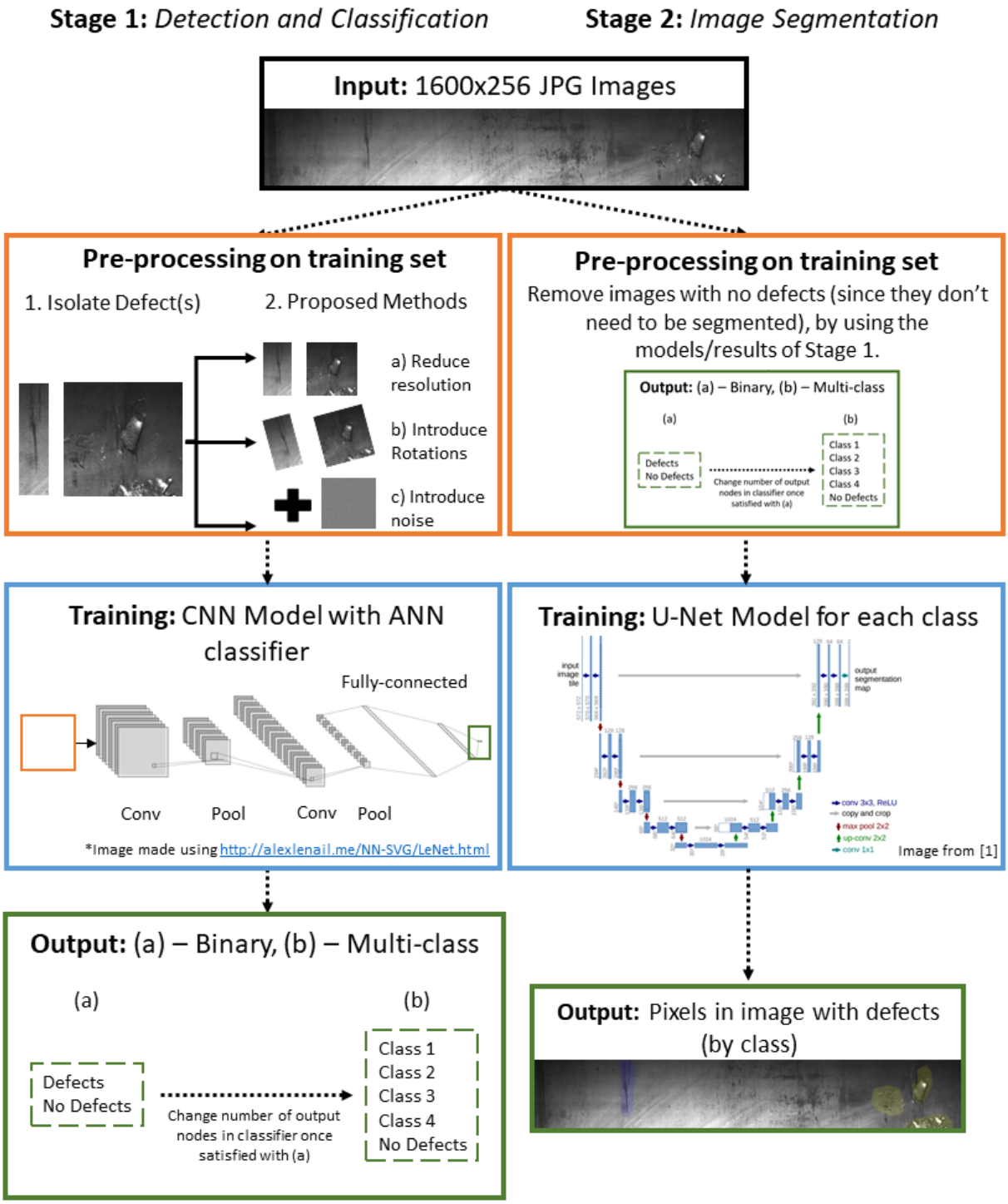
Background Work

One well-known architecture for segmentation is U-Net. It is similar to an autoencoder; a path of convolutions and max-pooling learns lower-level features and transpose convolutions upsample them into a segmentation map. While upsampling, features from the encoder on the opposite side of the "U" are copied and concatenated to help the model localize better. U-Net performs better than older models, having won the 2015 ISBI cell tracking challenge on the DIC-HeLa dataset with an intersection-over-union accuracy of 77.6%, compared to the runner-up at 46% [1].

The main problems with employing CNNs for SDD are that complex-looking defects are not classified well by CNNs, and generalizability is low. Due to time, process, and financial constraints, labeling training data correctly is nearly impossible. To meet industry requirements, an unsupervised approach is necessary. One solution is the CAE-SGAN; convolutional autoencoders learn lower-level features from unlabelled defects, which trains a GAN where the generator creates new defects, and the discriminator classifies them [2].

Illustration

Figure 1 - Illustration



Data Processing

12568 steel images and ground-truth run-length-encoded pixel defects for four defect classes were obtained [3]. We randomly split this 70-15-15 into train, validation, and test sets. Another separate dataset of 1801 images without ground-truths was used to further validate the model's performance.

Table 1 - Number of images with defects by class

Class	Number of images
Class 0 (no defects)	5902
Class 1	897
Class 2	247
Class 3	5150
Class 4	801
Total (>12568 because some images belong to multiple classes)	12997

Figure 2 - Images by class percentages

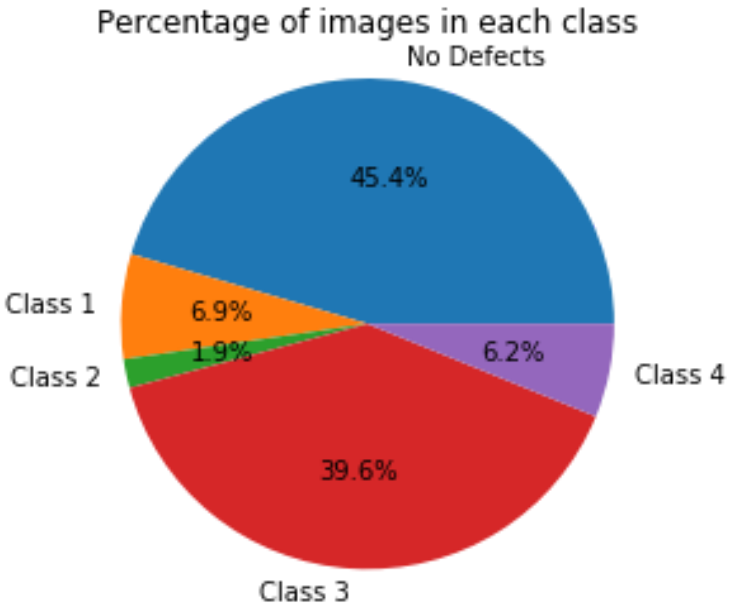
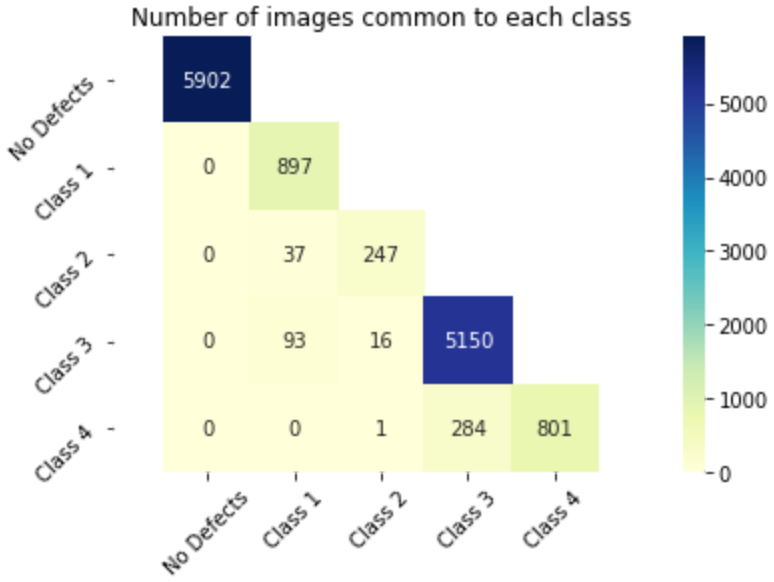


Figure 3 - Images common to two classes



A custom dataset class was written to load the segmentation run-length-encodings (RLEs) with their images, convert the RLEs to masks, and to apply augmentation transformations. The following transformations are done on the images and masks in the training and validation sets:

- Resized from (256,1600) to (96, 576).
    - This resize was chosen because:
        - 1) both 96 and 576 are divisible by 32, a necessary condition for the ResNet 18 portion of the model
        - 2) 96 x 576 images have a similar amount of pixels as an image AlexNet uses (224^2)
        - 3) aspect ratio of 1600/256 is near 576/96
- Randomly flipped horizontally and/or vertically, and/or rotated ±5° approximately 25% of the time
- Converted to tensor

Figures 4 and 5 show an example verifying that the transformations are applying correctly. Figure 4 shows an image from the training set with its masks for classes 1 (red) and 3 (blue). Figure 5 shows this same image and the masks after transformations are applied. Throughout this document, red, green, blue, and yellow are used to show classes 1, 2, 3, and 4 respectively.

Figure 4 - Example

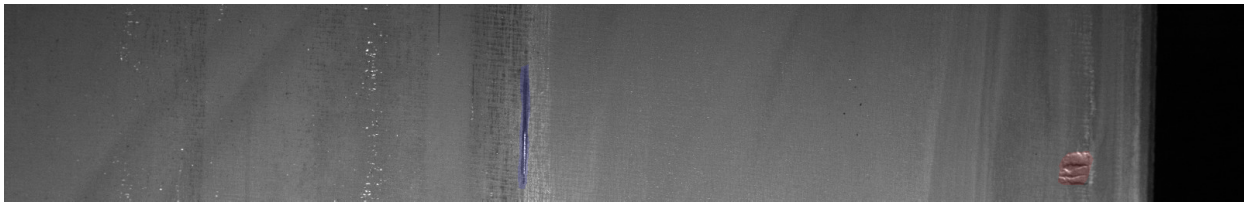
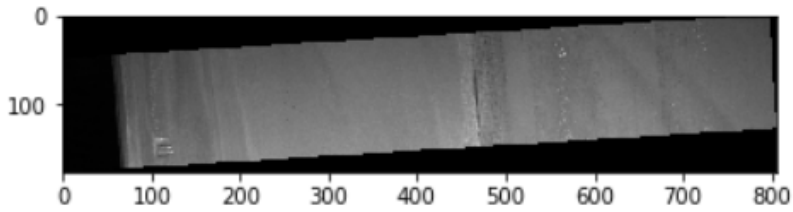


Figure 5 - Example with transforms

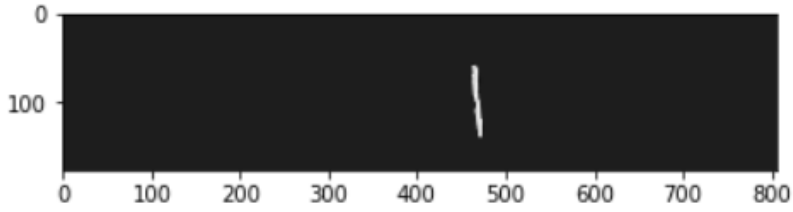
```
[23] # a known image with defects in class 1 and 3
     image, masks = train_dataset[4137]
     plt.imshow(image.numpy(), cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7fbc22e21e48>
```



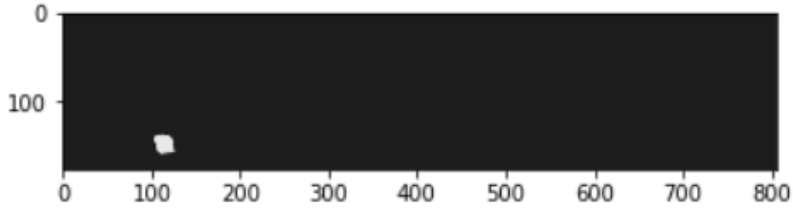

```
[24] #verify that class 3 map is transformed correctly
     plt.imshow(masks.numpy()[2], cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7fbc22de1cc0>
```




```
[25] #verify that class 1 map is transformed correctly
     plt.imshow(masks.numpy()[0], cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7fbc22da3748>
```



Architecture

The final architecture (Figure 6) is a fully-convolutional network based on U-Net incorporating transfer learning from a pre-trained ResNet 18.

The input to the model is a Nx3xHxW tensor, where N is the batch size, 3 is for RGB channels, H is an image's height, and W is its width. The model outputs a Nx4xHxW tensor of one-hot segmentation masks for the four classes.

The input tensor is first passed through a pre-trained ResNet 18 model. The resulting features are then passed through our implementation of U-Net.

The downsampling stage consists of four layers. Each "layer" consists of the following steps:

1. A 2D convolution where the number of output channels equals the number of input channels, a kernel size of 1, and no padding
2. ReLU activation
3. Max pooling with kernel size 2, stride 2

Between each layer, the number of output channels is doubled after applying the max pool, but before the 2D convolution. In the first layer, the 3 channel input is mapped to a 64 channel output; thus, after the 4th layer, there are 512 feature maps.

The next stage consists of four "layers" of upsampling. Using nn.Upsample instead of nn.ConvTranspose2D, the last output from the downsampling stage is upsampled to double its height and width. This upsample is then concatenated with layer 3 from the downsampling stage (resulting in 256+512 channels). A 2D convolution of kernel size 3 with zero padding is then done on this tensor, and the number of output channels is set to 512.

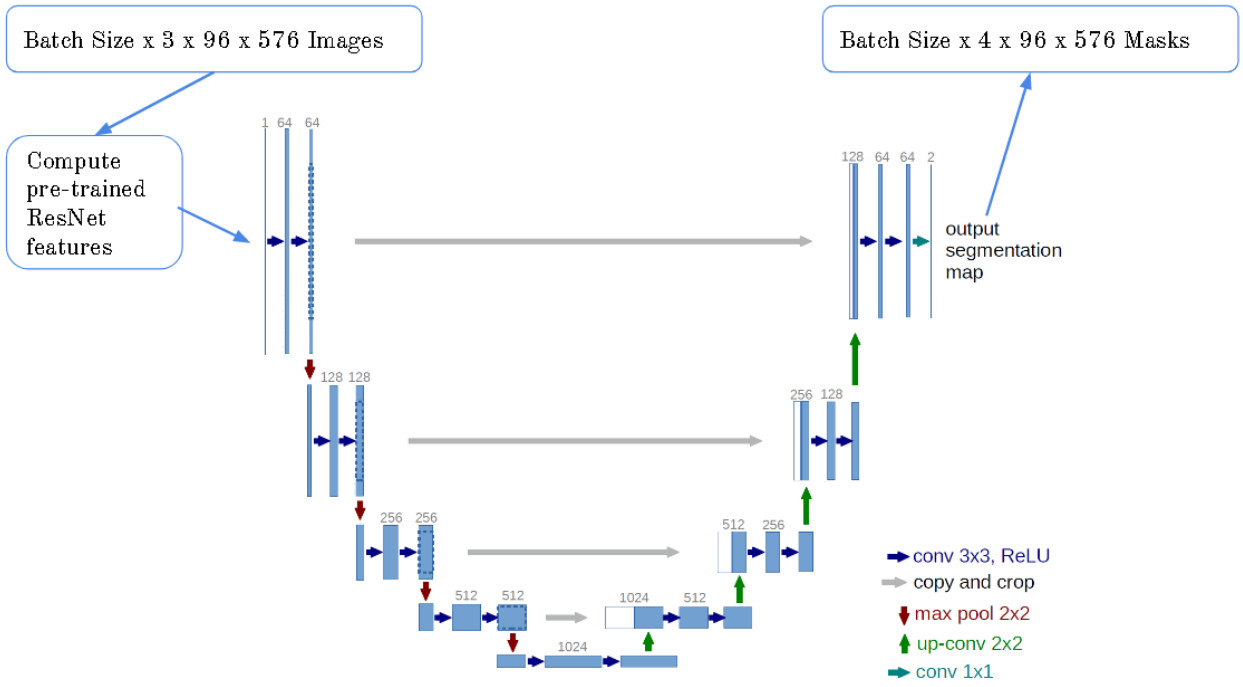
This process is repeated three more times, bringing the tensor height and width back to the initial. Table 2 shows the number of input and output channels of this process. The input is shown summed, where the left side is the number of channels from the downsampling layers being concatenated with the most recent upsampling output (right side). All these convolutions have kernel size of 3 and zero padding.

Table 2 - Convolution parameters for upsampling layers

Layer	Channels	
	Input	Output
1	256+512	512
2	128+512	256
3	64 + 256	256
4	64 + 256	128

Finally, one last 2D convolution with four output channels is used to get the tensor to have the same number of channels as the number of defect classes.

Figure 6 - Model architecture



Baseline Model

The baseline model is a SVM from SKLearn. This is implemented and evaluated as follows:

```
BaselineSVM = svm.NuSVC(gamma = 'scale')
BaselineSVM.fit(imgs, floatLabels)          #Fitting
BaselineSVM.score(imgs, floatLabels)        #Accuracy
```

Accuracy is calculated based on average batch accuracy. SVM performance degrades with larger input size. Batch size was set to 64 to achieve a balance between performance and realism.

Quantitative Results

Accuracy is computed using the dice coefficient (see Figure 7), which weighs sensitivity and precision together, penalizing false predictions and rewarding true predictions.

Figure 7 - Dice Coefficient Expressions

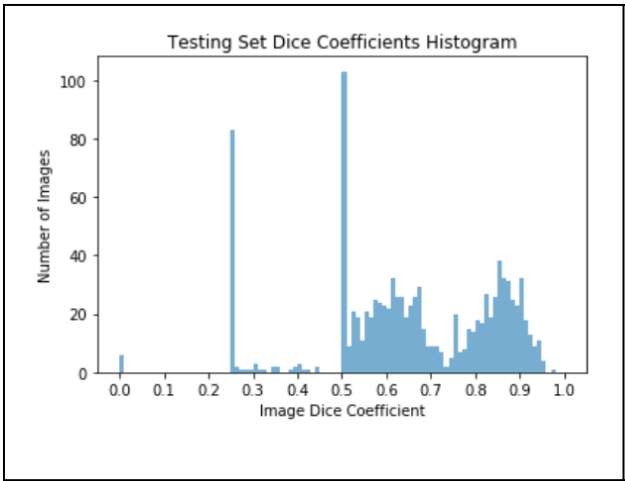
$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

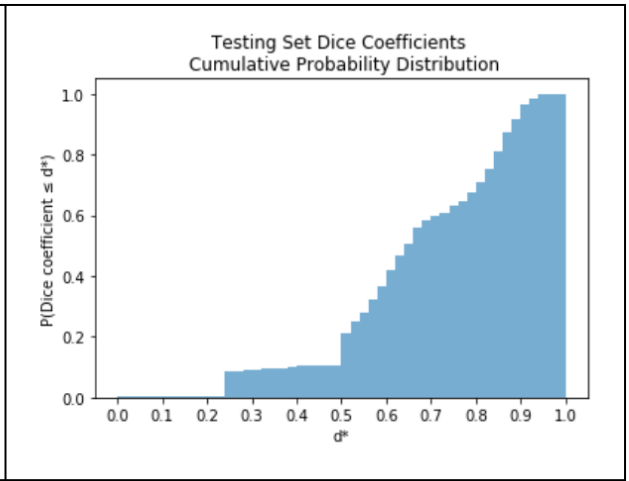
$$\frac{2 \cdot (\text{number of true positives})}{2 \cdot (\text{number of true positives}) + \text{number of false positives} + \text{number of false negatives}}$$

Table 3 - Accuracy Results

Set	**Baseline SVM** (Batch Average)	**Our Model** (Dice Coefficient Accuracy)
Validation	72%	81.2%
Test		83.2%

Figure 8 - Testing set histogram of dice coefficients calculated on single images and cumulative probability distribution






```
The maximum dice coefficient (for a single image) was 0.9749
The minimum dice coefficient (for a single image) was 1.145e-12
```

Qualitative Results

Defect predictions are predominantly classes 3 and 4. Classes 1 and 2 are not detected well.

For class 3, the model achieves similar results to ground-truths (Figures 9, 10).

Figure 9 - Sample image with ground-truth overlayed

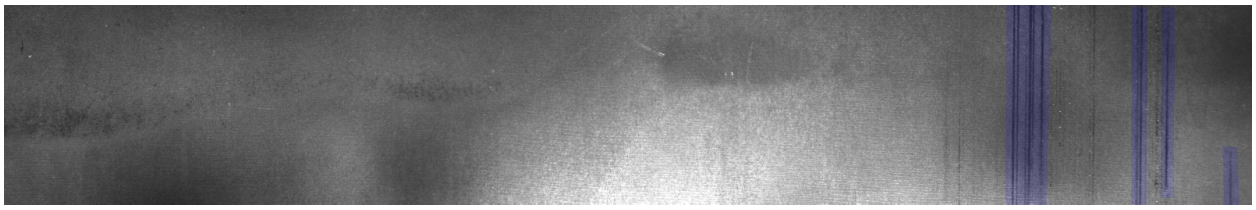
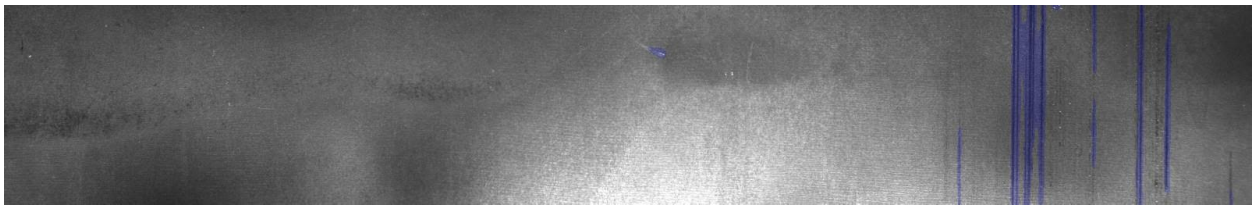


Figure 10 - Prediction on Figure 9's image



Class 3 ground truths are not just vertical lines, but the model segments vertical lines in class 3 well. Figure 11 shows a wide ground-truth beyond what appears to be the true vertical defect, but in Figure 12 showing the prediction, only the vertical lines are segmented. However, we also see false positives for classes 3 and 4 on the right side of Figure 12.

Figure 11 - Wide labelled ground-truth

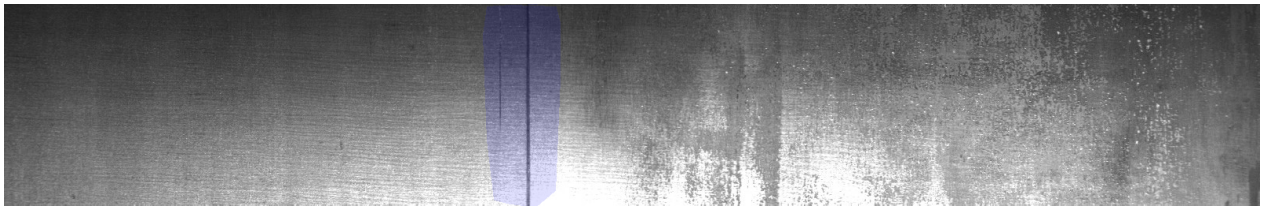
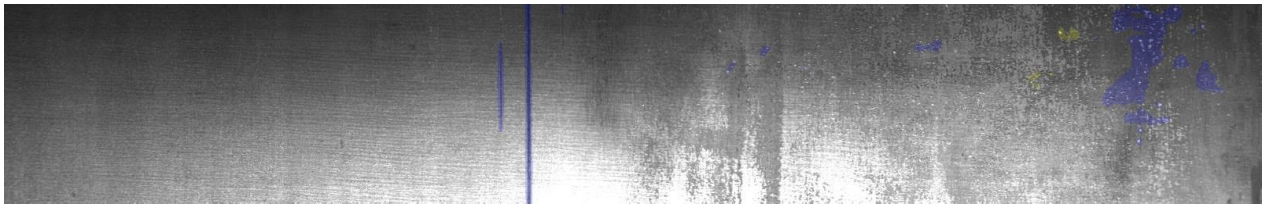
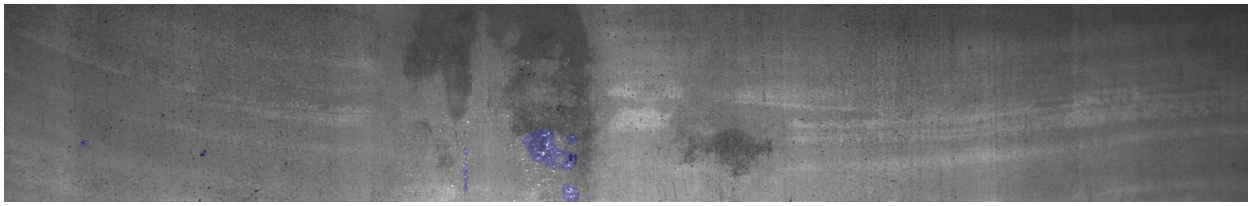


Figure 12 - Prediction on Figure 11's image



We also notice that predictions are not made solely based on shade (relative darkness of each pixel); there is more sophistication. In Figure 13, the darker regions are not predicted to be defects.

Figure 13 - Dark patches not predicted defective



Generally, the model performs well at locating defect edges. This suggests edges play an important role in the prediction process. Potential defects that appear to be contoured are generally predicted as class 4, whereas linear, flat surface faults/scratches with light centers are overwhelmingly class 3 (Figures 14, 15).

Figure 14 - Light interior, linear class 3 defects

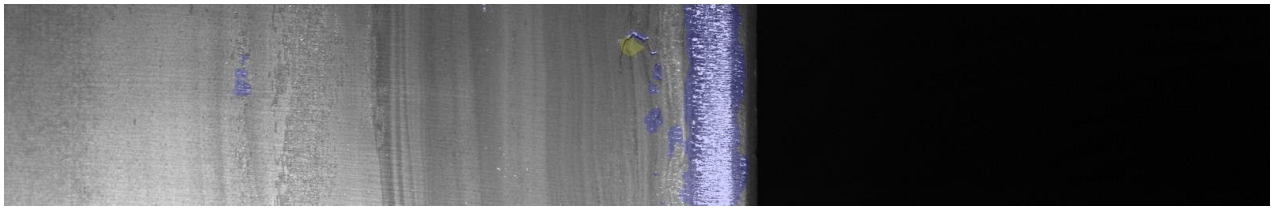
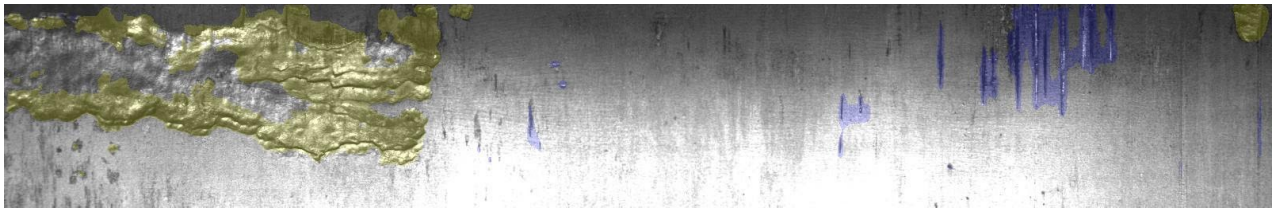


Figure 15 - Contoured class 4 defects



Predictions for classes 1 and 2 were rare, and when predicted, tiny (Figure 16). This would suggest that the model didn't learn to classify the other two fault classes well.

Figure 16 - Tiny class 1 prediction

Evaluation on new data

To validate our model's performance beyond our original test set, we took a second dataset of 1801 images with unknown ground-truths and got our model to make predictions on that. Since these images have unknown ground-truths, they had no effect on our model training or hyperparameter tuning. However, that also means these predictions can only be evaluated qualitatively.

Similar to the other testing set, the predictions are mainly in classes 3 and 4. Class 1 and 2 predictions, if made, are tiny (Figure 17).

Figure 17 - Prediction with class 1 (middle-left), class 3, class 4

Predictions on class 3 seem to be accurate in terms of shape and size, with minor missed regions (Figures 18, 19).

Figure 18 - Prediction with classes 3, 4

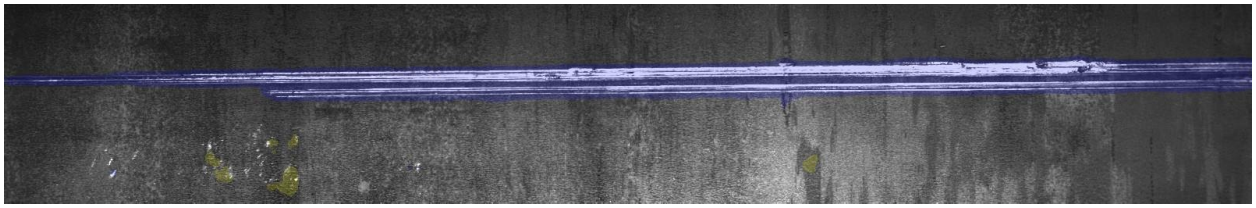
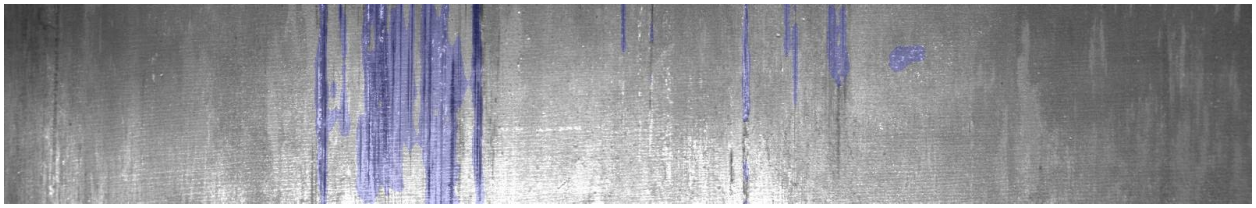
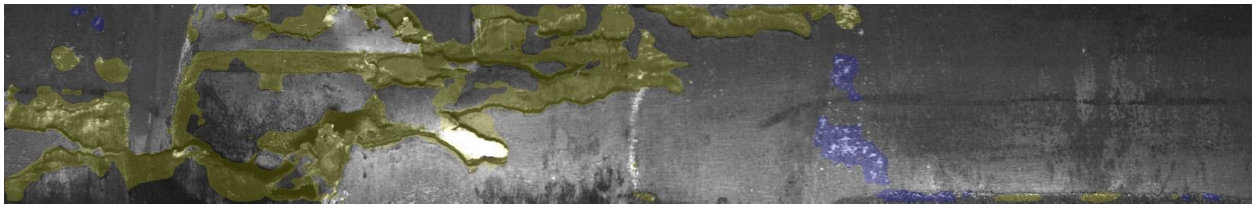


Figure 19 - Prediction with class 3



Class 4 is generally well predicted, but there are several instances where the inner region is missed and only the outer boundaries are segmented (Figure 20).

Figure 20 - Prediction with classes 3, 4



One interesting observation is that non-defective, textured steel are having the textures selected as class 3 and 4 defects (Figures 21, 22). This suggests that the model wasn't adequately trained on images of steel with textures, and didn't learn how to differentiate a texture from a defect. Just like previous observations, these textures are being selected as classes 3 or 4 and not 1 or 2. Looking at the figures, it is difficult to see how the model decided to predict a texture as class 3 vs. class 4, raising doubts about how the model understands and differentiates between these two classes.

Figure 21 - Prediction with classes 3, 4

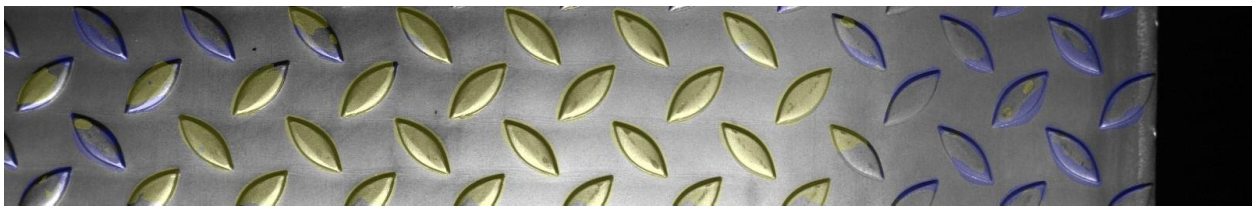
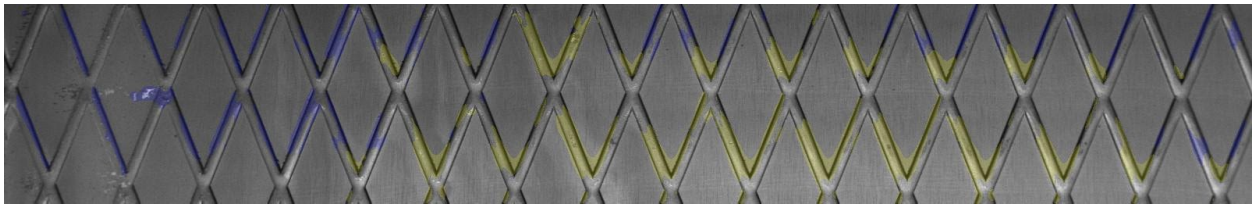


Figure 22 - Prediction with classes 3, 4



Although the model seems to be performing reasonably on this new dataset, one caveat is that these images are still similar to the ones from the original train, validation, and test sets from the same manufacturer. Therefore, results on data from other manufacturers/situations would likely be worse. Nevertheless, these results do show the model performs reasonably on unseen data.

Discussion

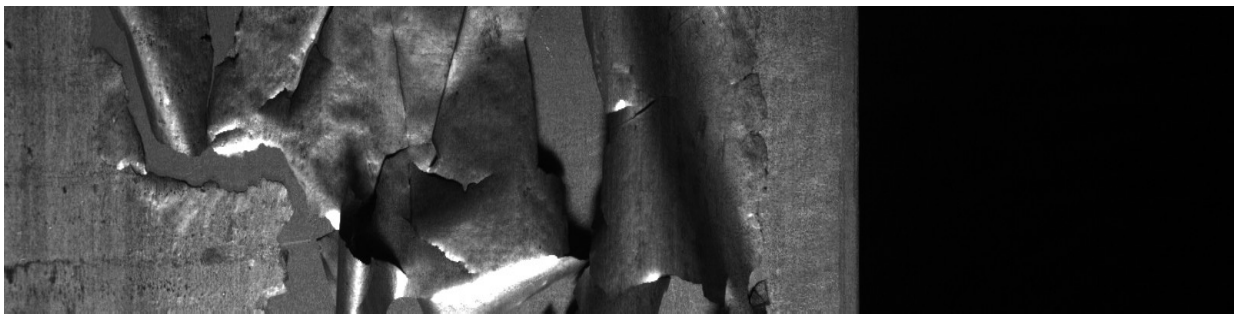
To put the results into context, it is important to understand the data's distribution. From Figure 3's diagonal, the majority of images only have one class of defects. Thus, a naive prediction of always predicting no defects for all classes would yield an accuracy of 75%; the naive prediction would be correct for the three classes that had no defects, and wrong on the single class with defects, so the average is 75%. To show learning has occured, a model should score at least 75%. Since our model had dice accuracies of 84%, 82%, and 83% on the training, validation, and testing sets, the model has achieved some learning.

Although an 83% test accuracy seems good, looking at the distribution that led to it provides interesting insights. From Figure 8 where dice was calculated for each image individually rather than all at once, we see that the majority of dice coefficients for a single image were in the 50% or 25% bins. However, the majority of bins were for dices greater than 50%, so the average being around 83% seems reasonable. Nevertheless, this reveals that the model may be implementing the naive prediction on multi-class images since many dices were exactly 50% or 25%. Thus, it can be said that the model is working well only on images with one defect class, which makes sense because the majority of the training data was like this.

From Figure 8's cumulative probability plot, a 50% probability corresponds to a dice of about 70% (i.e. a 70% dice is the dividing line; if you were to compute the dice on a brand new image, there is a 50% chance that dice will be above 70%, and a 50% chance of being below). Reading the plot, the probability that the dice computed on a single image will be "good" (between 80% and 100%) is about 1-0.7 = ~30%. This suggests that the overall 83% accuracy on the test set is misleading. There are more outliers of dice near 100% than there are outliers near 0%, and this skews the overall average to 83%, but the chance of consistently getting good predictions is poor, suggesting the model is not robust enough for industry.

A qualitative reason supporting the foregoing conclusion is misleading class labels. The sample shown in Figure 23 has a ground truth of no defects because the defects there are neither class 1, 2, 3, or 4. Clearly, 4 classes are not enough to fully classify all possible defects. Although predicting Figure 23 as having no defects is technically correct because that is the labelled ground-truth, such a prediction is fundamentally dangerous.

Figure 23 - Sample with ground-truth of no defects



Some of the ground-truth labels are also inaccurate and missing defects that we believe should be highlighted (left side of Figure 24). There are also inconsistencies in marking defects; Figure 24 shows a wide area covering seemingly non-defective steel, and Figure 25 shows narrow areas only on the defects. This causes issues with model training and makes the data unreliable.

Figure 24 - Missing label, wide ground-truth

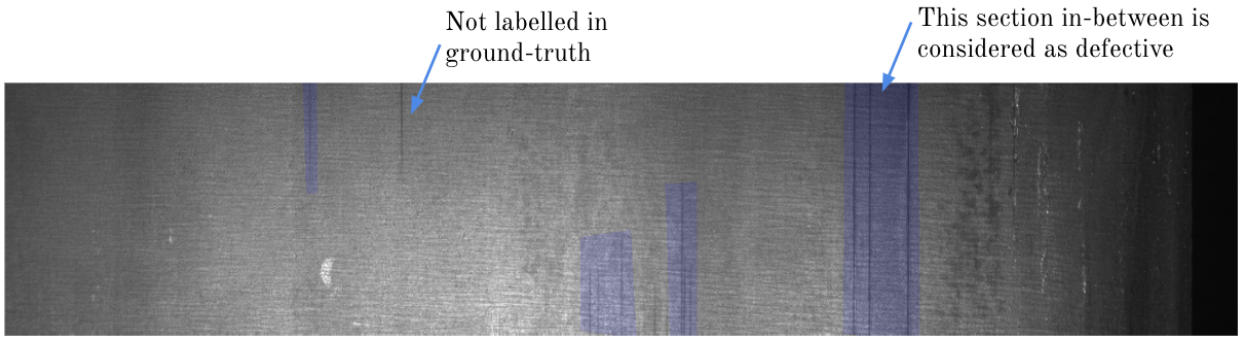
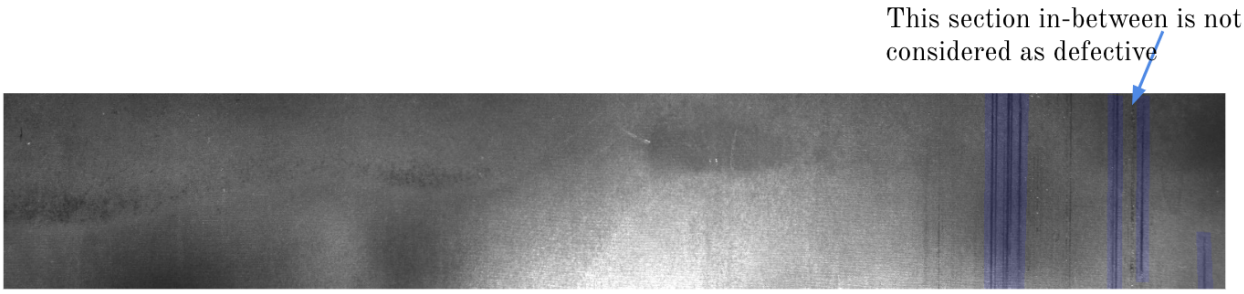


Figure 25 - Narrow ground-truth



Although we have confidence that SDD can potentially be solved with DL, we saw as [2] did that inconsistent/incorrect labelling significantly affects accuracy and prevents practical usage.

Ethical Considerations

SDD models are used in safety-critical industries, and our model was trained on a limited set of data that didn't include all possible steel defect classes. Consequently, using this model in industry may be dangerous.

Another reason for caution is that all the data were manually labeled and provided by one manufacturer. In reality, images come from many manufacturers and may be of different ages. The model will be less accurate on age-related defects (e.g. corrosion, which itself wasn't a labelled training class), or defects from other manufacturers.

To address these ethical considerations associated with implementing our model, we would need to widen the dataset to include images and labels representing these pitfalls, and ideally have a human verify results.

Project Difficulty/Quality

SDD was more complex than expected. The biggest contributor to unexpected complexity was our lack of familiarity with segmentation problems. While image classification was covered early in APS360, segmentation wasn't. Consequently, this project required us to learn and familiarize ourselves with segmentation, which we have successfully achieved.

To split the images and masks correctly and apply augmentations correctly, we could not use the library functions that we used in the labs as those were only for classification. We managed to overcome this by writing custom implementations. Moreover, our model's performance is decent considering the problem, achieving accuracies in the low 80s.

Notebook

https://colab.research.google.com/drive/1iZ9Qx-RnWIGvC2xHbzpKzoHFVdUl1BxH

References

[1] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", *Lecture Notes in Computer Science*, pp. 234-241, 2015. Available: 10.1007/978-3-319-24574-4_28 [Accessed 5 November 2019].

[2] H. Di, X. Ke, Z. Peng and Z. Dongdong, "Surface defect classification of steels with a new semi-supervised learning method", *Optics and Lasers in Engineering*, vol. 117, pp. 40-48, 2019. Available: 10.1016/j.optlaseng.2019.01.011.

[3] "Severstal: Steel Defect Detection | Kaggle", *Kaggle.com*, 2019. [Online]. Available: https://www.kaggle.com/c/severstal-steel-defect-detection/data. [Accessed: 06- Oct- 2019].