Capstone 2: Milestone Report 1

The data for the steel sheets defect image segmentation project was obtained from Severstal and Kaggle repositories: https://www.kaggle.com/c/severstal-steel-defect-detection/overview/description. Identifying different types of steel sheet defects is critical to improving Severstal's automation, increasing efficiency, and maintaining high quality in their production. The company is taking major steps towards combining newer technologies with steel production. As such, deep Learning techniques are currently being employed to characterize and correctly identify defects in images taken by high frequency cameras.

Description of Dataset

The provided dataset contains train and test images with an area of 256 x 1600 pixels each, and the labels have the pixel values where the defects are segmented. They have been encoded in a run-length encoding (RLE) style, and need to be first transformed into an area of 256 x 1600 pixels, before being fitted on the images. For example, '1 3 10 5' implies pixels 1,2,3,10,11,12,13,14 are to be included in the mask. The metric checks that the pairs are sorted, positive, and the decoded pixel values are not duplicated. The pixels are numbered from top to bottom, then left to right: 1 is pixel (1,1), 2 is pixel (2,1), etc.

The Sorensen-Dice coefficient is used as the metric of evaluation: it is used to gauge the similarity of two samples. It is commonly used in image segmentation, and can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. The formula is given by the following, where X is the predicted set of pixels and Y is the ground truth:

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

The analysis of this project was possible with the help of xhlulu's boilerplate code (https://www.kaggle.com/xhlulu/severstal-simple-keras-u-net-boilerplate). The author graciously agreed to share their work on public Kaggle kernels, and I was able to fork off of their work and introduce many of my own concepts.

The images are represented as follows (Figure 1). The left image shows the original image as converted using the CV2 library, and the right image shows the defects masked on to the original image. The right images have been transformed using the encoded pixels as explained below. Viridis color scale has been employed to showcase the four defects by different colors: 1 = green, 2 = light blue, 3 = dark blue, 4 = yellow.

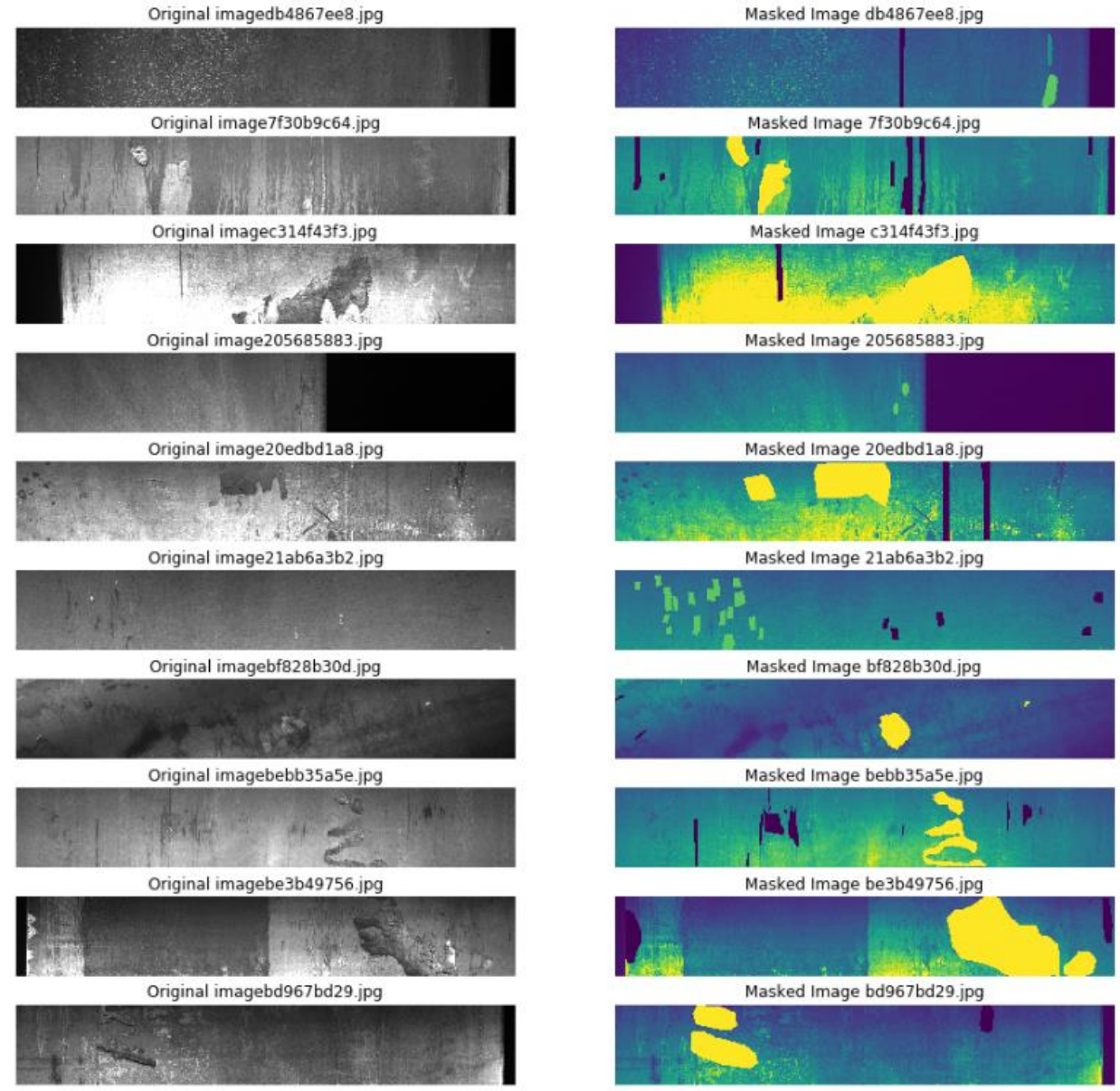

Figure 1: Original and Masked Image Subset of the 12568 Steel Images

DATA PREPROCESSING AND WRANGLING

The preprocessed data had columns of ImageId_ClassId and EncodedPixels. The first step involved splitting the ImageId and ClassId from each other, and in classifying whether the image has a mask of defects. After wrangling, the data looked like the following table (Table 1).

Table 1: Train data after separating ImageID and ClassID

	ImageId_ClassId	EncodedPixels	ImageId	ClassId	hasMask
0	0002cc93b.jpg_1	29102 12 29346 24 29602 24 29858 24 30114 24 3...	0002cc93b.jpg	1	True
1	0002cc93b.jpg_2	NaN	0002cc93b.jpg	2	False
2	0002cc93b.jpg_3	NaN	0002cc93b.jpg	3	False
3	0002cc93b.jpg_4	NaN	0002cc93b.jpg	4	False
4	00031f466.jpg_1	NaN	00031f466.jpg	1	False

As portrayed in the table, each image has a separate mask for the four types of defects, and these masks will be used as labels for the image set.

To convert the values and images from pixels to images and vice-versa, the OpenCV Python library was used (https://opencv.org/). Functions that built masks from RLEs and those that built RLEs from masks were defined. For example, *build_masks* function was used to first create a layer of zeros, and then call the *rle2mask* function. The *rle2mask* function was used to convert "10 4 30 2" to 10, 11, 12, 13, 30, 31, and assign them a value of 1. This function is a mask encoding function. Similarly, mask decoding function was also defined for converting mask pixel data to RLE data.

Exploratory Data Analysis

The four types of defects were worth looking into, and their summary statistics are shown below in Figure 2. Defect 3 is quite common, showing up in about 40% of all images.

ClassId	num_defects
1	897.0
2	247.0
3	5150.0
4	801.0

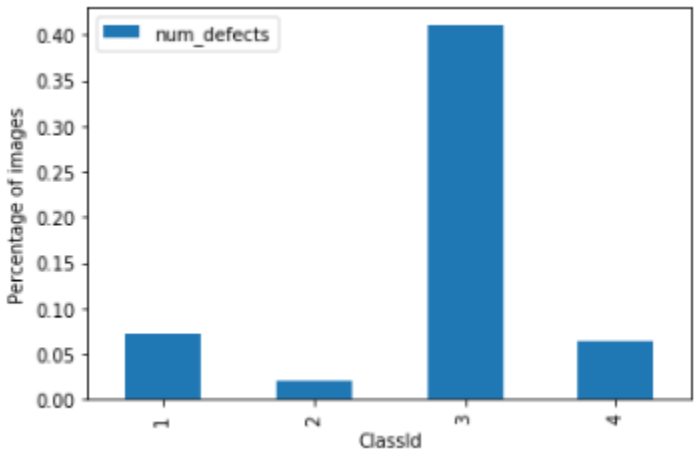

Figure 2: Absolute and Relative Number of Defects in the Dataset

Studying the defects more deeply, the area they covered per image has been visualized in their log histograms. The following figure (Figure 3) shows the number of images that have zero defects in each of the four defect categories, and the distribution of the area covered by the four defects (calculated as total number of pixels of defect / total number of pixels).

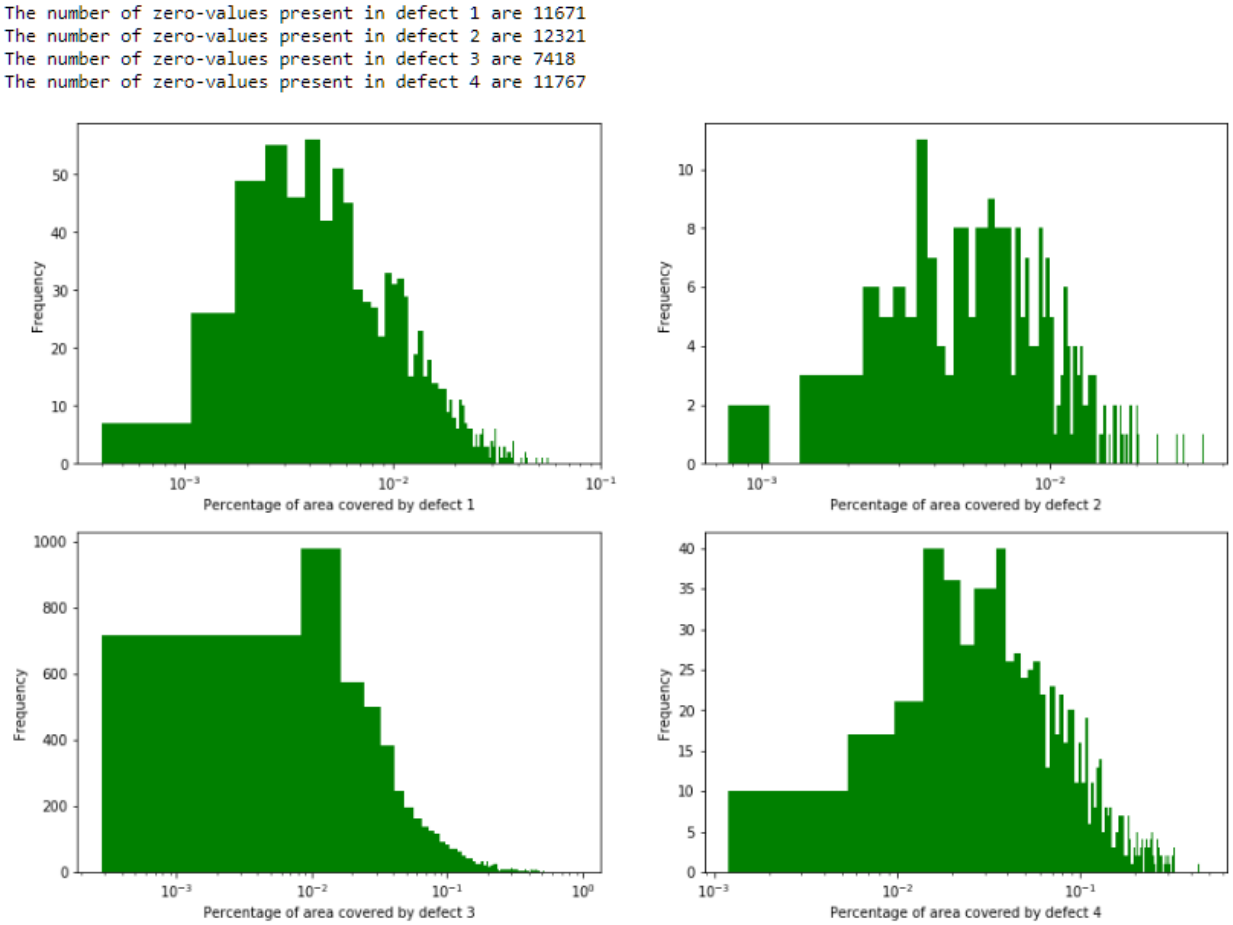

Figure 3: Total Areas covered by each of the 4 Defects on Log Histogram Distributions

The areas covered by both defects 3 and 4 occupy major portions of the image (between 0.01 and 0.1% of the total area), whereas defects 1 and 2 occupy tinier portions. The following table shows the descriptive statistics (Table 2) for the four defects. Again, defect 3 shows the highest values for means.

Table 2: Descriptive Statistics for the 4 Defects

	num_total_masks	defect_0_percentage	defect_1_percentage	defect_2_percentage	defect_3_percentage
count	12568.000000	12568.000000	12568.000000	12568.000000	12568.000000
mean	0.564529	0.000760	0.000162	0.025507	0.005349
std	0.560757	0.003605	0.001342	0.066735	0.027432
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000	0.020638	0.000000
max	3.000000	0.076423	0.034236	0.899023	0.470654

Next Steps

The next steps deal with fitting various convolutional neural network (CNNs) models to the dataset. These should accurately predict each of the 4 defect masks given any test image. Of course, with GPUs and higher processing times on a distributed computing system, one can obtain results closer to 100% accuracy. But the challenge is to run models offline within an hour of runtime.