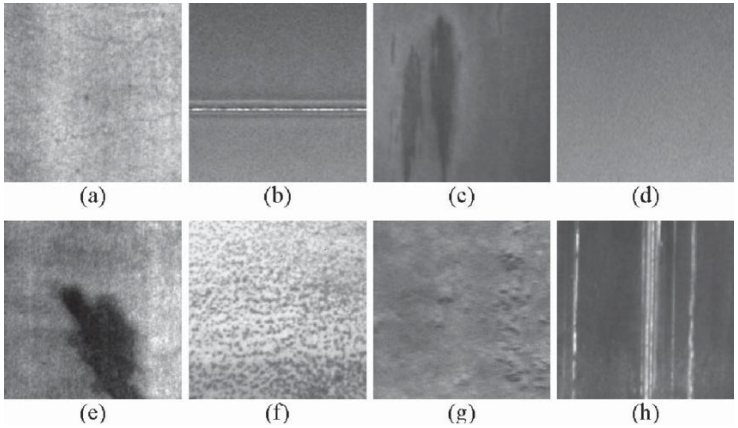
Detection, Classification, and Segmentation of Steel Defects

—

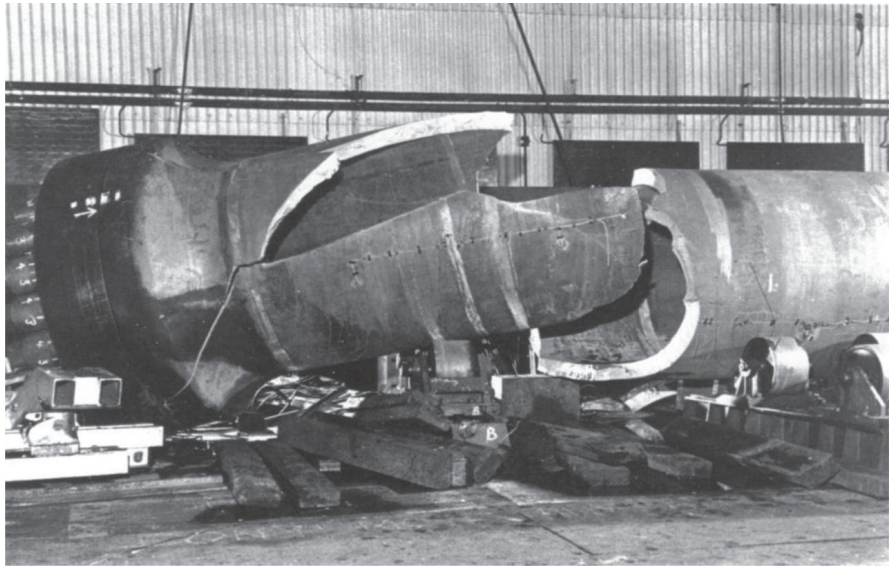
Farhan Wadia, Hiranya Maharaja,
Wenju Zhang, Zhipu Zhang

Introduction & Problem

1. **Detect Steel Defects** — Binary. Defects suspected or not.
2. **Classify Steel Defects** — What is the type of defect? (Class 1, 2, 3, 4)
3. **Segment Steel Defects** — Where are the defects for each class?.



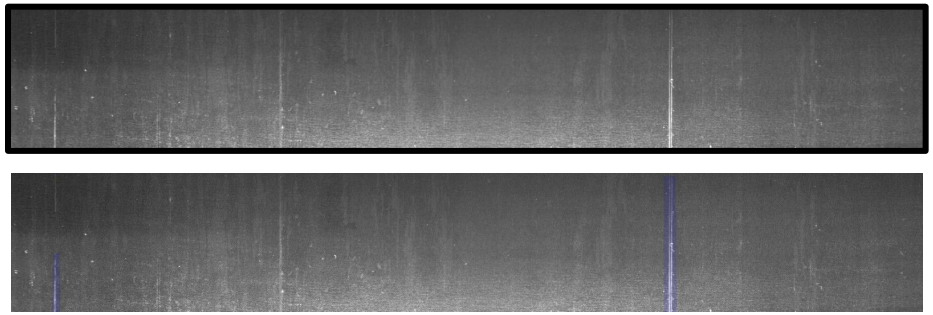
Significance



Data

ImageId_ClassId	EncodedPixels
0007a71bf.jpg_1	
0007a71bf.jpg_2	
0007a71bf.jpg_3	18661 28 18863 82 19091 110 19347 110 19
0007a71bf.jpg_4	
000a4bcdd.jpg_1	37607 3 37858 8 38108 14 38359 20 38610
000a4bcdd.jpg_2	
000a4bcdd.jpg_3	
000a4bcdd.jpg_4	
000f6bf48.jpg_1	
000f6bf48.jpg_2	
000f6bf48.jpg_3	
000f6bf48.jpg_4	131973 1 132228 4 132483 6 132738 8 132
0014fce06.jpg_1	
0014fce06.jpg_2	
0014fce06.jpg_3	229501 11 229741 33 229981 55 230221 77
0014fce06.jpg_4	
001982b08.jpg_1	
001982b08.jpg_2	
001982b08.jpg_3	
001982b08.jpg_4	

12568 images (256 x 1600) and 12568×4 run-length encodings.

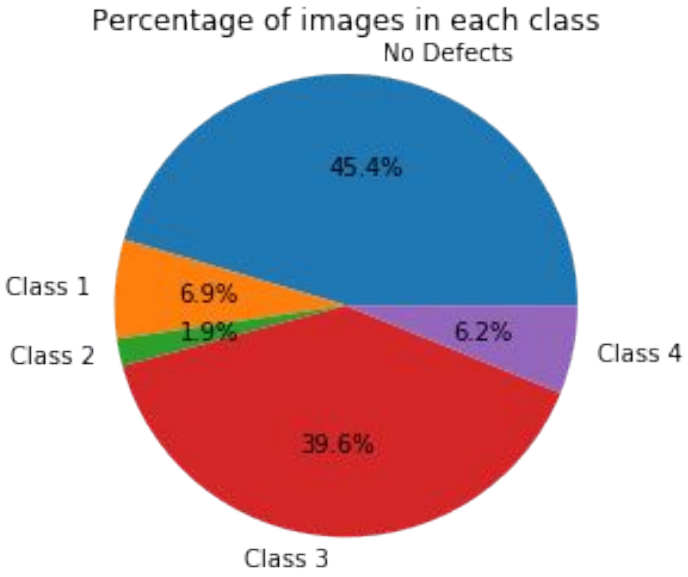


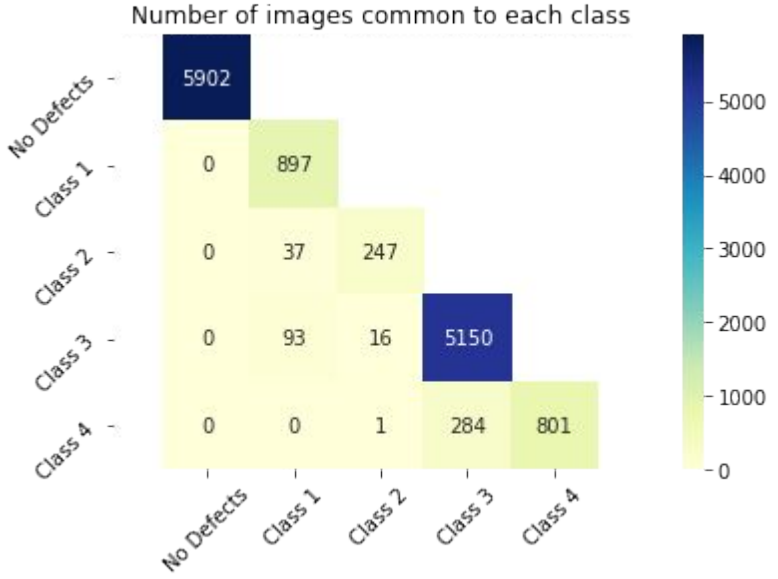
1	7	13	19	25	31
2	8	14	20	26	32
3	9	15	21	27	33
4	10	16	22	28	34
5	11	17	23	29	35
6	12	18	24	30	36

EncodedPixels: 8 3 18 4 33 2

Data

- Classes 1-4 split 70-15-15 for stage 2 training, validation, and testing
  - 4760 training images, 1055 validation images, and 1055 test images



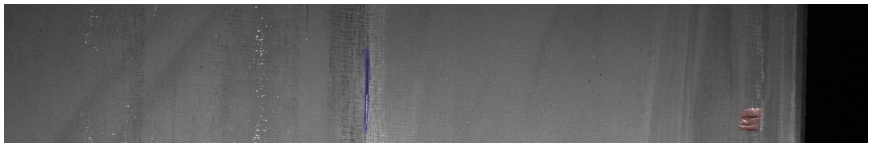


Data Processing

- Convert run-length encodings to one-hot masks
- Downsize images (trained with 96 x 576)
- Randomly apply (25% of the time):
  - Horizontal flips
  - Vertical flips
  - Rotations
- Convert to tensor

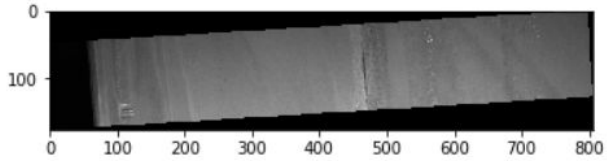
Issue: How to apply a random transformation to an image, but also apply that same transformation to its segmentation mask?

Solution: Create a custom dataset class to load and process images and masks before passing to the data loader



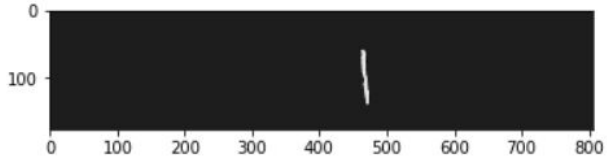
```
[23] # a known image with defects in class 1 and 3
     image, masks = train_dataset[4137]
     plt.imshow(image.numpy(), cmap = 'gray')
```

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



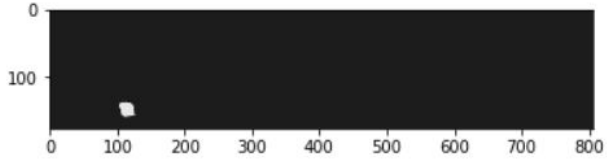

```
[24] #verify that class 3 map is transformed correctly
     plt.imshow(masks.numpy()[2], cmap = 'gray')
```

```
<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>
```



Models Overview

Stage 1: Binary Classification

AlexNet

CNN with ANN classifier

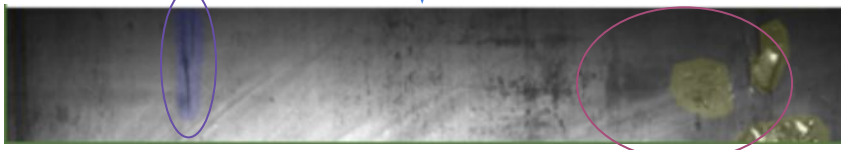
No Defects

Defects

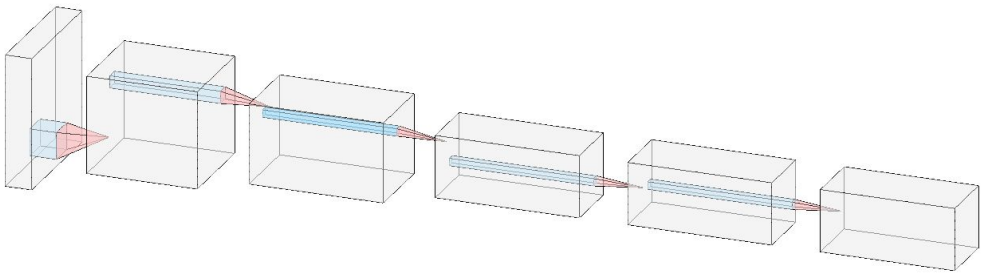
Stage 2: Semantic Segmentation

ResNet

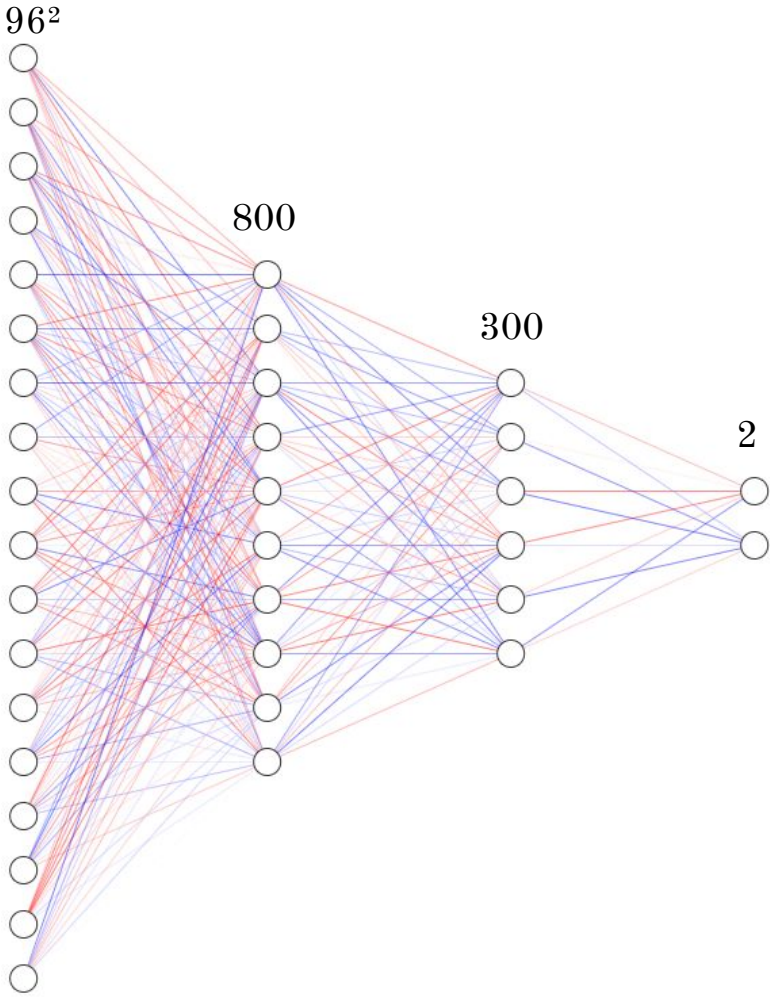
U-Net



Stage 1: Binary Classification

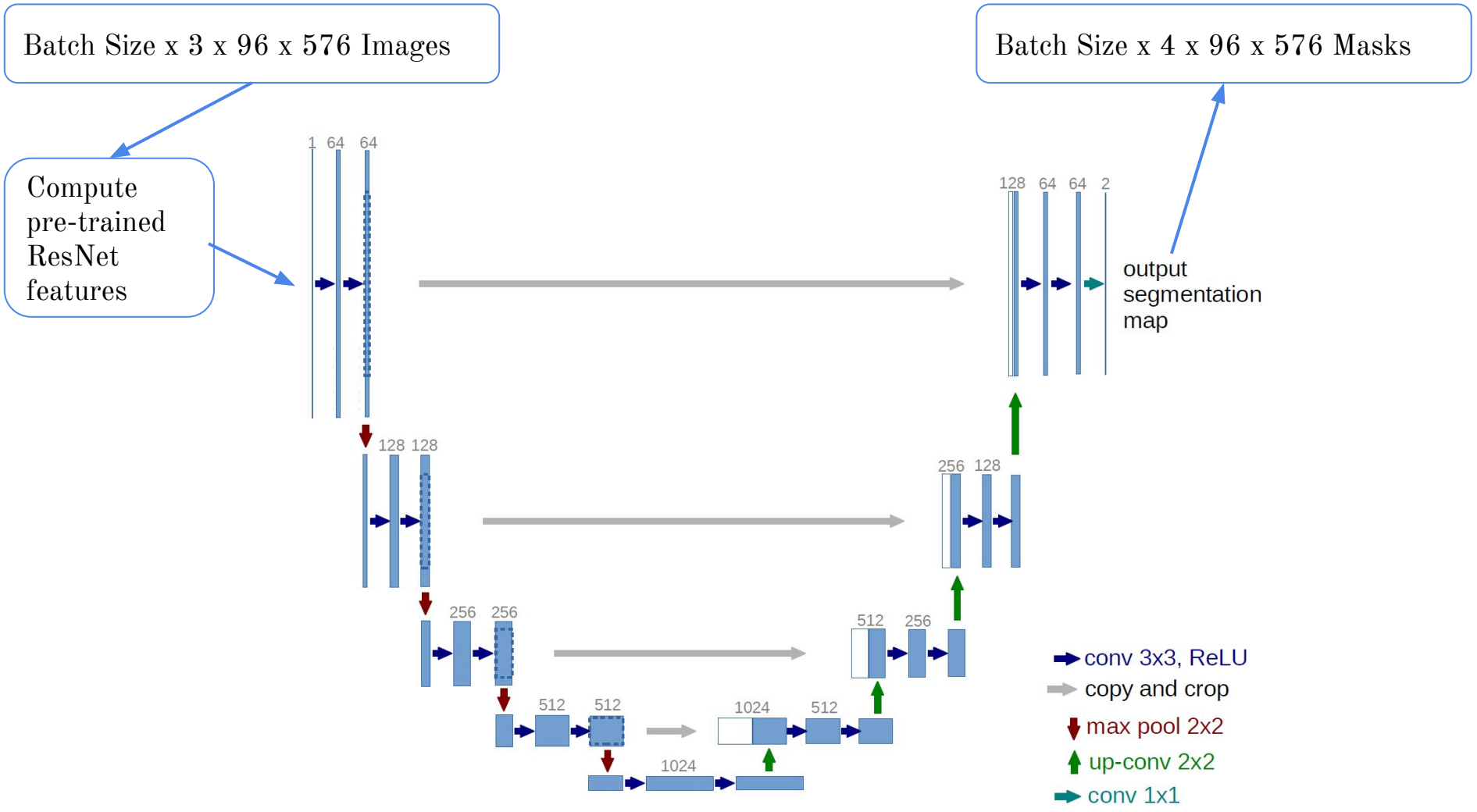


Pre-trained AlexNet feature extractor



ANN Classifier

Stage 2: ResNet U-Net Semantic Segmentation



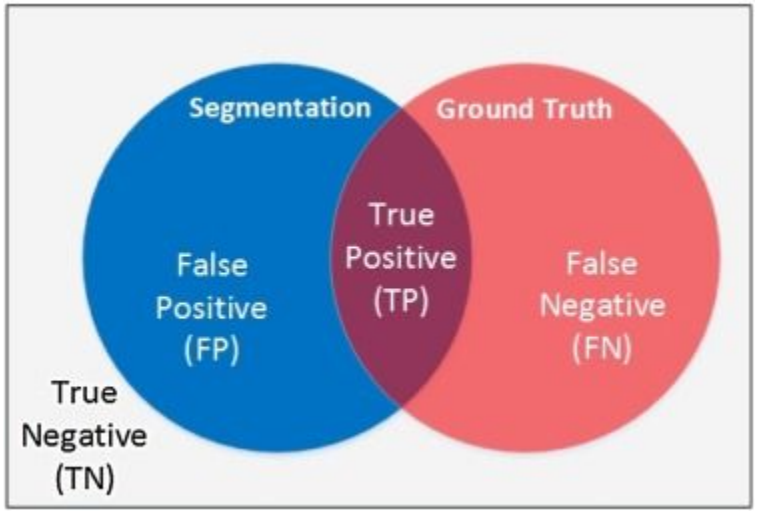
Stage 2: Hyperparameter Tuning

- Tried U-Net model without pre-trained ResNet features
- Changing number of epochs
- Changing learning rate
- Changing loss function
  - Weighted combination of BCE and Dice
  - Best model used only BCE

Quantifying Accuracy: Dice Coefficient

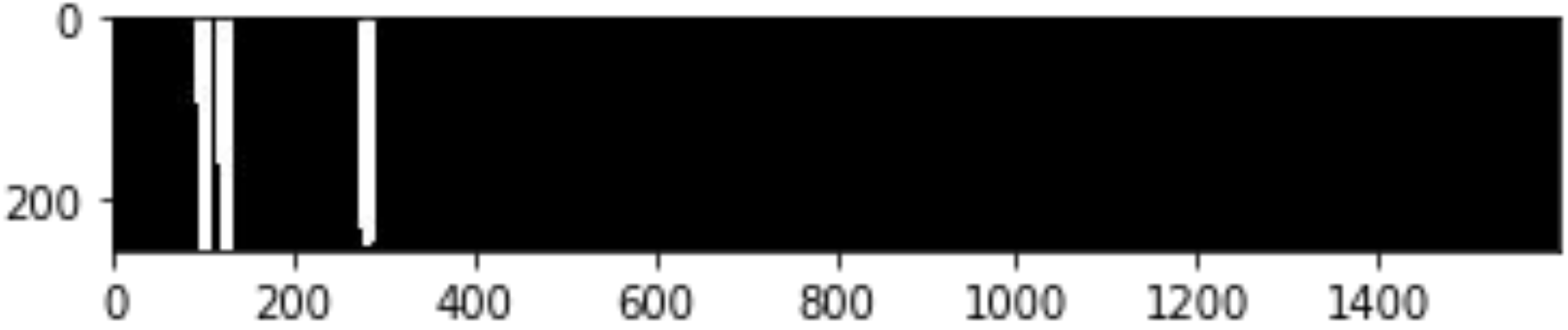
- Measures the accuracy of the semantic segmentation model
- Indicates how similar the predictions are to the targets pixel-by-pixel within each class
- Average the dice coefficients over all the classes and the number of images

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

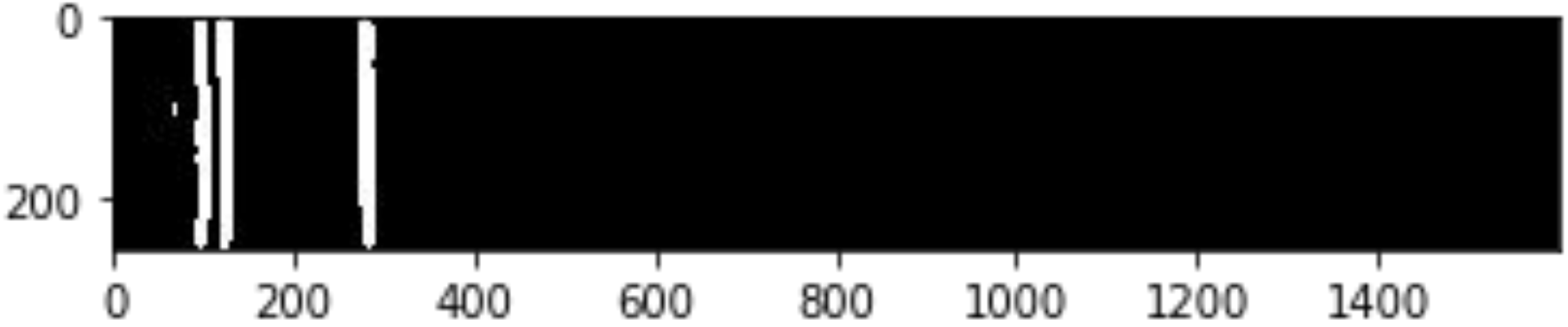


So what does a result look like?

Ground Truth (for one class)



Model Prediction (for one class)



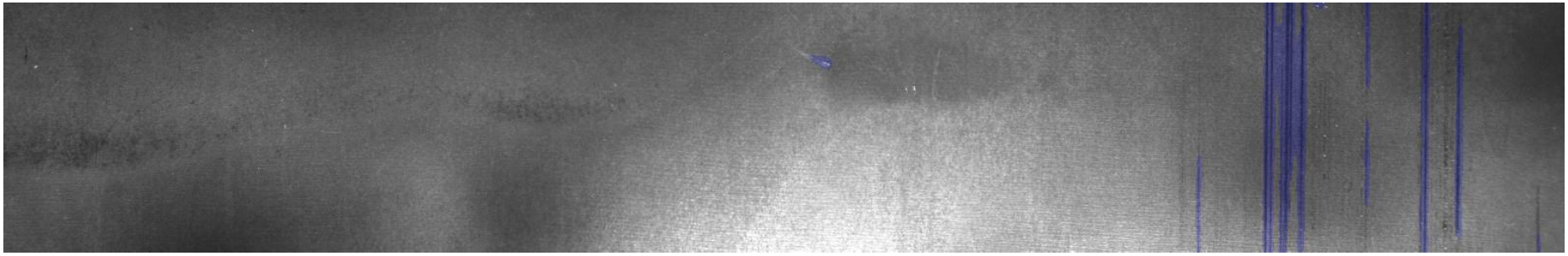
Test Results

	SVM Baseline	Model
Stage 1: Binary Detection Accuracy	91%	100%
Stage 2: Semantic Segmentation Accuracy	72%	83% (Mean Dice)

Examples from Test Results

Ground Truth

Predicted



Mean Dice Coefficient: 92.4%

Examples from Test Results

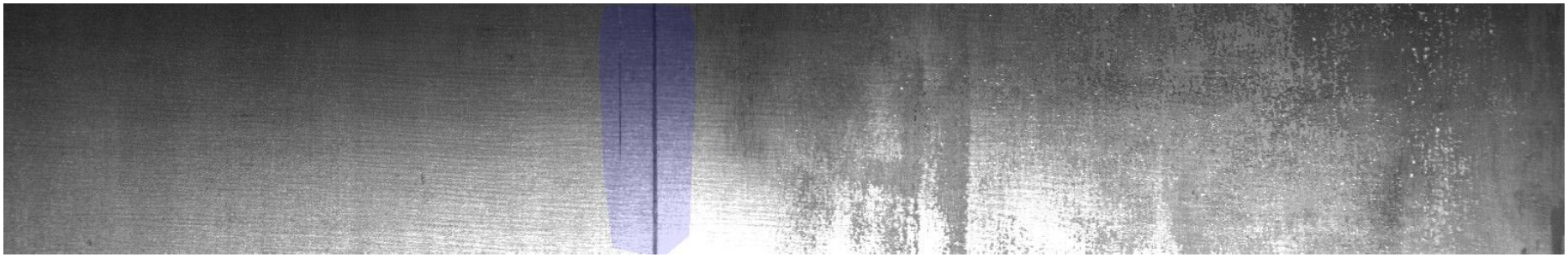
Ground Truth

Predicted

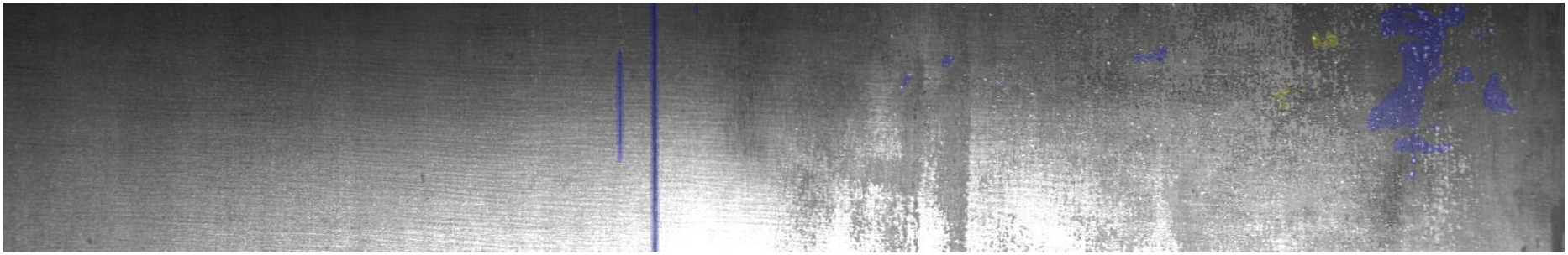
Mean Dice Coefficient: 87.3%

Examples from Test Results

Ground Truth



Predicted



Mean Dice Coefficient: 56.3%

Qualitative Results - Reasons for Success

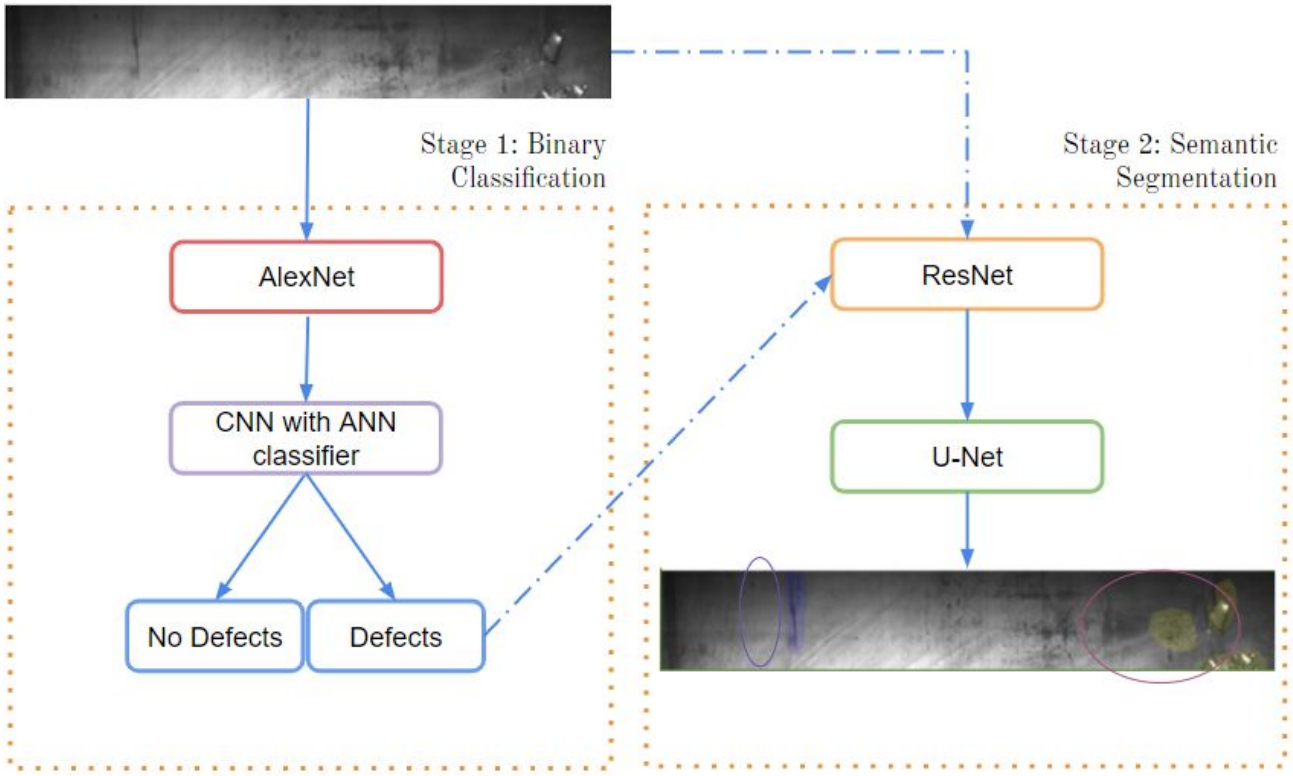
- Usage of transfer learning:
  - AlexNet
  - ResNet 18

- ResNet skip connections allow for deeper learning without vanishing gradients

- U-Net is well-known for semantic segmentation in the biomedical industry

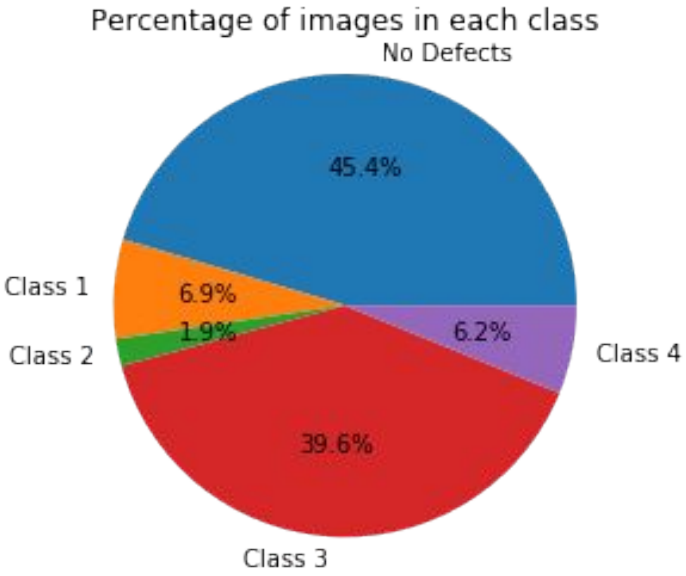
Qualitative Results - Comparisons

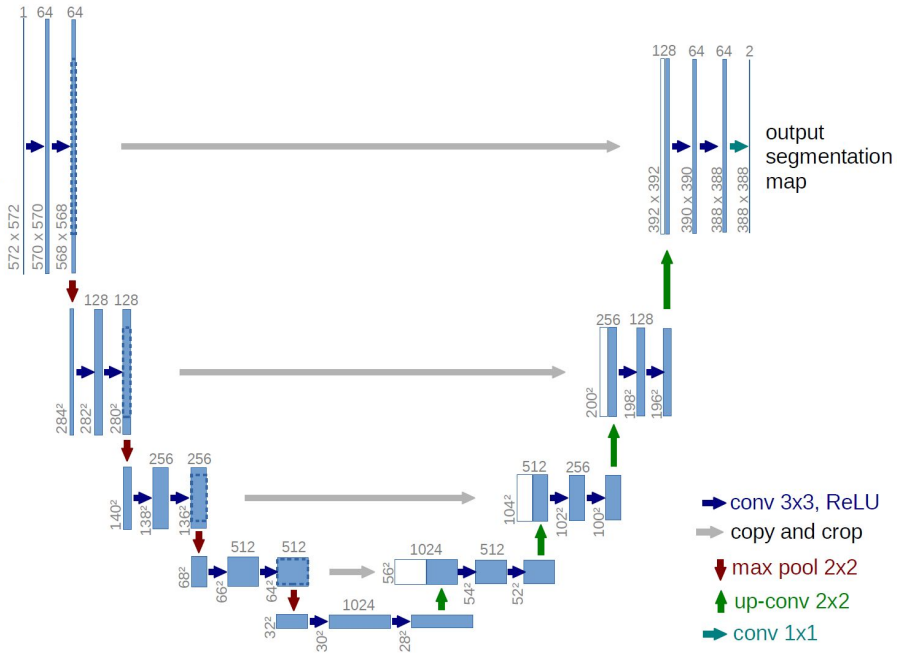
- No significant difference in accuracy for model with Stage 1 vs. without
  - ResNet - Enough layers to go from 4 to 5 classes
  - Stage 2 convolutions also extract features - similar to AlexNet
- Sanity check - both models should result in the same answer



Qualitative Results - Reasons for Error

- Dataset is imbalanced
- Information lost when encoding/decoding features through U-Net

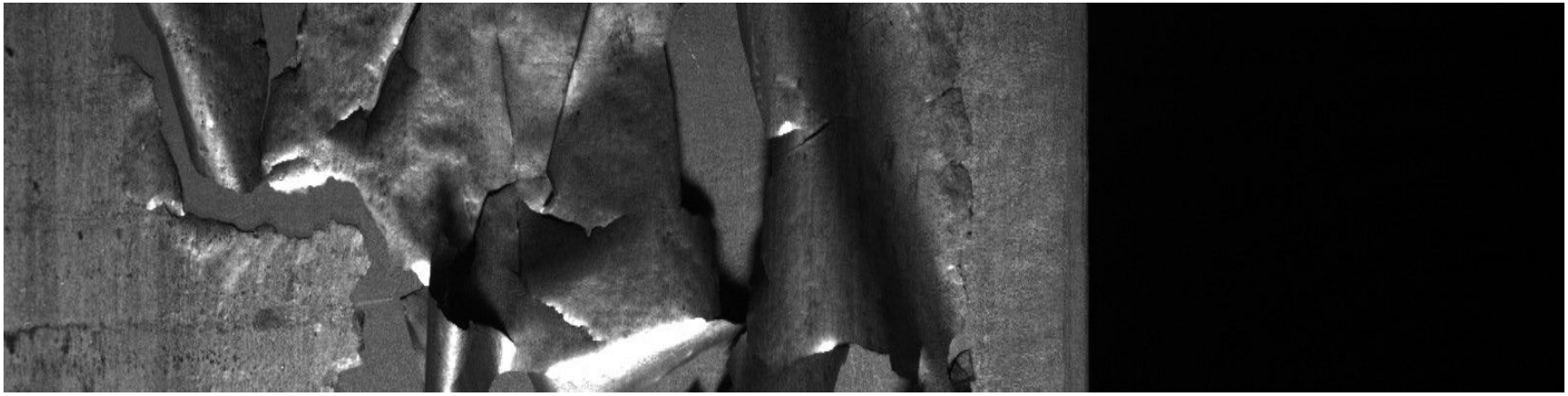




Discussion / Insights

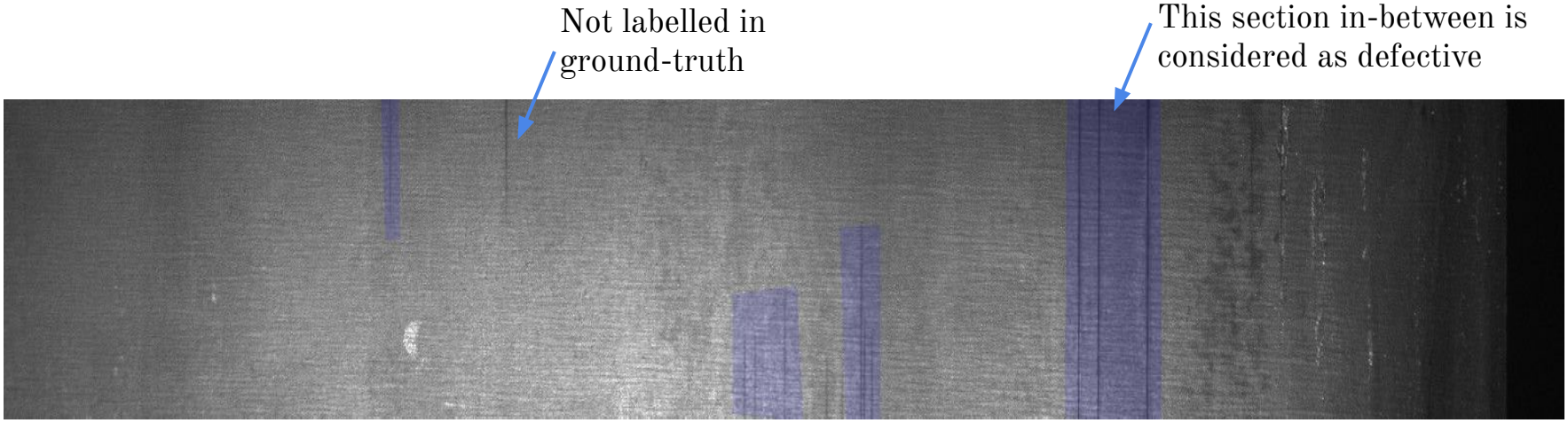
- Misleading Labels
  - The image below is considered to have no defects because this type of defect is not part of class 1, 2, 3, or 4

- Incomplete Number of Classes
  - Steel defects cannot be segmented into only 4 classes; there are many more types of defects than the ones our model was trained to identify



Discussion / Insights

- Mislabeling
  - Datasets were manually labeled - ground truth labels might be inaccurate
- Manufacturer Bias
  - All data is of new steel sheets from one manufacturer
  - Potentially cannot be extrapolated to samples from other manufacturers
  - Will not extrapolate to more complex in-field situations (e.g. rail crack detection, rebar crack detection etc.)
- Overfitting Potential
  - Images that appear grayscale are being passed in as RGB



Demonstration

(Kaggle-provided test set of 1800 images, ground truth unknown)