Severstal Steel Defect Detection (Image Segmentation Architecture)

Springboard Capstone Project II

Shubham Tiwari

Table of Contents

The Severstal steel defects detection task is to localize and classify surface defects on a steel sheet. Flat sheet steel production is quite delicate.

Throughout the processes of heating and rolling, drying and cutting, several machines are in contact with the steel, and hence introduce a variety of defects on the surface. Severstal uses images from high frequency cameras to power a defect detection algorithm, and are continuously improving industrial algorithms using deep learning techniques.

Problem Statement

To correctly segment by pixels and classify the 4 types of defects found in the steel sheets images.

Methodology

Create convolutional neural network (CNN) architecture used for image segmentation tasks, fit training images and check against the target metrics.

Acknowledgement

The following code has been completed with the help of xhlulu's boilerplate code (xhlulu, 2019): The author graciously shared their work on public Kaggle kernels, and I was able to fork off of their work and introduce many of my own concepts.

xhlulu. (2019). *Severstal: Simple Keras U-Net Boilerplate.* (Kaggle) Retrieved 08 05, 2019, from https://www.kaggle.com/xhlulu/severstal-simple-keras-u-net-boilerplate

Training data file format

	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

Training Data

- 12568 images with 4 defects each
- 256 x 1600 pixels each
- Labels include run-length encoded (RLE) image masks

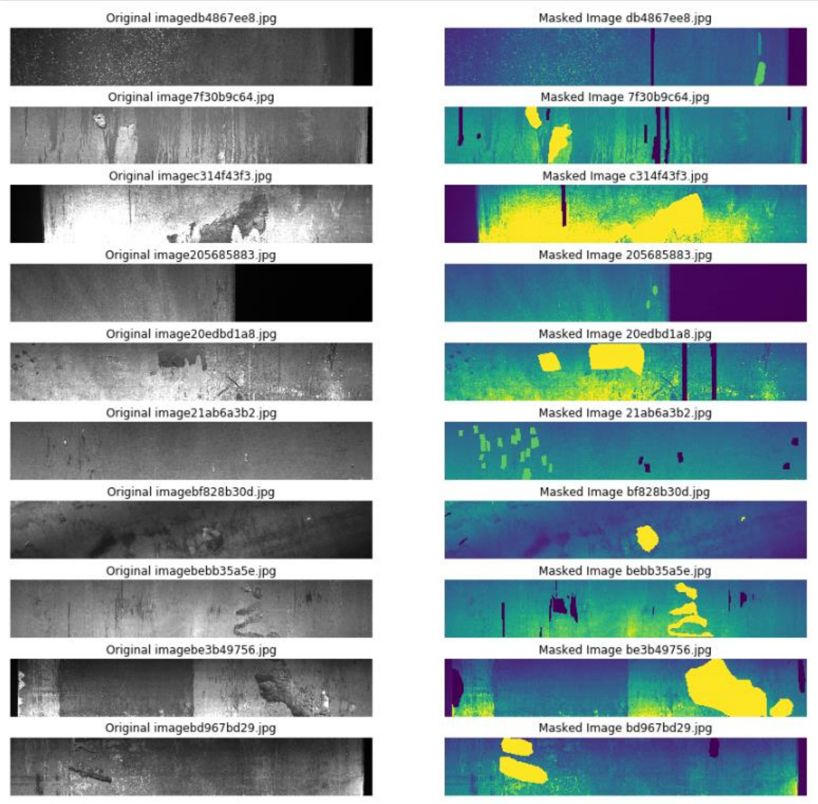
RLE Masks

'1 3 10 5' implies pixels 1,2,3,10,11,12,13,14 are to be included in the mask.

Evaluation Metric

Sorensen-Dice coefficient: used to gauge similarity of two samples – X = predicted pixel set; Y = the ground truth

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



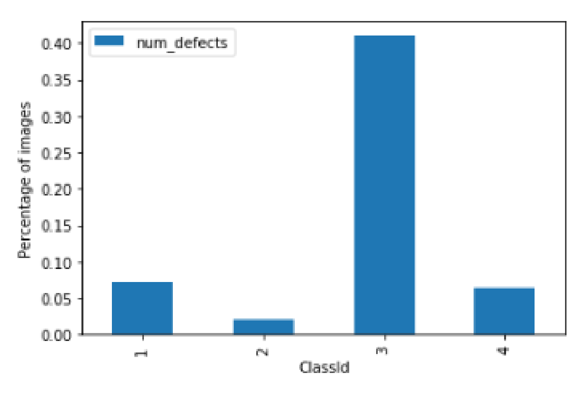
Original (Left) and Masked Image Subset of the 12568 Steel Images

Image and Mask Visualization

- Built using OpenCV library and viridis color scheme
- Images on the left are original
- Images on the right are images with mask on
- Color codes for the defects:
  - 1 = green,
  - 2 = light blue,
  - 3 = dark blue,
  - 4 = yellow.

Distribution of Defects and their Areas

ClassId	num_defects
1	897.0
2	247.0
3	5150.0
4	801.0



Absolute and Relative Number of Defects in the Dataset

Defect 3

Percentage of images that have defect 3

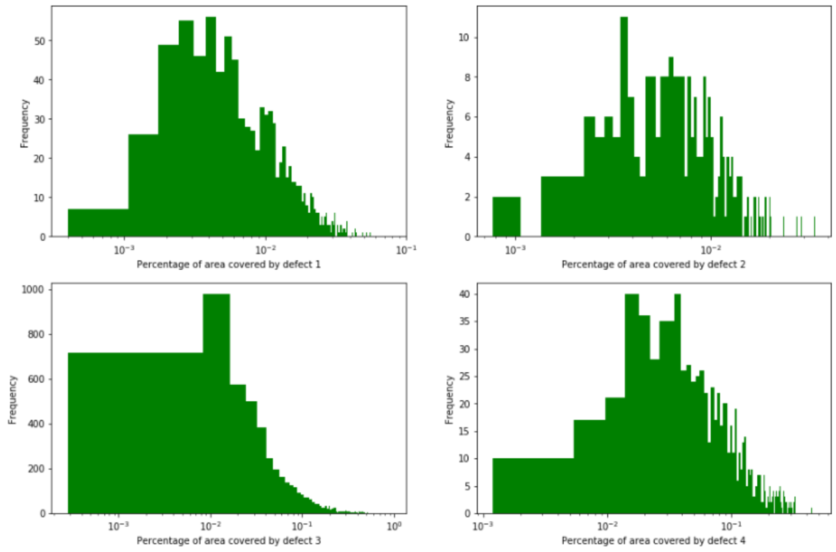
40%

0.01% - 0.1%

Percentage of area of an image covered by defects 3 and 4

Log-scale used for distribution of areas (pixels occupied by mask / total pixels) to better visualize the spread

```
The number of zero-values present in defect 1 are 11671
The number of zero-values present in defect 2 are 12321
The number of zero-values present in defect 3 are 7418
The number of zero-values present in defect 4 are 11767
```



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

Image Segmentation Problem

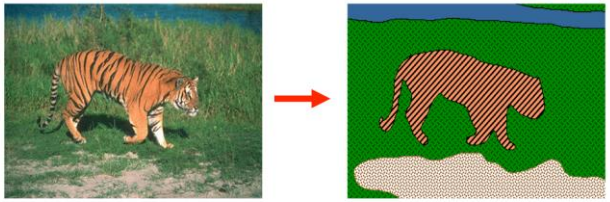


Image segmentation task, which requires encoding-decoding neural networks (Image source: Stanford AI Introduction to Computer Vision (Amy, 2019))

1. Involves convolution and max pooling to lower the input image dimensions
2. Then employs deconvolution and concatenation to reconstruct input image
3. Mask labels then are used on this reconstructed image to train model

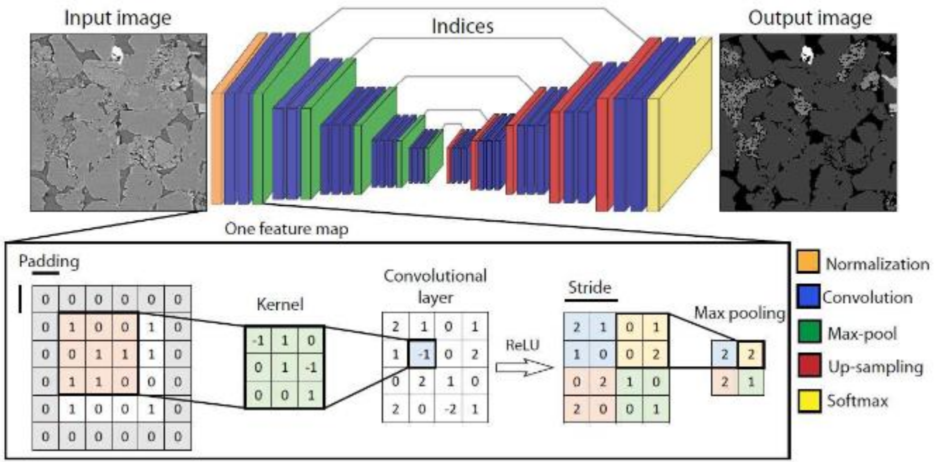


Image segmentation describing the convolution, max-pool and deconvolution operations. (image source: (Sadegh Karimpouli, 2019))

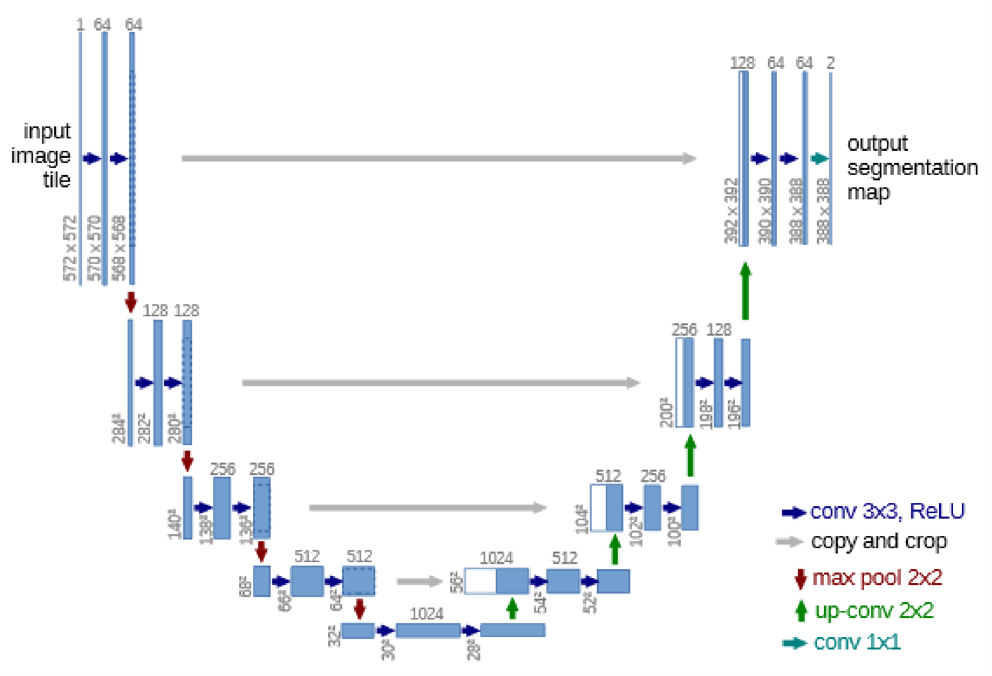
Amy, V. a. (2019). Introduction to Computer Vision: Tutorial 3: Image Segmentation. Retrieved 08 02, 2019, from https://ai.stanford.edu/~syyeung/cvweb/tutorial3.html

Sadegh Karimpouli, P. T. (2019). Segmentation of digital rock images using deep convolutional autoencoder networks. Computers and Geosciences, 126(May), 142-150.

U-Net Architecture

- Model used primarily in medical image segmentation techniques

- Custom data generator class with the *keras.utils.Sequence* parameter generally instantiated to feed images in batches
- *__len__* method returns the number of batches per epoch
- *__get_item__(index)* returns the images (X) and the masks (y) associated with the indices of the training and the validation data
- The *on_epoch_end()* function used to trigger a shuffle of the indices at the end of an epoch, to create a more robust model



U-Net architecture, for a 572 x 572 pixel image. The arrows represent convolution or max-pooling operations. The number of channels or nodes, are denoted at the top of the boxes. The blue boxes represent multi-channel feature maps, and each white box represents copied feature maps. Source: Ronneberger et. al. (Olaf Ronneberger, 2015)

Olaf Ronneberger, P. F. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In H. J. Navab N. (Ed.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 (Vol. 9351). Springer, Cham. doi:https://doi.org/10.1007/978-3-319-24574-4_28

```python
def build_model(input_shape):
    inputs = Input(input_shape)

    c1 = Conv2D(8, (3, 3), activation='elu', padding='same') (inputs)
    c1 = Conv2D(8, (3, 3), activation='elu', padding='same') (c1)
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = Conv2D(16, (3, 3), activation='elu', padding='same') (p1)
    c2 = Conv2D(16, (3, 3), activation='elu', padding='same') (c2)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = Conv2D(32, (3, 3), activation='elu', padding='same') (p2)
    c3 = Conv2D(32, (3, 3), activation='elu', padding='same') (c3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = Conv2D(64, (3, 3), activation='elu', padding='same') (p3)
    c4 = Conv2D(64, (3, 3), activation='elu', padding='same') (c4)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = Conv2D(64, (3, 3), activation='elu', padding='same') (p4)
    c5 = Conv2D(64, (3, 3), activation='elu', padding='same') (c5)
    p5 = MaxPooling2D(pool_size=(2, 2)) (c5)

    c55 = Conv2D(128, (3, 3), activation='elu', padding='same') (p5)
    c55 = Conv2D(128, (3, 3), activation='elu', padding='same') (c55)

    u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c55)
    u6 = concatenate([u6, c5])
    c6 = Conv2D(64, (3, 3), activation='elu', padding='same') (u6)
    c6 = Conv2D(64, (3, 3), activation='elu', padding='same') (c6)

    u71 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c6)
    u71 = concatenate([u71, c4])
    c71 = Conv2D(32, (3, 3), activation='elu', padding='same') (u71)
    c61 = Conv2D(32, (3, 3), activation='elu', padding='same') (c71)

    u7 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c61)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(32, (3, 3), activation='elu', padding='same') (u7)
    c7 = Conv2D(32, (3, 3), activation='elu', padding='same') (c7)

    u8 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(16, (3, 3), activation='elu', padding='same') (u8)
    c8 = Conv2D(16, (3, 3), activation='elu', padding='same') (c8)

    u9 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(8, (3, 3), activation='elu', padding='same') (u9)
    c9 = Conv2D(8, (3, 3), activation='elu', padding='same') (c9)

    outputs = Conv2D(4, (1, 1), activation='sigmoid') (c9)

    model = Model(inputs=[inputs], outputs=[outputs])
    adam = Adam(learning_rate=0.001)
    model.compile(optimizer=adam, loss=bce_dice_loss, metrics=[dice_coef])

    return model
```

U-Net Architecture

Our particular model uses:

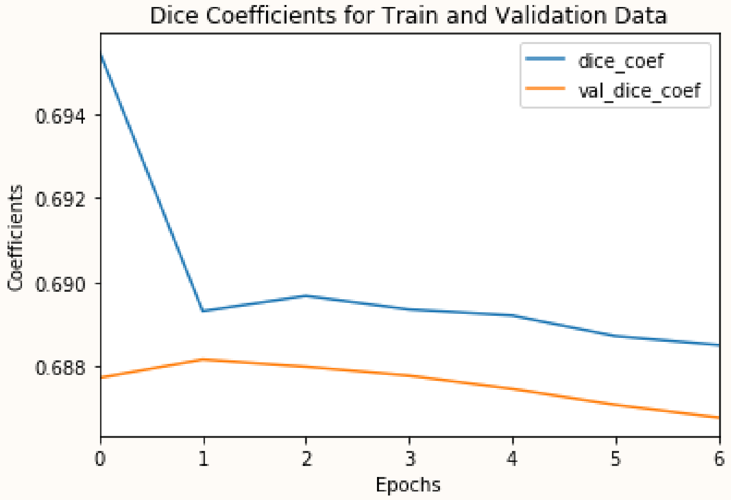
- 3x3 convolution boxes with padding and rectified linear unit (ReLU) averaging
- 2x2 max pooling
- Adam optimizer with a learning rate of 0.001
- Dice Coefficient used as Evaluation Metric
- Binary cross-entropy used for loss function
- Batch size of 32 images randomly fed for training

Training Results

- Trained over 7 epochs, a final train dice coefficient of 0.6885 and validation dice coefficient of 0.6868 was registered
- Masks for test images generated using this model's weights: please find attached *submission.xlsx* file

Future Work

- Number of nodes per layer could be doubled, thus increasing resolution of trained weights
- Another convolution and corresponding deconvolution layer can be added
- A train-test split more representative of the defect distribution could have improved model accuracy
- Train images over other architectures like FCNs, FPN, ParseNet, Mask R-CNN, EncNet etc.



Dice coefficients for the training and the validation datasets, over each epoch

Credits

- Mentor guidance and support by Dipanjan Sarkar
- Dataset by Kaggle, Inc.
- Presentation template by Slidesgo
- Icons by Flaticon
- Infographics by Freepik
- Images created by Freepik