Project Report : CS 7643

Ksenia Katasonova
Georgia Institute of Technology
Institution address
kkatasonova3@gatech.edu

Vera Katasonova
Georgia Institute of Technology
Institution address
vkatasonova3@gatech.edu

Abstract

This scientific paper presents a study on predicting the location and type of defects in steel manufacturing. The study involves segmenting defects of each class, with each image having no defects, a defect of a single class, or defects of multiple classes. To increase the size of training data, the study implements image augmentation with the Albuminations library and compares the results with and without augmentation. The study evaluates the performance of U-Net models using mean Dice coefficient and other possible metrics. The results of the study provide insights into the effectiveness of different models and the impact of image augmentation on model performance.

1. Introduction/Background/Motivation

The goal of the study was to predict the location and type of defects found in steel manufacturing. The study aimed to segment defects of each class (the classes were predefined by Severstal company) for each image, where each image may have no defects, a defect of a single class, or defects of multiple classes. To increase the size of training data, the study implemented image augmentation with the Albuminations library and compared the results with and without augmentation. The study tested U-Net models with different backbones to determine the most effective model for this task. The study used mean Dice coefficient and other possible metrics to evaluate the performance of the models. Overall, the study aimed to improve the accuracy of detecting and classifying defects in steel manufacturing, which is an important problem in the manufacturing industry.

Defect detection in steel manufacturing is typically done through visual inspection by trained human inspectors or automated systems using cameras and image analysis techniques. The limits of the human-involved practice include the following: the inspection can be subjective, time-consuming, and prone to errors due to different factors. Moreover, practices may not be able to detect defects in real-time, leading to delays in the manufacturing process.

The implementation of deep learning can help make production of steel more efficient and cost-effective. Furthermore, it may improve product quality, improve efficiency, reduce waste and increase competitiveness in the industry. In this study we evaluated different models used for detecting and classifying defects in steel manufacturing and we tried to improve their accuracy by implementing image augmentation.

The data used in the study can be found here. The complete analysis and code of our project can be found in out gihub repository github. The dataset used for this study consists of two folders of images in JPG format, images have dimensions of $(256, 1600, 3)$. The first folder, train_images, contained 12,568 images used for training and evaluating the model's performance, while the second folder, test_images, was used for submission on Kaggle. The OpenCV library's imread function was utilized to read and resize the images to a size of $(128, 800, 3)$.

Annotations for the images in the train_images folder are provided in the train.csv file. Each ImageId in the CSV file has four rows, one for each of the four defect classes (ClassId = [1, 2, 3, 4]). The pixel numbering convention follows a top-to-bottom, left-to-right order, such that the first pixel is (0,0), and the second pixel is (1,0). The pixel number for a pixel located at (i, j) in the image is calculated as $n = 256j + i + 1$. Conversely, the row and column indices (i, j) can be computed from the pixel number n as follows: $j = \text{int}(n/256)$, $i = n - 256j - 1$.

The target for each image is represented by four masks, one for each class of defect. These masks are generated based on the pixel numbers of the defects present in the image.

2. Approach

The objective of our study was to address a semantic segmentation problem using the U-Net architecture. Given that U-Net [4] was initially developed for medical imaging segmentation and is one of the earlier deep learning segmentation models, we considered it a suitable approach for our problem. To begin, we used the U-Net model from the Seg-

mentation models library segmentation_models, which is a Python library that leverages neural networks for image segmentation based on the Keras (Tensorflow) framework [2].

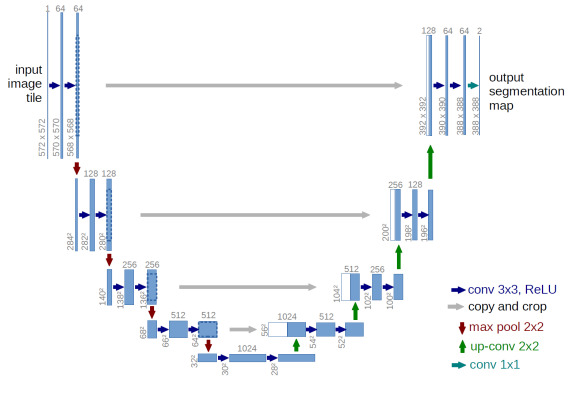

Figure 1. U-net-architecture

We were able to modify the network architecture by selecting backbones with different parameters, and pre-trained weights were used to initialize it. We compared the performance of U-Nets with various backbones, including ResNet34, ResNet50, and ResNet101. To increase the size of the training dataset, we applied image augmentation with the Albumentations library and compared the results with and without augmentation. Pre-trained weights from Imagenet were used for all models.

The use of the U-Net architecture for semantic segmentation is not a new approach, as it has been used in previous studies. However, our study is unique in that we compared the performance of U-Nets with various backbones and employed image augmentation to increase the size of the training dataset. The use of pre-trained weights from Imagenet is a common practice in deep learning, but its application in our study is relevant given the size and complexity of the dataset. Overall, our approach builds upon established techniques and libraries commonly used in semantic segmentation tasks.

In our study, we anticipated potential challenges in dealing with a complex dataset, as well as issues with overfitting and limited training data. We also recognized that selecting the appropriate backbone for the U-Net architecture could impact the model's performance.

During the course of our study, we encountered some difficulties with balancing the trade-off between model accuracy and computational efficiency. Additionally, the time and resources required for training and evaluating multiple models with different backbones was a limitation. Our initial approach of utilizing the U-Net model from the Segmentation models library did work, but we had to experiment with various backbones and augmentation to optimize the model's performance.

After comparing the results of prebuilt architectures, we sought to further improve our segmentation model by implementing our own U-Net using Keras layers. The U-Net architecture consists of an encoder and a decoder, with skip connections between them to preserve spatial information.

3. Experiments and Results

For our image segmentation task, we have chosen the binary cross-entropy (BCE) loss, which is commonly used for binary classification problems. It measures the differences between the actual and predicted image masks by computing the cross-entropy between the two distributions.

To optimize the parameters of our models, we used the Adam optimizer, a popular choice in deep learning due to its several benefits. It is known for its simplicity of implementation, fast computation time, low memory requirements, and requiring less tuning than other optimization algorithms.

The performance of the models is evaluated on the mean Dice coefficient. This was the main metric that we used to evaluate our models. The Dice coefficient is commonly used to measure the overlap between the predicted segmentation mask and the ground truth segmentation mask. The formula is given by:

$$\text{Dice Coefficient} = 2|X \cap Y|/(|X| + |Y|),$$

where X is the predicted segmentation mask and Y is the ground truth segmentation mask. The Dice coefficient ranges from 0 to 1, where a value of 1 indicates a perfect overlap between the predicted and ground truth masks, and a value of 0 indicates no overlap at all. In order to avoid over-fitting while training we saved the model that had the best mean dice coefficient on validation set. Another metric that we used to evaluate the performance is the Intersection-Over-Union (IoU), also known as the Jaccard Index. It is one of the most commonly used metrics in semantic segmentation. The formula is given by:

$$\text{IoU} = 2|X \cap Y|/(|X \cup Y|),$$

where X is the predicted segmentation mask and Y is the ground truth segmentation mask. From the formula we can see that the IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth. This metric ranges from 0 to 1 (with 0 signifying no overlap and 1 signifying perfectly overlapping segmentation). For binary or multi-class segmentation, the mean IoU of the image is calculated by taking the IoU of each class and averaging them.

3.1. U-Net from segmentation models library without augmentation

We trained multiple U-Net models with different backbones from the segmentation library and present their

corresponding results in the following plots.
All models were trained for 20 epochs. We used early stopping with patience 10 (monitoring Dice coefficient on training) and we reduced learning rate when on plateau with factor 0.33 (monitoring dice coefficient on validation) and patience 5. The initial learning rate was set to the default value of 0.001
The metrics that we achieved training U-Net with backbone ResNet34 and pre-trained weights (from ImageNet) are:

```
loss:  0.0101
dice coefficient on training:  0.6149
iou coefficent:  0.3074
binary accuracy:  0.9961
dice coefficient on validation 0.60970
```

Figure 2. U-Net with backbone=ResNet34 and pre-trained weights: Training history

The metrics that we achieved training U-Net with backbone ResNet34 (without pre-trained weights) are:

```
loss:  0.0098
dice coefficient on training:  0.6203
iou coefficent:  0.3101
binary accuracy:  0.9963
dice coefficient on validation 0.61205
```

Figure 3. U-Net with backbone=ResNet34: Training history

The metrics that we achieved training U-Net with backbone ResNet50 and pre-trained weights (from ImageNet) are:

```
loss:  0.0076
dice coefficient on training:  0.6915
iou coefficent:  0.3457
binary accuracy:  0.9970
dice coefficient on validation 0.63640
```

The metrics that we achieved training U-Net with backbone ResNet50 (without pre-trained weights) are:

```
loss:  0.0090
dice coefficient on training:  0.6445
iou coefficent:  0.3223
```

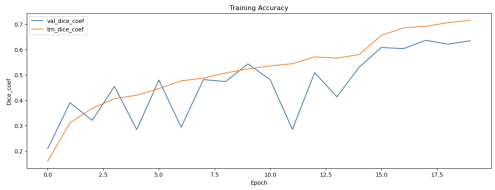

Figure 4. U-Net with backbone=ResNet50 and pre-trained weights: Training history

```
binary accuracy:  0.9965
dice coefficient on validation 0.59371
```

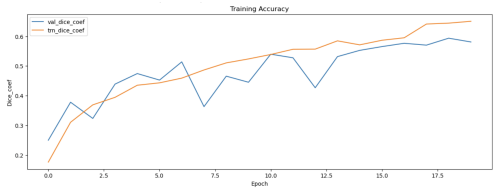

Figure 5. U-Net with backbone=ResNet50: Training history

The metrics that we achieved training U-Net with backbone ResNet101 and pre-trained weights (from ImageNet) are:

```
loss:  0.0099
dice coefficient on training:  0.6138
iou coefficent:  0.3069
binary accuracy:  0.9962
dice coefficient on validation 0.60203
```

The

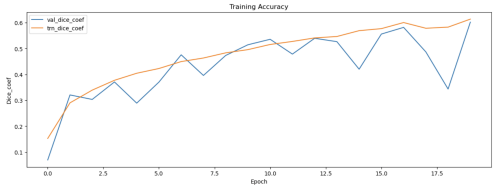

Figure 6. U-Net with backbone=ResNet101 and pre-trained weights: Training history

metrics that we achieved training U-Net with backbone ResNet101 (without pre-trained weights) are:

```
loss:  0.0086
dice coefficient on training:  0.6619
iou coefficent:  0.3310
binary accuracy:  0.9966
dice coefficient on validation 0.57535
```

3.2. U-Net from segmentation models library with augmentation

We use data augmentation [3] to increase the complexity (possibly increase number of samples when training on

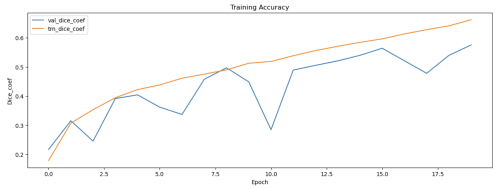

Figure 7. U-Net with backbone=ResNet101: Training history

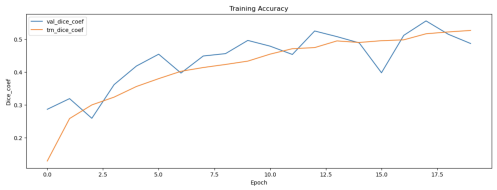

Figure 8. U-Net with backbone=ResNet34: Training history

both: augmented and non-augmented data) and introduce more variance in the training dataset. Data augmentation helps to reduce overfitting. We can apply data augmentation and produce a variety of images with different orientation, location, scale, brightness etc. These images would introduce variance and make the model more robust. Albumentations library provides many different functions that can be used to transform the images and their corresponding masks and create an augmented dataset [1].

In our study to transform the training dataset we implemented the sequence of following functions:

1) HorizontalFlip with probability 0.5;
2) VerticalFlip with probability 0.5;
3) ShiftScaleRotate with probability 0.3 and the parameters shift_limit=0.01, scale_limit=0.06, rotate_limit=0;
4) RandomGamma with probability 0.25;
5) Blur with probability 0.5 and the parameter blur_limit = 3.

To ensure that models with data augmentation are properly validated, we train these models on an augmented training dataset and evaluate their performance using the original, non augmented validation data. This approach enables us to accurately assess the generalization capabilities of the model on real-world data.

All models were trained for 20 epochs. We used early stopping with patience 10 (monitoring dice coefficient on training) and we reduced learning rate when on plateau with factor 0.33 (monitoring dice coefficient on validation) and patience 8. The initial learning rate was set to the default value of 0.001

The metrics that we achieved training U-Net with backbone ResNet34 with augmentation and pre-trained weights (from ImageNet) are:

```
loss:  0.0132
dice coefficient on training:  0.5164
iou coefficent:  0.2582
binary accuracy:  0.9952
dice coefficient on validation 0.5556
```

The metrics that we achieved training U-Net with backbone ResNet50 with augmentation and pre-trained weights (from ImageNet) are:

```
loss:  0.0136
dice coefficient on training:  0.5046
iou coefficent:  0.2523
binary accuracy:  0.9953
dice coefficient on validation 0.53919
```

Figure 9. U-Net with backbone=ResNet50: Training history

The metrics that we achieved training U-Net with backbone ResNet101 with augmentation and pre-trained weights (from ImageNet) are:

```
loss:  0.0142
dice coefficient on training:  0.4922
iou coefficent:  0.2523
binary accuracy:  0.9950
dice coefficient on validation 0.50117
```

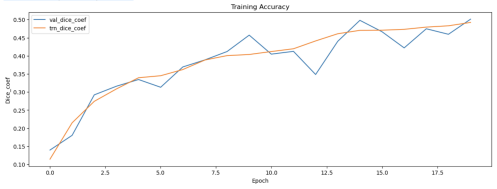

Figure 10. U-Net with backbone=ResNet101: Training history

3.3. U-Net built from Keras layers without augmentation

When training U-Net we used early stopping with patience 8. We trained the model for 20 epochs. The metrics that we achieved are:

```
loss:  0.0157
dice coefficient on training:  0.4507
iou coefficent:  0.2254
binary accuracy:  0.9947
dice coefficient on validation 0.4798
```

3.4. U-Net built from Keras layers with augmentation

When training U-Net with augmentation we used early stopping with patience 8. We trained the model for 20

Figure 11. U-Net from keras layers: Training history

epochs. The metrics that we achieved are:

```
loss:  0.0225
dice coefficient on training:  0.2867
iou coefficent:  0.1434
binary accuracy:  0.9934
dice coefficient on validation 0.3580
```

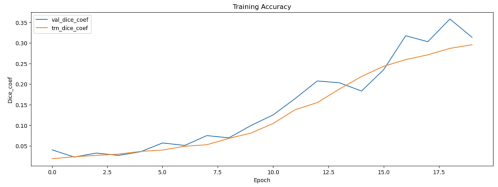

Figure 12. U-Net from keras layers with augmentation: Training history

3.5. Building the best model

After analysing the performances of different models that we tested, we decided to attempt to build the best model: we trained U-Net with backbone ResNet50 with pre-trained weights from ImageNet for 50 epochs and after that retrained it on the augmented data for another 50 epochs. After training on original data for 50 epochs we achieved the following results:

```
loss:  0.0022
dice coefficient on training:  0.9062
iou coefficent:  0.4531
binary accuracy:  0.9991
dice coefficient on validation 0.6754
```

After re-

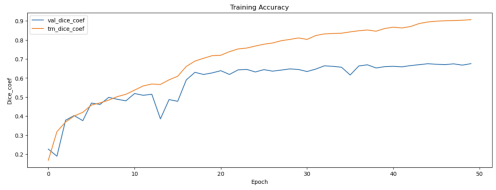

Figure 13. Best model on original data: Training history

training the model showed the following results:

```
loss:  0.0079
dice coefficient on training:  0.6813
iou coefficent:  0.3406
binary accuracy:  0.9969
dice coefficient on validation 0.66180
```

The accuracy of retrained model is worse than the one trained on the original data.

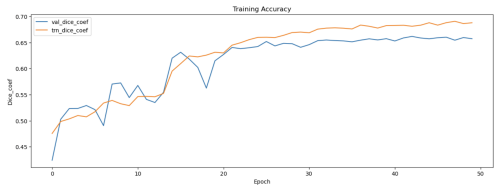

Figure 14. Best model on augmented data: Training history

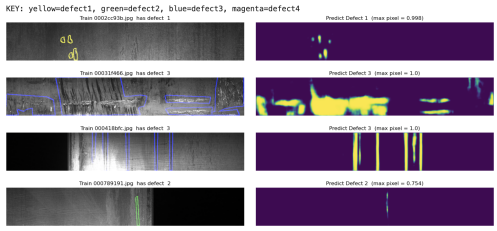

Figure 15. Predictions vs train data for the best model trained on the original data.

3.6. Experiments Summary

In our experiments, we evaluated the performance of a custom U-net model created from Keras layers against several pre-trained models. Our results demonstrate that all pre-trained models outperformed the custom U-net model. Thus, our findings suggest that pre-trained models are a more effective approach for medical image segmentation tasks than training custom models from scratch. What is more, using pre-trained weights from ImageNet increased the dice coefficient in 2 out of 3 cases (ResNet34,ResNet101).

We observed a positive trend among models trained on augmented data, as their performance on real-world data, measured by the dice coefficient on the validation set, was consistently higher than on the training set. This indicates that training on augmented data can improve the model's ability to handle variability in real-world scenarios. However, the models that were trained on augmented data still exhibited lower performance on the validation set compared to those trained on the original dataset.

4. Other Sections

4.1. EDA

The predominant class of defects is class 3, since 5150 images contain it. In second place is class 1 with 897 im-

ages, and in third place is class 4 (801 images). Defects of class 2 has the fewest number of images (247 images).

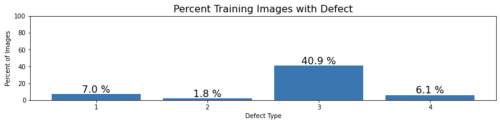

Figure 16. Percentage of training images with defect of each class.

Since we have binary masks (1 or 0), we can count the number of pixels with each class of defects in the mask in order to somehow estimate the size of the defects of each class and see how it depends on the defect class.

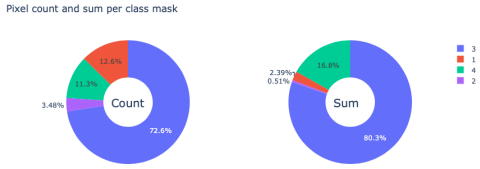

Figure 17. On the right - number of images containing defects of each class. On the left - total number of pixels containing defects of each class.

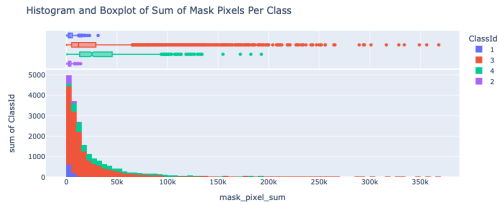

Figure 18. Histogram and box plot for size of masks of each class.

From the histogram and boxplot we can be confirm that defects of class 4 are usually larger than defects of class 3 and much larger than defects from classes 1 and 2. Defects of class 3 contain more outliers.
Although defects of class 4 are usually larger than defects of class 3, outliers of class 3 can be significantly larger than of class 4.

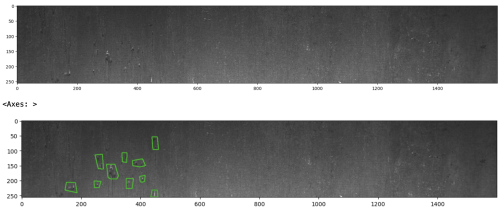

Figure 19. Sample Image with defect of class 1.

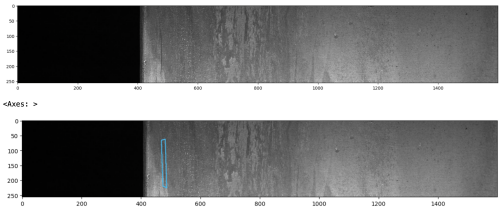

Figure 20. Sample Image with defect of class 2.

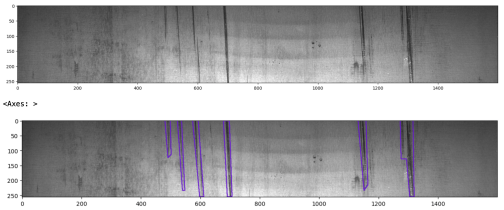

Figure 21. Sample Image with defect of class 3.

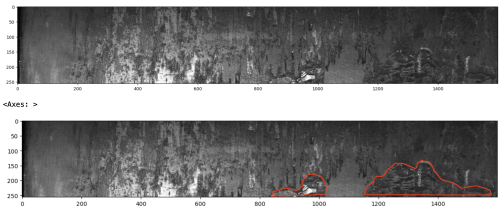

Figure 22. Sample Image with defect of class 4.

5. Work Division

References

[1] Olga Chernytska. Complete guide to data augmentation for computer vision. *complete-guide-to-data-augmentation-for-computer-vision*, 2021. 4

[2] Harshall Lamba. Understanding semantic segmentation with unet. *understanding-semantic-segmentation-with-unet*, 2019. 2

[3] Alvaro Fuentes Dong Sun Park Mingle Xu, Sook Yoon. A comprehensive survey of image augmentation techniques for deep learning. *arxiv.org/pdf/2205.01491.pdf*, 2022. 3

Student Name	Contributed Aspects	Details
Vera Katasonova	Modeling. Implementation of augmentation. Analysis of results	Implemented augmentations. Training U-net with augmentation (with and without augmentation).
Ksenia Katasonova	Data Analysis. Implementation of augmentation. Modeling	Implemented Data Generator with possible augmentation. U-Net from keras layers. Trained U-nets without augmentation. Best model training.

Table 1. Contributions of team members.

[4] Thomas Brox Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation. *arXiv:1505.04597*, 2015. Cornell University. 1