# Teeth Instance Segmentation on X-ray Images

Yuting Ji Robert Smithers Daniella Zunic

## 1. Introduction

The use of computer-aided design (CAD) tools has surged in recent years in the orthodontic field. The addition of quick and precise tools enables orthodontic experts to provide a faster and more accurate understanding of a patient's teeth. These tools can dramatically help dentists simulate teeth extraction, movement, deletion, and rearrangement. In addition to the benefits dental experts encounter, clients also experience a more enjoyable dentistry process, enabling single visit treatments, digital impressions for future use, and further improving the cost-effectiveness of treatment.

There are numerous problems faced in the dental industry in pursuit of these improvements. For X-ray images, common problems include unclear images that are overly bright or overly dark. Labeling also becomes difficult in children when they have new teeth growing underneath older ones as an X-ray sometimes cannot accurately capture that. The same applies to significant gaps in patients' teeth as well as teeth that grow in abnormal positions. These issues make it hard for dentists to figure out the exact problem associated with patients. Furthermore, sometimes there are numerous patients who need to wait for the dentist to interpret their problems from their X-ray results, which leads to low efficiency.

By using a machine-learning-driven automated algorithm, we hope to segment and label teeth to increase the efficiency and accuracy of detecting teeth and any dental problems from X-ray images for dentists.

## 2. Related Work

Previously, a group from Düzce University developed a project for segmenting teeth using a basic Unet model, TensorFlow, and a dataset of 105 images[1]. As an encoder/decoder and a model invented for biomedical segmentation, the Unet works well even with relatively fewer data. Strengths besides their model selection include effectively segmenting the teeth and color labeling them using connected components.

However, their approach has two weaknesses: they barely used data augmentation techniques when training their model and their model sometimes fails to capture detail, such as the gaps between teeth and tips of teeth roots.

We decided to use a data set of 116 images and try similar models as them: an Unet model and a Res-Unet model, which is an improved version of Unet. For both models, instead of using PyTorch's built-in loss function, we included customized dice_loss and customized dropout layers in the model. We also applied data augmentation techniques such as cropping and resizing in order to achieve a better model. As for the labeling part, we first found the contours of the images and then applied watershed and connected component segmentation so that the program could label crowded teeth more accurately. Their post-processing step also focused on grayscale morphological and filtering operations, while we relied on OpenCV to extract contours of the segmented teeth and use watershed and connected components to label and color the teeth.

## 3. Method

### 3.1. Pre-Process

The pre-processing focuses on resizing and grayscaling the input original X-ray and masked images. First a function, convert_one_channel, was created to convert an image to one channel as some of them had three channels, despite being grayscale images. The next function used for pre-processing, pre_images, sorted the images in numerical order based on their name, and then iterated through each one to resize the image to 3104x1200. We chose a size of 3104x1200 as the model we used required the dimensions to be divisible by 16. We also expanded the images so the small gaps between the teeth could be more easily identified. Once each was resized, the convert_one_channel function was called to grayscale the image. The new images were downloaded and re-uploaded to Github's data folder, so they could easily be referred to later in the code by their name. [3]

### 3.2. Dataloader

The data loader does three jobs: organize files, split, augment data, and feed them into three loaders. We first initialize a dataset class. For organizing files, it takes images from the ‘masked_img’ folder, renames pictures with the addition of ‘_m’. It was then moved to the ‘original_img’ folder which contains pairs of original and masked images. For the split task, we used train_test_split to extract 70% of the data to become train_data. Similarly, the rest of the data is split by half to become validation and test data. For the augmentation, we tried techniques including vertical flip, horizontal flip, resizing, and cropping. Next, we call the dataset class to obtain data pairs and apply normalization and resizing to these data. The purpose of normalization and resizing is to foster our learning. Eventually, we load the dataset into train_loader; validation_loader; test_loader.

### 3.3. Network Architecture

This project utilizes a UNet architecture with an encoding/decoding structure. As the model goes deeper, it extracts double the channels from the images and half the pixel dimensions, allowing the model to gradually pick up on edges, shapes, and then details as it goes further down.

There were numerous challenges trying to implement this network, mostly due to the fact that we had to transfer a Keras implementation into a more familiar PyTorch one. We used this as a learning opportunity to understand Keras and rewrite the entire model in neat blocks. Ultimately, we faced many obstacles in implementing the shortcuts correctly, and we fear that our implementation may have contributed to our model often getting stuck by zeroing its gradients and predicting 0.5 values for almost every pixel. After many attempts at stabilizing the model, including LeakyReLU, additional batch normalization, reducing input size, and more, the model developed slightly improved results, but it still did slowly. This could be caused by our data being too noisy and some pictures have no teeth or region to segment (even before our add-noise techniques to increase model resiliency). After much trial and error, we were able to stabilize the model using the GPUs provided and by reducing the image dimensions to a smaller 512X512 or 256x256 from the earlier 3104x2000 that we originally were trying to train on.

### 3.4. Post-Process

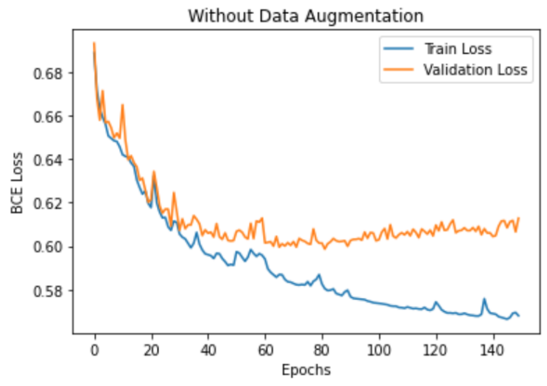
After our model outputs a masked/segmentation image. We start to label the masked image by relying on OpenCV. We first extract the contour of the segmented teeth and apply that to the original image to show the contour of the teeth. Then we use connected components and watersheds to label these teeth in color. Although some watersheds may overlap, with contours, it is sufficient to facilitate dentists to identify teeth and problems associated with teeth. [3]

## 4. Experiments

The data we used is from the test_loader which was declared earlier in the data loader, with each dataset[img],[annotation]. The model is evaluated by calculating the dice loss between the predicted masked img versus the actual masked img. The baseline approach used Unet on Kera. But we are more familiar with PyTorch. So we implemented both Unet and Resnet[5] on PyTorch. Because of its effectiveness in the works of Doctor Olaf Ronneberger[2], we eventually decided on Unet. By including data augmentation, LeakyReLU, and batch normalization. Our model shows a better outcome throughout our experiments.

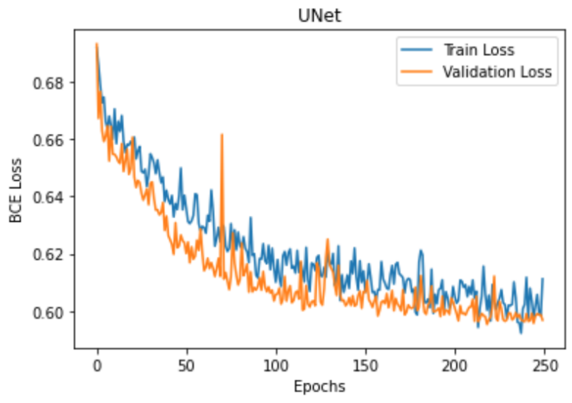
### 4.1. E1 (without data augmentation)

Adam Optimizer @ 0.0001 LR, 150 epochs Using just the base data, we were only able to achieve a BCE loss of about 0.6.



### 4.2. E2 (with basic data augmentation: vertical flip, horizontal flip)

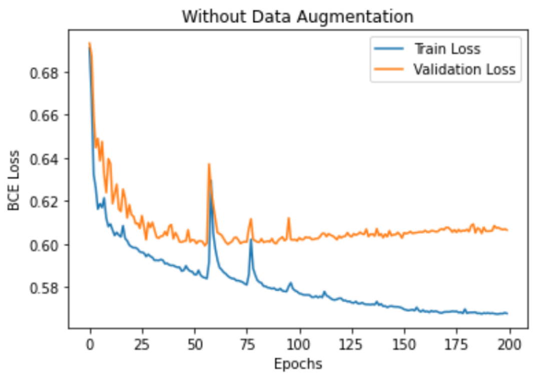
Adam Optimizer 150 epochs @ 0.00003 LR, 100 epochs @ 0.000005 LR



*The inability to achieve a loss lower than 0.59 is partially attributed to issues on our model’s shortcuts. This was fixed by E3. Likewise, the validation loss being lower is attributed to not flipping the validation data.*

### 4.3. E3 (with more data augmentation: crop, resize) (plus LeakyReLU, improved model shortcuts)

Adam Optimizer 150 epochs @ 0.0001 LR + 50 epochs @ 0.00002 LR



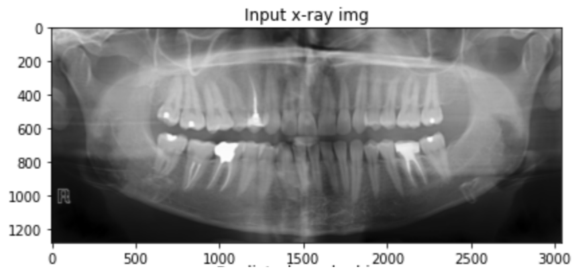
### 4.4. E4 (Using Res-Unet instead of Unet) (LeakyRelu, vertical flip augmentation)

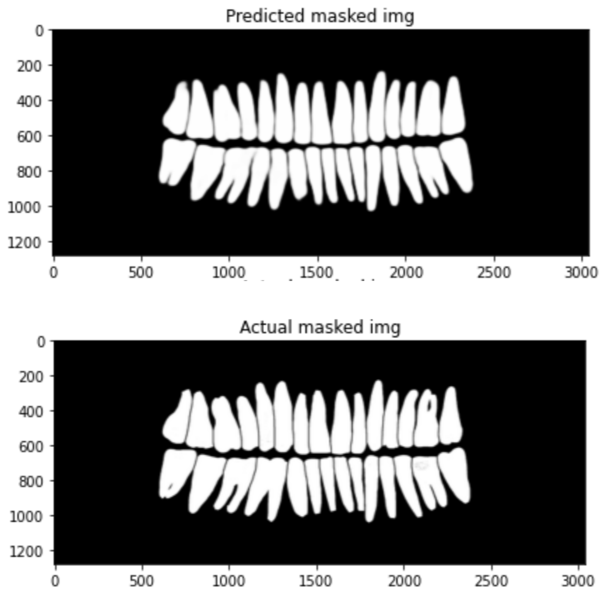
Adam Optimizer @ 0.0001 LR

| Experiment | Epoches | Dice Score |
|---|---|---|
| E1 | 150 | 0.7665 |
| E2 | 150+100 | 0.8140 |
| E3 | 150+50 | 0.9117 |
| E4 | 50+50 | 0.8902 |

Table 1. At E3, we achieve a 91% dice score

### 4.5. Sample input vs output vs mask:





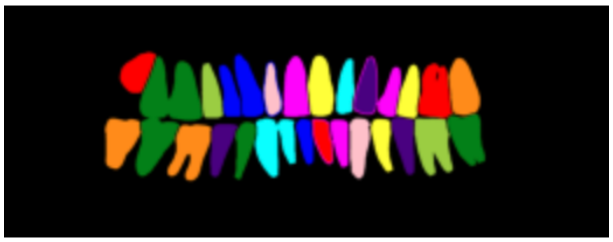


Figure 1. Connected Component Segmentation Result

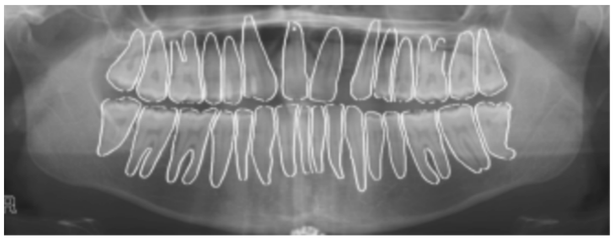


Figure 2. Contour Result

## 5. Conclusion

Compared to baseline methods, our UNet shows a slight improvement in reducing loss. More notably, however, is that upon further analysis, our model does well to identify shapes in the teeth, correctly finding the general outline of tooth structure, but has greater difficulty with some of the finer details of individual teeth. This is demonstrated in output that marks the correct tooth location and outline but fails to properly incorporate unusual tooth ridges, incisions, or minor abnormalities. We may try some other novel models and see how they perform. The biggest takeaway is the importance of implementing shortcuts, leakyReLUs, and additional methods to solve the zero-gradient problem. This may prove useful to the academic community in hopes that further research can be applied to this model in strengthening the high-channel low dimensional data convolved from the lower layers in the diagram.

## 6. Contributions

1. Introduction: Robert, Yuting, Daniella
2. Related Work: Yuting, Daniella
3. Method: Yuting, Robert, Daniella
4. Experiments: Robert, Yuting
5. Conclusions: Yuting, Robert
6. Reference: Daniella
7. Latex Formatting: Daniella, Yuting, Robert
8. Github Project Link

## References

[1] Pytorch features. URL: https://pytorch.org/features/.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, May 2015. URL: https://arxiv.org/abs/1505.04597v1.

[3] S Serdar Helli. Segmentation of teeth in panoramic x-ray image using u-net. URL: https://github.com/SerdarHelli/Segmentation-of-Teeth-in-Panoramic-X-ray-Image-Using-U-Net. 1, 2

[4] Pankaj Singh. Hc18-grand-challenge. URL: https://github.com/Pankaj1357/HC18-Grand-Challenge.

[5] Nikhil Tomar. Semantic-segmentation-architecture/pytorch/resunet.py, May 2021. URL: https://github.com/nikhilroxtomar/Semantic-Segmentation-Architecture/blob/main/PyTorch/resunet.py.