Semantic Segmentation with Deep Learning for Microstructural Characterization and Analysis

Ricardo Zambrano
University of Maryland
College Park, U.S.A.
rzambrano@gmail.com

***Abstract*— This is a pilot project meant to leverage prior studies in the field of semantic segmentation of metallographic specimens. The goal of the project was to train a model for microstructure segmentation, a common task in metallography. The selected technique for the task was using a convolutional neural network architecture known as U-Net. Several experiments were run and the results were benchmarked against the results obtained by the neural network trained on the MicroNet dataset, the work Stuckner et al. It was found that with the U-Net architecture the performance of the intersection over union accuracy metric improved with the number of training samples as well as with incrementing the training epochs. Results were not as good as those obtained by the model trained with the MicroNet dataset. However, the U-Net architecture proved valuable for the task of semantic segmentation of metallographic specimens. To advance this field of study a dataset comparable in size with MicroNet is much needed, thus there is an urgent need to compel organizations to share their data.**

Keywords—computer vision, deep learning, U-Net, semantic segmentation, microstructure, microscopy, metallography

I. INTRODUCTION

Metallography is the study of the microstructure of all types of metallic alloys. This technique was introduced in the 1940s as a means to study the physical properties of metallic materials by observing the structures present in the material when inspected under the microscope.

In short, the microstructure of a material are the structures observed at microscopic level. When typically mentioned, the microstructure are defects, impurities, grains, and grain boundaries. Anything that is not regular from a given crystalline structure is a microstructure. Useful definitions in this domain include:

- **Defects** in general are simply errors or interruptions in the uniform crystalline lattice.
- **Impurities** are atoms (like dirt) that don't belong in the regular crystalline structure.
- **Grains** are pure crystals or uniform sections of crystal growth.
- **Grain boundaries** are boundaries around the separated grains.

In metallurgy, the study of microstructure using metallographic techniques pays special attention to observing the arrangement and distribution of the different phases within a metallic alloy sample (e.g. a sample of steel, which is an alloy of iron and other alloying elements including carbon). For individual alloy compositions, different phases are identified by the shape of grains as well as the location of impurities. Each distinct arrangement is categorized as a typical microstructure. In general, a phase diagram maps the microstructure of a solid solution at given temperatures.

Different microstructures arise from a variety of factors such as: composition, heat treatment, cooling rate, and processing methods. The microstructure of steel stores the material's genesis. Most importantly, it determines the material's chemical and physical properties.

Traditionally, the analysis of metallographic specimens was carried by hand. In particular, the analysis included making measurements in a metallographic image using a ruler and then estimating areas of grains in the given image. These procedures have been govern and maintained by ASTM standards. Advancements in image processing techniques have helped further standardize metallographic analysis as they have increased expert consensus. This have been achieved by automating the grain's area estimation.

Materials scientist are currently pursuing reverse the metallographic analysis process. That is, using machine learning to make possible starting with the desired material's chemical and physical properties, introduced as inputs in a given model, to produce as an output the alloy's composition and the heat treatment required to manufacture such material. The first step in this line of research is using cutting edge technology in computer vision and deep learning to analyze the microstructure in metallographic specimens.

This project was meant to be a pilot test that will leverage previous research in the fields of metallography and computer vision. The goal of the project was to train a deep learning model for microstructure segmentation. Originally the project aimed to train a model that followed the encoder-decoder architecture

implemented by Stuckner et al in the paper "Microstructure segmentation with deep learning encoders pre-trained on a large microscopy dataset".

In the final version of the project a U-Net architecture was trained on a small-sized dataset. The dataset used in this project is the "Aachen-Heerlen annotated steel microstructure dataset", one of the few publicly-available annotated dataset for microstructure analysis.

The developed system expects a microscopy image with magnification from 50x to 4000x, which is the typical rage used in metallography. These images are taken with a standard light microscope with the ability to resolve features of around 0.2 micrometers and larger.

The trained model detects microstructural features in a given micrography and returns a semantic segmentation mask as an output. These masks categorize each pixel in a given micrography image into a class or object. In this particular case it classifies each pixel in the given input into one of two phases expected to be observed in the material.

II. Literature Survey

This project was based mainly in two articles:

1. Stuckner,J., Harder, B., and Smith, T.M. "Microstructure segmentation with deep learning encoders pretrained on a large microscopy dataset", npj Computational Materials volume 8,Article number: 200 (2022)

   https://doi.org/10.1038/s41524-022-00878-5

2. Iren 1, D., Ackermann, M., Gorfer, J., et al "Aachen-Heerlen annotated steel microstructure dataset", Scientific Data, 8:140 (2021) https://doi.org/10.1038/s41597-021-00926-7

The first article provided the metrics benchmark and the technical approach for this project. The second article provided the dataset used for training.

The literature survey included review of other articles including articles by Elizabeth Holm, one of the most prominent researchers in the field of computational materials science.

Stuckner et al's article has a two-fold focus: (i) using transfer learning by using the first blocks of the convolutional base of models trained on ImageNet, and (ii) fine-tune a model pre-trained with ImageNet using a large dataset called MicroNet. Each one of these approaches reached an intersection over union accuracy of 96.5% and 96.4%.

One of the obstacles replicating the results obtained by Stuckner et al is that the MicroNet dataset contains proprietary data -owned by NASA- that is not publicly available. In the search for a comparable dataset, during the research phase of this project, it was found that most research institutions rely on private datasets. Furthermore, there are not large-scale datasets with microstructure images available to the public. This was a major limitation found in most of the papers reviewed in the literature survey.

Among the datasets available to the public the "Aachen-Heerlen annotated steel microstructure dataset" was selected for this pilot project. It is worthwhile emphasizing the scale differential between MicroNet and the aforementioned dataset. MicroNet has over 100,000 labeled microscopy images from 54 material classes, whereas the "Aachen-Heerlen annotated steel microstructure dataset" has about 2,000 metallographic specimens from one material at a fixed composition and heat treatment. Furthermore, the dataset focuses in steel micrographs with Martensite-austenite (MA) structures in them. In other words, the used dataset is not only smaller but it has not the same variety of specimens that MicroNet has.

III. Methodology

A. Dataset

The "Aachen-Heerlen annotated steel microstructure dataset" is publicly available through the Springer/Nature website. The dataset contains exactly 1,705 scanning electron microscopy images along with a set of 8.909 expert-annotated polygons to describe the geometry of the Martensite-Austenite islands that appear on the images.

From a materials science perspective a dataset such as this one would allow researchers to explore the relationship between the morphology of bainitic steel and its mechanical characteristics. However, this dataset was made available to the public with the explicit purpose of having computer vision researchers and practitioners use the data for training state-of-the-art object segmentation models to detect abstract geometries such as Martensite-Austenite islands.

The dataset was created by having three domain experts annotate the electron microscopy images. Annotators used a web-based 2-D image annotation platform to either mark multiple point of interest (POI) or draw one polygon through consecutive clicks.

The first step of the annotation protocol consisted of marking a point that was inside the blocky Martensite-Austenite region in a given image. The annotators were instructed to provide one and only one POI marker per structure. Therefore, the assumption was made that every marker put by a particular annotator belonged to exactly one Martensite-Austenite structure that is visible on the image. For completeness, the

annotators marked all Martensite-Austenite structures that they could identify on the images.

Even for the experts, detecting bainite structures is a challenging task. As an attempt to decrease the number of falsely identified bainite structures, the authors of the dataset used majority decision to mark features in the image as a Martensite-Austenite structure.

Euclidean Distance measurement was used along with a threshold to calculate agreement. Thus, if the distance between the markers placed by different annotators is smaller than the threshold, the markers are assumed to indicate the same point of interest.

The number of points of interest resulted by the agreement of all three annotators was 2.913, while the number of points of interest agreed by two or three annotators was 8.909. At this step, 63 images were removed from the dataset because there were no agreed-upon points of interest marked on them.

After having the list of agreed points of interest coordinates, the expert annotators performed the segmentation task using the annotation software. Each of the 8.909 agreed points of interest were displayed on the base image and the annotator was asked to draw a bounding polygon around the blocky structure. In order to avoid distraction, the expert was provided with one points of interest at a time and drew only one polygon to segment a single bainite structure at a time. After the completion of each segmentation task, the annotation software displayed a randomly chosen points of interest from the agreed-upon points of interest list. The polygons were represented as an ordered list of point tuples with x and y coordinates that resulted from the mouse clicks of the expert annotators.

Given the methodology followed by Deniz Iren et al to develop the "Aachen-Heerlen annotated steel microstructure dataset" the resulting binary segmentation masks were provided as x, y coordinates in a .csv file.

In other datasets available for binary semantic segmentation, it is common to have a structured dataset with a folder containing the original images and another folder containing the binary segmentation masks saved as images. This was not the case for the "Aachen-Heerlen annotated steel microstructure dataset", where the mask was provided as coordinates in a .csv file. Thereby, an extra preprocessing step was required to reconstruct the binary segmentation masks.

B. Reconstructing Mask Dataset

In the .csv file provided in the "Aachen-Heerlen annotated steel microstructure dataset" in lieu of binary segmentation masks, each line represents an annotated polygon in a given image. The structure of the data is as follows, each line contains:

- The name on the image, provided as a string, in the following format: 'IMG_XXXXX.png' (where 'X' represents an integer)
- The coordinate of the point of interest's centroid, provided as a string. This is the '(x,y)' coordinates of the marked point inside the blocky Martensite-Austenite region
- A list of tuples, passed as a string, with the (x, y) coordinates of the vertices of the polygon that demarcated the border of the point of interest

Images with more than one polygon were listed more than one time in the .csv file.

The first step to reconstruct the binary segmentation masks was writing a function to parse the lines in the .csv file. This function expected as an input a line with the data structure described above and returned a list with the following elements: a string (image name), a numpy array with the coordinates of the Martensite-Austenite centroid, and a list of numpy arrays with the coordinates of the vertices of the annotated polygon.

After parsing the .csv file a counter object (from Python's collections library) was used to identify how many polygons were associated with a given image. Figure 1 showcases the image names along with the polygons associated with them. These were the images with most annotated polygons.

```
files_counter = Counter(files)
files_counter.most_common(15)
```

```
[('IMG_01732.png', 22),
 ('IMG_01604.png', 21),
 ('IMG_00737.png', 21),
 ('IMG_01700.png', 19),
 ('IMG_01887.png', 19),
 ('IMG_00536.png', 19),
 ('IMG_01729.png', 18),
 ('IMG_00544.png', 18),
 ('IMG_00738.png', 18),
 ('IMG_01097.png', 17),
 ('IMG_01724.png', 16),
 ('IMG_01603.png', 16),
 ('IMG_01619.png', 16),
 ('IMG_01633.png', 16),
 ('IMG_01742.png', 16)]
```

Figure 1. Top fifteen images with the most associated polygon structures

To represent the polygons a data structure was designed: the 'annotated_polygon' class. Python's dataclass object from the dataclasses library was used. This data structure would represent an annotated polygon by storing its centroid and its vertices.

Given that a binary segmentation mask would have several polygons, another data structure was created to gather all the 'annotated_polygons' associated with a given image: the 'img_mask' class. This object was also designed using the

dataclass template. The name of each object would be the name of the image as provided in the "Aachen-Heerlen annotated steel microstructure dataset" and the data structure could support any number of polygons.

The 'annotated_polygons' class as well as the 'img_mask' class that would be associated with a given image are showcased in Figure 2 and Figure 3.

```
@dataclass(frozen=True)
class annotated_polygon:
    center: npt.NDArray
    nodes: List[npt.NDArray]

    def __str__(self):
        return f'Polygon:\ncenter -> {self.center}\nnodes  -> {self.nodes}'

    def __repr__(self):
        return f'Polygon:\ncenter -> {self.center}\nnodes  -> {self.nodes}'

    def __getitem__(self,position):
        return self.nodes[position]

    def __len__(self):
        return len(self.nodes)

    def get_center(self):
        return self.center

    def get_nodes(self):
        return self.nodes
```

Figure 2. annotated_polygon class

```
@dataclass
class img_mask:
    img_name: str
    polygons: List[annotated_polygon]

    def __str__(self):
        return f'img_mask:\nimg_name -> {self.img_name}\npolygons -> {self.num_polygons()}'

    def __repr__(self):
        return f'img_mask:\nimg_name -> {self.img_name}\npolygons -> {self.num_polygons()}'

    def num_polygons(self):
        return len(self.polygons)

    def add_polygon(self,new_polygon):
        self.polygons.append(new_polygon)

    def __getitem__(self,position):
        return self.polygons[position]

    def __len__(self):
        return len(self.polygons)

    def get_polygons(self):
        return self.polygons

    def get_name(self):
        return self.img_name
```

Figure 3. img_mask class

The next step involved using helper functions to build a representation of the binary image's dataset. The lines_to_polygons function was used to convert the parsed lines into tuples. Each tuple contained the image's name and only one distinct annotated_polygon object.

Then the lines_to_polygons function was used in the body of the build_mask_dataset function. The later function would take the list containing the parsed lines of the .csv file and build a list of img_mask objects. When analyzing each line, this function would check if an img_mask an instance of the object already exists with the same name, if so, it would only add the incoming polygon to the existing img_mask object. Otherwise, it would create a new img_mask instnce with the incoming image name and the annotated_polygon object.

The next step was rebuilding the binary segmentation mask for a given image using the img_mask representation. To achieve this, a helper function called nodes_to_points was written to convert vertices in numpy arrays. The way to extract the vertices was by using the get_nodes() method of the annotated_polygon class.

Then the 'plot_image_and_mask' function would use the get_polygons() method of the img_mask class to get the polygons of an img_mask object. Within the function, vertices would be extracted and typecasted to be plot in matplotlib using the nodes_to_points function. At this point the 'plot_image_and_mask' function would have loaded the original micrography image using the image name. The result would be a visualization of the image with an overlay of the recovered annotated polygons.

Figure 4 showcases the vertices of an annotated polygon as well as the filled polygon. Figure 5 showcases the visualization result of the 'plot_image_and_mask' function.

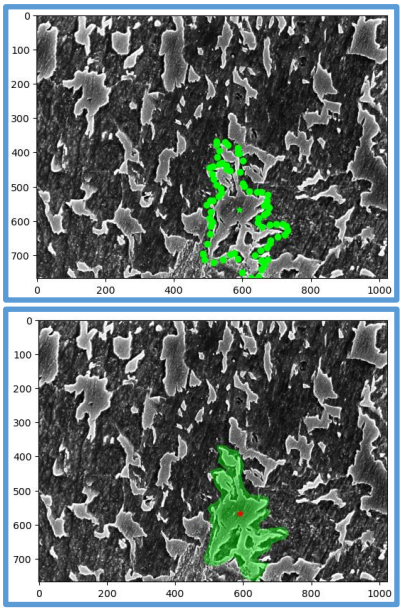

Figure 4. Rebuilding an individual annotated polygon from an annotated_polygon object

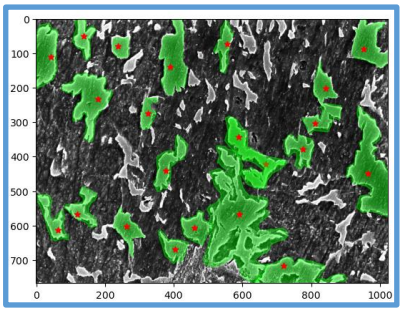

Figure 5. Output of 'plot_image_and_mask' function: the original microscopy image with all the recovered annotated polygons. Build from the 'img_mask' data structure used to represent a binary segmentation mask

The final step in the pre-processing stage was rebuilding a dataset with images of the binary segmentation masks from the representations recorded in the 'img_mask' objects. To do this task the 'create_binary_mask' function was developed. This function takes an 'img_mask' object as an input and returns a

matplotlib figure. An example of the output of the 'create_binary_mask' function is showcased in Figure 6.

Figure 6. Output of the 'create_binary_mask' function using the image name displayed in Figure 5

With this functions and data structures (classes), the .csv file could be traversed with the goal of rebuilding the dataset of binary segmentation mask images. A small sample of the original images and the corresponding rebuilt segmentation masks is showcased in figure 7.

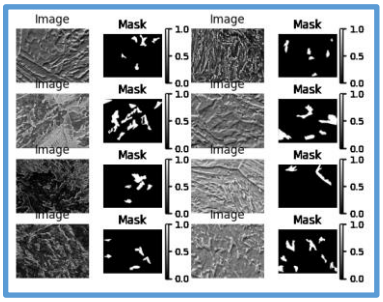

Figure 7. Sample of the rebuilt dataset for semantic segmentation

C. Model Architecture

U-Net is a convolutional neural network originally developed for segmenting biomedical images. It was first introduced in the "U-Net: Convolutional Networks for Biomedical Image Segmentation" paper. The primary purpose of this architecture was to address the challenge of limited annotated data in the medical field. This network was designed to effectively leverage a smaller amount of data while maintaining speed and accuracy. Figure 8 depicts the architecture of the U-Net neural network.

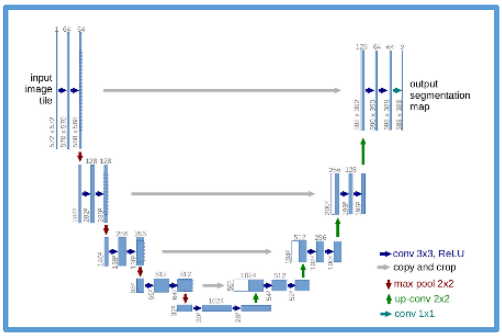

Figure 8. U-Net architecture

The U-Net architecture is made up of two parts, the left part or input part is known as the contracting path. In the contracting path the image is reduced in size while the number of channels increases. The contracting part is the encoder side of the architecture. The right part, that goes from the image with 1024 channels toward the final output (in the case of four downhill steps in the contracting path) is known as the expansive path. The expansive side is the decoder side of the architecture.

The contracting path is made up of two three-by-three convolutions. The convolutions are followed by a rectified linear unit and a two-by-two max-pooling computation for downsampling.

In general terms, the purpose of the contracting path is to capture context while the role of the expansive path is to aid in precise localization. More specifically, the contracting path in U-Net is responsible for identifying the relevant features in the input image. The encoder layers perform convolutional operations that reduce the spatial resolution of the feature maps while increasing their depth, thereby capturing increasingly abstract representations of the input. This contracting path is similar to the feedforward layers in other convolutional neural networks. On the other hand, the expansive path works on decoding the encoded data and locating the features while maintaining the spatial resolution of the input. The decoder layers in the expansive path upsample the feature maps, while also performing convolutional operations. The skip connections from the contracting path help to preserve the spatial information lost in the contracting path, which helps the decoder layers to locate the features more accurately.

Skip connections, as in other architecture designs such as Res-Net, also help to propagate the information from the earlier layers to the deeper ones, thus helping to alleviate the vanishing gradients' problem by directly passing the information through the network.

For this project the U-Net model was implemented using the nn.Module of PyTorch.

D. Training Procedure

Even though there was a GeForce RTX GPU with CUDA capability available locally, the memory was quickly overflowed when attempts were made to train the network locally. To train the model compute units had to be purchased with Google Collaboratory, a cloud service provider.

Even then then, with an A100 GPU combined with a high-RAM runtime type, there were limits to the training set size and training epochs that could not be surpassed without running out of either RAM memory or GPU memory. Notwithstanding,

interesting experiments were run within the limits imposed by hardware available in the cloud computing provider site.

For each experiment different combinations of training set sizes and training epochs were tried. The range of training epochs went from 5 epochs to 250 epochs. Training set sizes that were tested out ranged from 100 up to 1,700 sampling units. A clarification note is in order at this point: the training set size refers to the total units passed to the train_test_split function. Meaning that the actual training used only 80% of the amount of sampling units specified by the training set size. The reminder 20% of the units were used to test the performance of the model as the training progressed. Training experiments were run in individual Google Collaboratory notebooks.

The first step in the training loop was assigning a subset of image names to a variable. The size of the subset would determine the training set size that would then be passed to the train_test_split function. The next step was splitting the names variable into a training set and a test set using scikit-learn's train_test_split function. The test size was set to 0.2.

Then the folders with the original images and their corresponding mask images (rebuilt from the .csv file) were traversed with the selected image names. In this step, images and binary segmentation masks were loaded into the system. The numpy arrays for both images and masks were transformed in to grayscale images using skimage's rgb2gray() function and stored in a list object. The resulting numpy arrays had only one channel.

The next step was creating a custom PyTorch dataset by creating a class that inherits from torch.utils.data.Dataset object. The final step prior to running the training loop was to load the dataset in a PyTorch DataLoader. The DataLoader parameters were batch_size=16 and shuffle=True. The following list specifies the hyperparameters used during training:

- LEARNING_RATE = 1e-4 (the paper used lr=2e-4)
- NUM_EPOCHS = [5, 10, 15, 20, 25, 50, 250] (Stuckner et al's paper used 1e10)
- IMAGE_HEIGHT = 768
- IMAGE_WIDTH = 1024
- loss_fn = torch.nn.BCEWithLogitsLoss() (Stuckner et al's paper used losses.DiceBCELoss(weight=0.7))
- scaler = torch.cuda.amp.GradScaler()
- optimizer = optim.Adam(model.parameters()

The loss function used was the Binary Cross-Entropy with Logits loss. This loss function combines a Sigmoid layer and the BCELoss (binary cross-entropy) in one single class. PyTorch's version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, they take advantage of the log-sum-exp trick for numerical stability.

E. Predicted Mask Post Processing

In the original test loop, the predicted semantic segmentation masks are post-processed by passing them through the torch.sigmoid() function; and then the binary mask is finished by taking any pixel with a value over 0.5 as 1, otherwise the pixel gets a value of 0 assigned to it.

While inspecting the raw mask (prior to post-processing with the sigmoid function), it was noticed that the raw mask tracked really well the ground truth binary segmentation mask. This fact inspired the idea of using thresholding and opening/closing operations to post-process the mask instead of using the sigmoid function for post processing. Using this approach a single image was tested; when comparing the resulting binary segmentation mask using the two approaches, the intersection over union accuracy went from 0.25 in the image mask processed with the sigmoid function to 0.55 in the image mask processed with thresholding and opening/closing operations (see Figure 9). Given this piece of evidence, when running the model training experiments, resulting mask images were evaluated using both post-processing techniques.

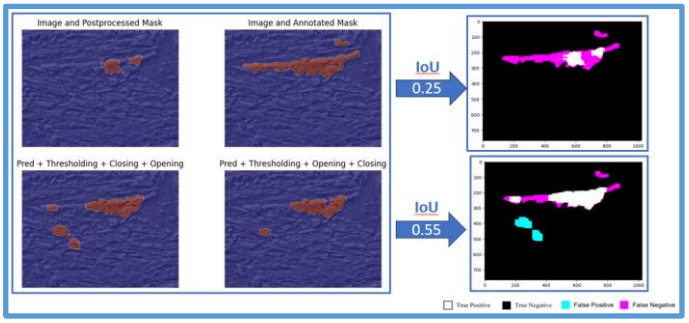

Figure 9. Increase in intersection over union accuracy when using thresholding and opening/closing operations (bottom row of images) instead of the sigmoid function (top lest image). Prediction with 100 training samples over 5 epochs

F. Metrics

The system's effectiveness was be measured using the intersection over union accuracy metric (IoU), also known as the Jaccard index or Jaccard similarity coefficient.

The IoU is computed by taking the predicted mask and ground truth mask for an object and computing the ratio of their area of intersection and their area of union. In the case of Image Segmentation, the area is not necessarily rectangular. It can have any regular or irregular shape. That means the predictions are segmentation masks and not bounding boxes. Therefore, pixel-by-pixel analysis is done.

For this metric there is particular interest in the following definitions:

- **True Positive (TP):** The area of intersection between Ground Truth (GT) and segmentation mask(S). Mathematically, this is logical AND operation of GT and S
- **False Positive (FP):** The predicted area outside the Ground Truth. This is the logical OR of GT and segmentation minus GT
- **False Negative (FN):** Number of pixels in the Ground Truth area that the model failed to predict. This is the logical OR of GT and segmentation minus S

With these values in mind, the intersection over union accuracy (IoU) metric is calculated as follows:

$$IoU = \frac{TP}{TP + FN + FP}$$

The function calculateIoU was developed to calculate the intersection over union accuracy. This function takes as an input the ground truth mask and the predicted mask and returns the calculated intersection over union accuracy.

Another helper function was written to visualize the intersection over union accuracy. This is the graphIoU function, which takes the ground truth mask and the predicted mask as input parameters. The function returns an RGB image, as a numpy array with uint8 data type. In the returned image: pixels in color black represent a true negative, withe pixels represent true positives, magenta pixels represent false negatives, and cyan pixels represent false positives.

IV. Results

A. *Effect of Training set Size*

With the number of training epochs fixed at 20, several experiments were run with varying values for the training set size. The sizes of the training set were 100, 500, 1000, and 1300.

It was observed that increasing the training set size led to an increase in the intersection over union accuracy. It is worthwhile noting that with 20 epochs the intersection over union accuracy of the predicted masks that were post processed using thresholding and opening/closing operations were higher than the intersection over union accuracy attained by the predicted masks that were post processed using the sigmoid function.

In Figure 10 it can be seen that the maximum intersection over union accuracy was not greater than 0.4. This result is by far inferior to the results of intersection over accuracy obtained by Stuckner et al.

B. *Effect of Training Epochs*

With the number of sampling units in the training set fixed at 1300, several experiments were run with varying values for the training epochs. As mentioned before, training epochs were bounded by the hardware capacity and the maximum computing time per session allowed in Google Collaboratory. The experiments were run for 20, 50, and 250 epochs.

As it can be seen in Figure 11, the average intersection over union accuracy metric reached values over 0.7. This is still far from the intersection over accuracy obtained by Stuckner et al. However, the overall results were is driven down by the fact that for some images the intersection over union accuracy was equal to 0. It was noticed that in the images mask images with the worst performance the points of interest in the images (the Martensite-Austenite islands), were tiny, located at corners, and thus difficult to predict. Once question that comes to mind is how the quality of the annotations and micrographies compare between the MicroNet dataset and the "Aachen-Heerlen annotated steel microstructure dataset". Specifically, it is not unreasonable to think about the possibility that annotation quality could drive down the quality of the predicted binary segmentation masks. Figure 12 showcases the ground truth masks vis-à-vis the predicted masks with intersection over union accuracy equal to 0.

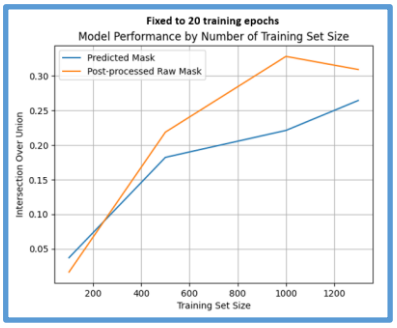

Figure 10. Effect of training set size in intersection over union accuracy metric

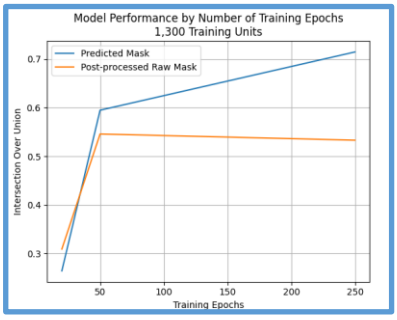

Figure 11. Effect of training epochs in intersection over union accuracy metric

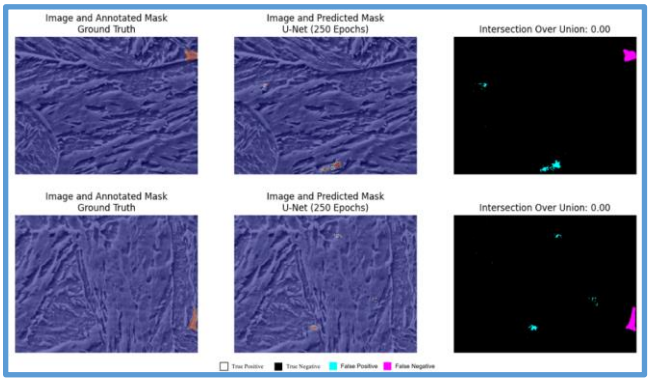

Figure 12. Worst predictions by the best trained model.

When taking 850 random images and predicting the binary semantic segmentation masks using the best training model (1300 samples in the training set with 250 training epoch), about 58% of the images have an intersection over union accuracy higher than .95, which is comparable to the results obtained by Stuckner et al. It has to be noted that the model may have seen some of these images though. Figure 13 depicts the best predictions obtained with the best model.

These results are remarkable given that the model developed by Stuckner et al had a training set with over 100,000 images from 54 material classes, which then were used to train a model in a hardware that allowed the training to last for as long as 1e10 epochs.

Given the resources available for this pilot project, it is consider a remarkable achievement the performance of the U-Net architecture.

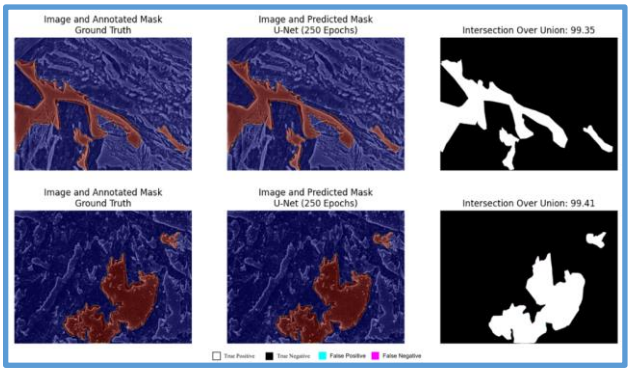

Figure 13. Best predictions by the best trained model.

V. Conclusion and Future Work

Given the little amount of data and the compute capacity available, U-Net proved to be a neural network architecture robust and effective for microstructure analysis. It is possible that cases with poor performance in the intersection over union accuracy metric were driven by cases of poorly annotated images or by the location of the points of interest in the image. This latter conclusion needs to be verified by a domain expert.

False positives occurred in regions of the image that look a lot like Martensite-austenite (MA) islands. These false positives did not occur in areas that were clearly not points of interest. This is a remarkable result and a signal that more training samples and training epoch during model training would result in improved performance on the intersection over union accuracy metric.

A clear result from this pilot test is that intersection over union accuracy metric performance increases with training set size and training epochs. For future work, if better hardware is available, data augmentation could be used to synthetically increase the size of the dataset. Since there is no ‘up’ and ‘down’ direction in metallographic specimens this could be a valid technique to increase the training samples fed to the model during training.

Finally, it was found that using image filters, such as thresholding and opening/closing operations, for post-processing raw the masks was not an effective technique in the aggregate. The intersection over union accuracy metric performance of the masks postprocessed with the torch.sigmoid() function took over the masks postprocessed using the filter techniques when the training epochs were increased. An expected result given that the training loop used the torch.sigmoid() function.

References

[1] Stuckner,J., Harder, B., and Smith, T.M. “Microstructure segmentation with deep learning encoders pretrained on a large microscopy dataset”, npj Computational Materials volume 8,Article number: 200 (2022) https://doi.org/10.1038/s41524-022-00878-5

[2] Iren 1, D., Ackermann, M., Gorfer, J., et al “Aachen-Heerlen annotated steel microstructure dataset”, Scientific Data, 8:140 (2021) https://doi.org/10.1038/s41597-021-00926-7

[3] Holm E. A., Cohn, R., Gao, N., Kitahara, A. R., Matson, T. P., et al “Overview: Computer Vision and Machine Learning for Microstructural Characterization and Analysis”, Metallurgical And Materials Transactions A, Volume 51a, December (2020)

[4] Szeliski, (2022) “Computer Vision”, 2nd Edition, Springer