Machine Learning and Neural Networks for Image Processing

Darren Au

Mateo Valadez

Rushil Patange

Cesar Ayala-Mendoza

University of Texas at Austin

Faculty Mentors

Dr. Michael Cullinan

Dr. Carolyn Seepersad

Eva Natinsky

Doug Sassaman

Background

Dr. Michael Cullinan's work involves working with nanomanufacturing systems which is a series of fabrication procedures used to build nanoscale structures and devices. As part of his research, he needs to take atomic force microscopy (AFM) image scans to analyze his work. AFM is a meticulous procedure that enables the imaging of most types of surfaces. The data that AFM images generate grants a certain environment a high resolution 3D visualization scan from the movement and change of high resolution areas. However, as this process is time-consuming, he was looking to identify feature-rich areas from a larger image before focusing on those areas at a higher resolution. Machine learning techniques involve computer science and artificial intelligence which require a set of data and algorithms. These models and programs can be reflective of the way humans learn new skills by practicing and studying in order to improve their overall efficiency and accuracy after validating itself in some sort of assessment. The purpose of this project would be to accomplish the identification of feature rich areas that undergo a change of gradient by incorporating machine learning techniques in order to do so. These machine learning techniques would use a data set of AFM images and use the masks produced by an edge detection algorithm as the expected output image, then the Machine Learning model will produce a prediction mask and compare it to the algorithm image to validate itself and repeat the process to improve its accuracy. A neural network will also be implemented into this Machine Learning model, for this specific type of deep learning a convolutional neural network is required, a type of artificial neural network that mostly serves for the analysis of visual imagery or image processing.

Objectives

In this project our group had the task to investigate the implementation of neural networks into a machine learning model so it can serve for image processing and the identification of feature rich areas in AFM images. Our group began by learning about neural networks in machine learning and how to build, train, and test these networks on image databases. We were then able to focus on a specific open-source implementation from GitHub and other libraries to practice using machine learning and perform our own image processing.

This investigation aims to develop an algorithm to identify these feature-rich areas in an image. Our group began by creating and testing several filters and techniques to perform this task. Our group then selected the best option and used this algorithm to generate a dataset before putting the dataset into a machine learning model. The architecture of this model is based on UNet, a common image segmentation model typically used for medical images. Through training with a batch size of 1 and an epoch of 5, our group achieved an accuracy of around 80% while testing the model on a validation set.

Methods

This project is predicated on the assumption that areas of high gradients are equivalent to feature-rich areas. From a human analysis standpoint, this has primarily appeared to be true. Consequently, most of the feature-rich detection is referred to as “edge detection” as well.

Our graduate mentor, Eva Natinsky, began by giving us a template of what she envisioned the algorithm would do.

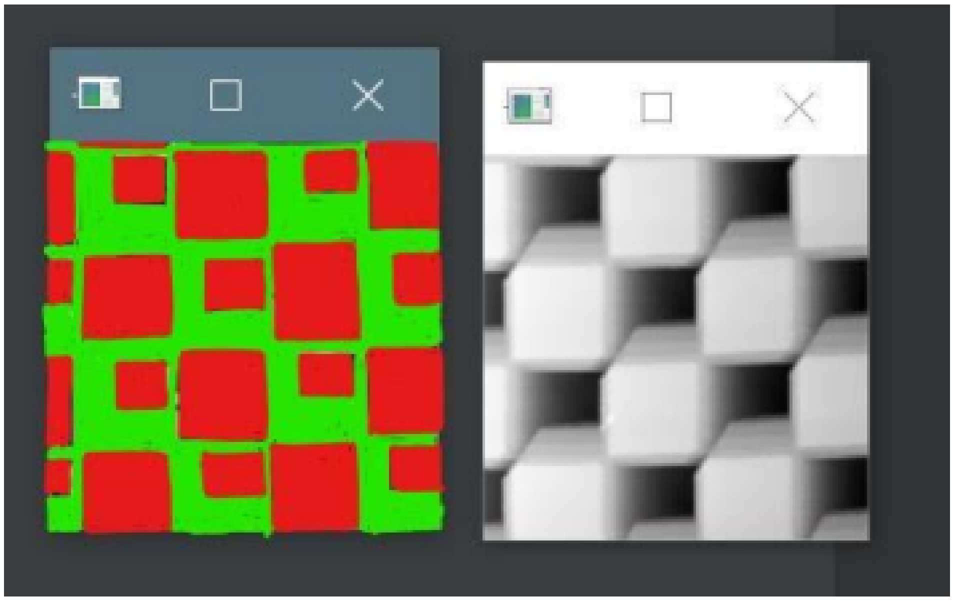
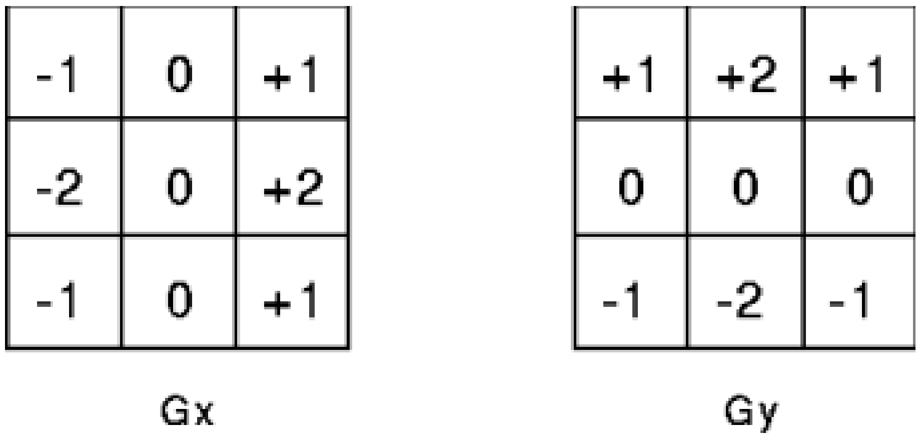


Figure 1: Envisioned Algorithm Results from Template Provided by Eva Natinsky

Our group began by researching methods to detect areas of high gradient. The most common result was using a “sobel filter”, which is a set of matrices convolved across the image. The matrices used can be seen below.



-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 2: Set of Matrices from Using a “Sobel Filter”

The two matrices are set up such that when convolved with the image, it will produce an output matrix whose magnitude depends on the gradient of the original image. Both matrices need to be applied as one determines the gradient in the x-direction while the other does this for the y direction. The two gradient matrices are then combined to a total magnitude that is determined with the distance formula of sqrt(x^2-y^2). By doing this, the resulting output is an image of the gradient magnitudes of the original image.

This image is then converted into a binary image, where values after a certain threshold are set to green while everything else is set to red.

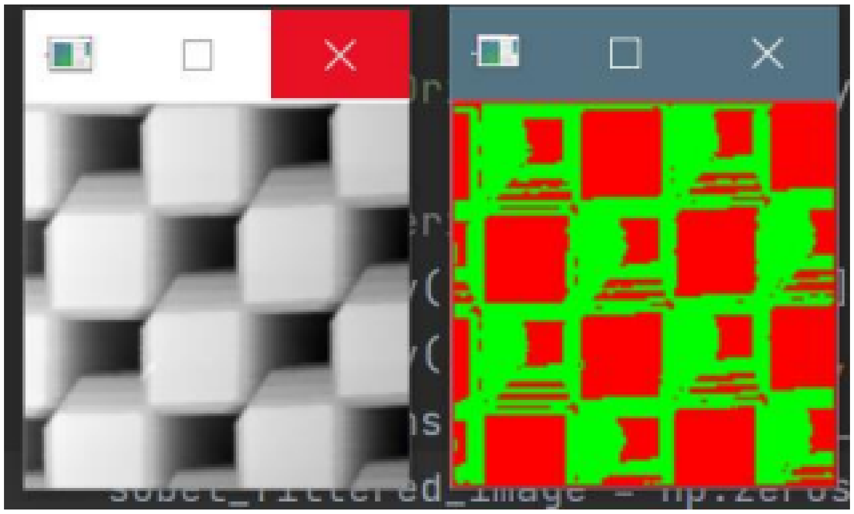


Figure 3: First Iteration of the Algorithm

This was the 1st iteration of the algorithm and, coincidentally enough, also the one that ended up being used. There were many attempts afterward to improve the algorithm; Specifically, attempts to emulate the template given by Eva. However, all of the changes had tradeoffs. All these iterations were created primarily using OpenCV[1] and NumPy[2] libraries. OpenCV and NumPy are open source python libraries focused on computer vision. Modifying these libraries to fit our needs saved a lot of time compared to fabricating a custom solution. We used OpenCV for most of the image manipulation like color to gray and image blurring by applying kernels which change the value of a pixel by merging it with the different values of neighboring pixels. A NumPy was used to manipulate and edit the matrices that were used. Each of the attempts can be found in this drive folder[3].

v1.0.1: multiplied gy by 5

This was just toying around with the filter. Evidently, the final result was very close to the “mask” that our group was aiming for, but it would only work for this specific picture.

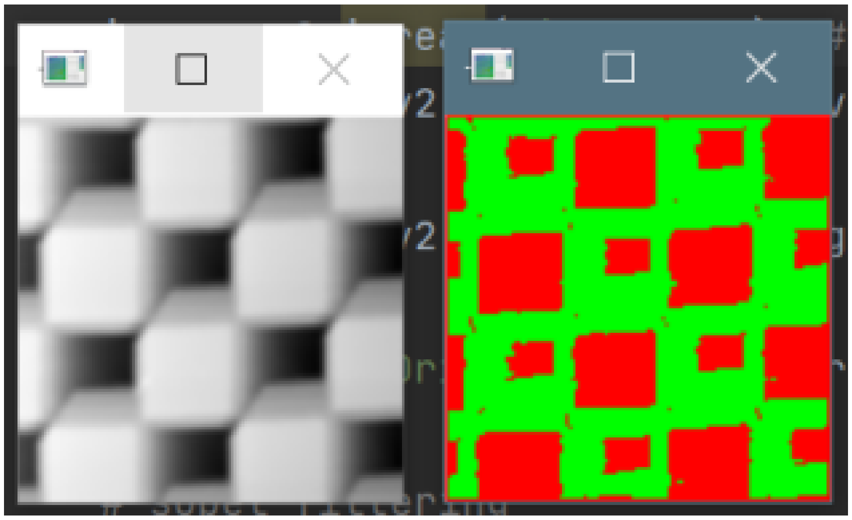


Figure 4: Iteration Results of v1.0.1: multiplied gy by 5

v2.0.0: used contour detection

To reduce the smaller specs of red, our group added a contour detection built into OpenCV that would draw a contour around all the red areas. It would then detect the red areas and only draw them if they met a certain threshold. This resulted in less noise but still missed detecting the small trapezoid below the smaller square as an area of high gradient.

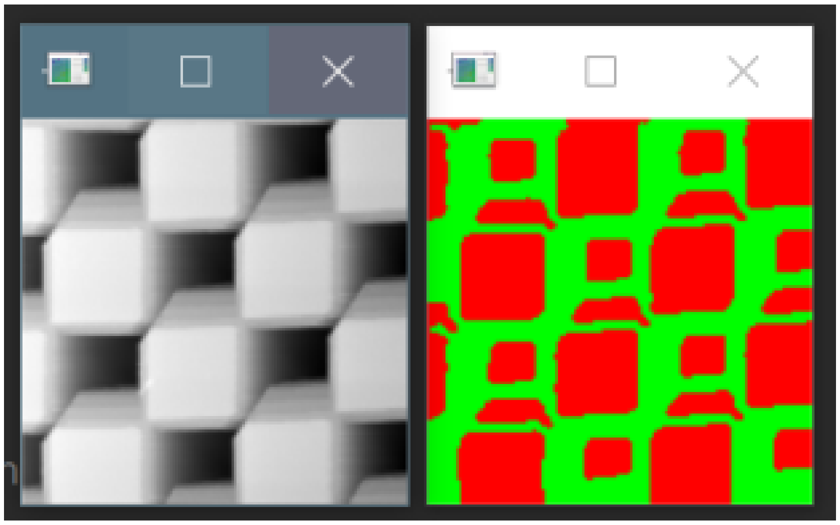


Figure 5: Iteration Results of v2.0.0: used contour detection

v2.0.1: variable threshold

Our group attempted to make the threshold vary based on the average pixels so that it wasn't hard coded and overfitted for that specific image. The results weren't that successful in making the algorithm work for multiple images though.

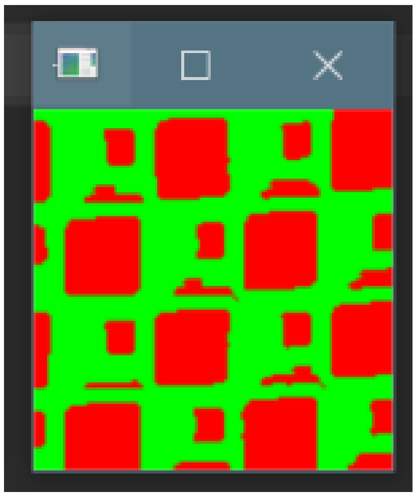


Figure 6: Iteration Results of v2.0.1: variable threshold

v2.0.2: changed so that value based on median instead of mean

The threshold was now based on the median value of the pixels, so the ratio of red and green pixels should in theory be constant. Results once again are seen below.

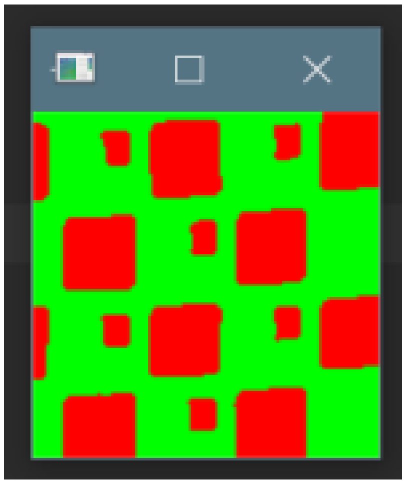


Figure 7: Iteration Results of v2.0.2: changed so that value based on median instead of mean

v3.0.0: output to excel file

Nothing in terms of output changed from v2.0.2 but now, the python file writes each of the gradient and actual values of the image to an excel file. Our group was hoping to use this to help debug and find a new approach to the algorithm

It was also at this point that the deadline for the project was approaching and attempts at altering the algorithm weren't seeing much progress. Each change would try to do something new, but it would also lead to issues with other images, sort of like whack-a-mole were trying to address one will lead to something else popping up. Eva and our group discussed and decided to move forward with the machine learning aspect of the assignment.

This would involve choosing one of the algorithms to form the dataset. Our group ended up choosing the 1st iteration, as despite its shortcomings, was the most stable and consistent among all the images.

Moving onto the Machine Learning portion of the project, the first task was to create a dataset. It was relatively easy to convert the previous code to create the dataset; it simply involved parsing a folder of images, running the edited algorithm found in the OpenCV, and writing it to a folder. Our group found a file of a thousand images online and ran that to create the output. The program also had to be altered such that the output mask was now in black and white instead of red and green which was done through the image manipulation of OpenCV.

An example of one of the images produced by the algorithm is seen below.

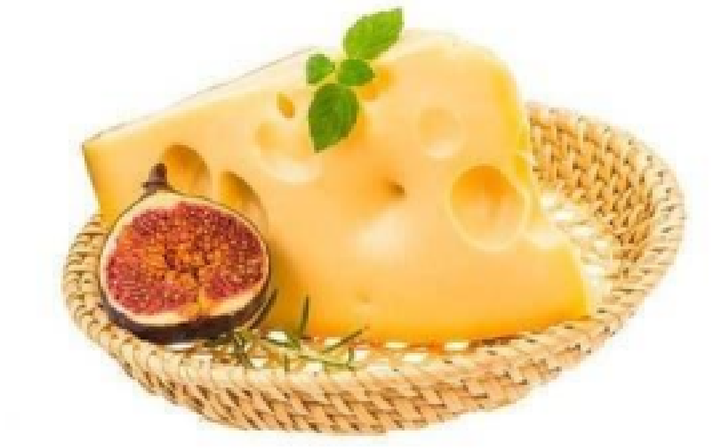


Figure 8: Original/Input Image

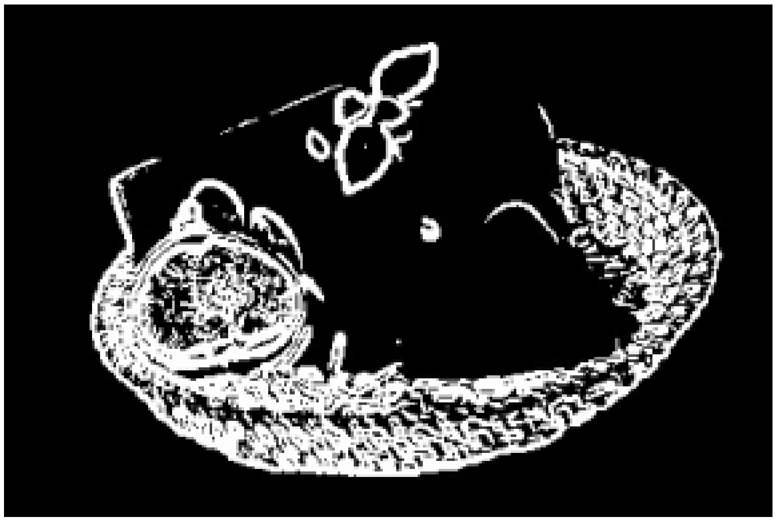


Figure 9: Binary/Output Image

The next process was finding a machine learning model to toss the dataset into. After quite a bit of research, our group determined that the specific problem of machine learning was image segmentation which is the process of separating a digital image into multiple parts. The most common architecture used is called UNet[4]. This architecture is a As seen in the image below, it gets its name from its U shape and is typically used on medical images to detect tumors.

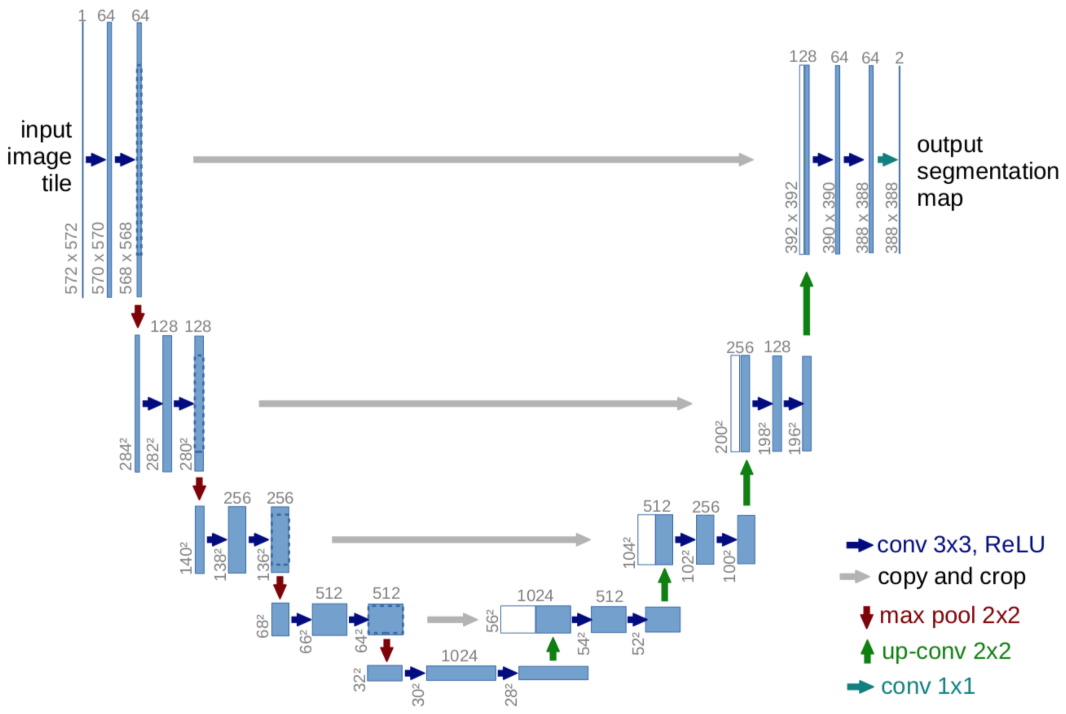


Figure 10: UNet Architecture[4]

However, in our case, it would detect areas of high gradient. It is different from normal architectures because it includes downsampling and upsampling, allowing the model to not only figure out what features are there but also where it is located. Normal models typically only use downsampling because a very common machine learning problem is image classification, which only requires knowledge of what the image is, and consequently, only downsampling.

The architecture for UNet was found on GitHub, created by aladdinpersson[5]. Our group chose PyTorch over Tensorflow as Eva and Dr. Cullinan's work is currently in PyTorch, an optimized tensor library that is mainly used in deep machine learning models that use either CPUs or GPUs, allowing for easier integration if it is used. Despite being premade, our group still ran into quite a few problems while trying to implement the model.

Results

Problem 1:

The model was training on the computer's CPU as opposed to GPU. This resulted in far longer training speeds. Resolving this involved a lot of trial and error with installing Nvidia CUDA drivers.

Problem 2:

The predictions the model was outputting were largely fully white images. This issue was a result of us switching the binary values. In other words, instead of having a black canvas with white mask values, our group had a white canvas with black mask values, causing the model to predict a fully white canvas.

These were the two largest issues besides small debugging to ensure the image path reading and writing were correct.

The model was run with a batch size of 1 and 5 epochs. After each iteration, it saves the model as a .pth.tar file

Below are the results from some of the validation sets.

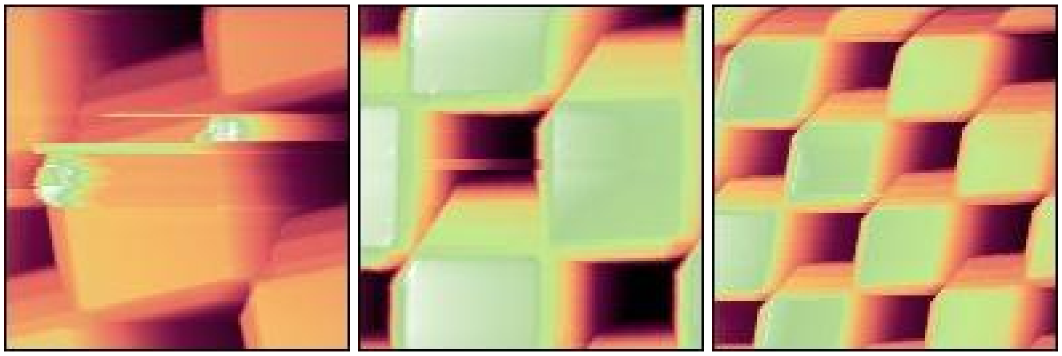


Figure 11: Original AFM Images

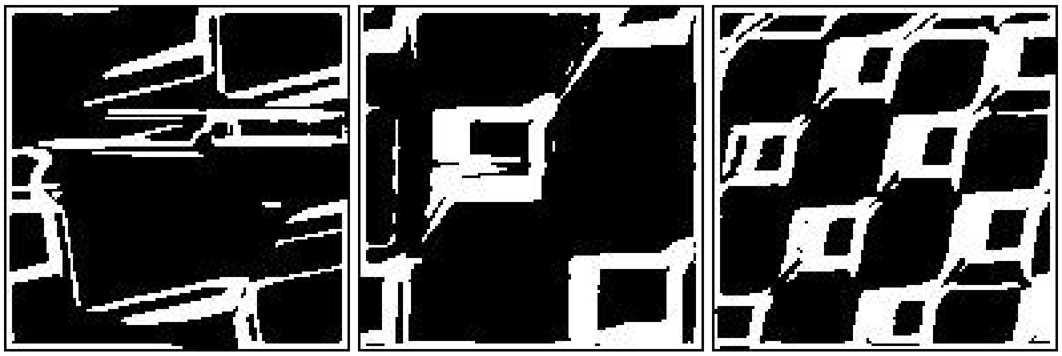


Figure 12: Algorithm Generated Masks

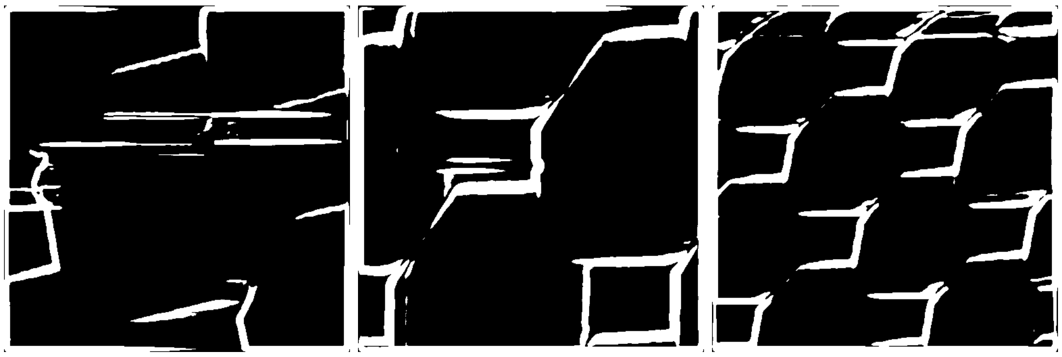


Figure 13: Predicted Masks

Predicted masks are outputs that are obtained after submitting the data set/original AFM images through our neural network. The images referred to as algorithm generated masks are how our output images/predicted masks are supposed to ideally look. These algorithm generated masks are obtained from submitting the data set/original through the edge detection algorithm. When the predictions are created, they are also tested for accuracy by determining the percentage of pixels correctly predicted by the model. This accuracy is tested through the comparison of pixel values from the algorithm generated masks and the predicted masks done by our neural network model. After our neural network model compares the output image/predicted mask it generated to the algorithm generated mask and learns from it is able to learn from its mistakes as it is able to view which pixels were not detected correctly. This allows for the neural network to learn after validation and apply that new knowledge to the next time this process is repeated. The average pixel percentage accuracy among the validation set was ~80%.

Conclusion

While the model is evidently imperfect, this percentage can be raised by increasing the epochs or increasing the dataset. Our group's process commenced by choosing one of the algorithms to form the dataset for the machine learning aspect of our assignment. Afterwards, we determined that we would implement the most common architecture, UNet, in order to both locate and identify the features of our images (which was found on Github). After debugging for errors that arose from incorrect implementation, we yielded our predictions that had a net accuracy of ~80% simply by comparing pixels from both the generated and predicted masks. Despite these imperfections, the purpose of this project was largely to serve as a proof of concept for creating a machine learning model to detect areas of the high gradient; the outcome was pretty successful in accomplishing that. Overall, our model works as a foundation for Dr. Cullinan and Eva's work on high gradient area detection and can be ameliorated further over time. The results are uploaded to GitHub[6].

References

[1] https://github.com/opencv

[2] https://github.com/numpy/numpy

[3] https://drive.google.com/drive/folders/1slHtSwUTrUPhkwh4MwEjK-W-jR-Vbian?usp=
sharing

[4] https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

[5] https://github.com/aladdinpersson/Machine-LearningCollection/tree/master/ML/Pytorch/
image_segmentation/semantic_segmentation_unet

[6] https://github.com/darrenau03/FIRE-Machine-Learning