Final Project Report for CS 184A/284A, Fall 2021

Project Title: Cell Instance Segmentation

Project Number: 15

Student Name(s)

Shu Nie, 94511877, nies1@uci.edu

Yuxuan Chen, 13272990, yuxuac9@uci.edu

Robert Armen Missirian, 1571355, missirir@uci.edu

Github: https://github.com/zjsscyx222/CS184

1. Introduction and Problem Statement (1 or 2 paragraphs)

Neurological disorders are a leading cause of death and disability across the globe. One accessible approach to detect response to the treatment is reviewing neuronal cells via light microscopy. However, segmenting individual neuronal cells manually can be both challenging and time-intensive.

Our objective is to develop a deep learning model to instance segment human neuronal cells with high accuracy on biological microscopy images. In this project, a Mask Region Based Convolutional Neural Network (Mask R-CNN) is introduced to distinguish and segment different kinds of neuronal cells using the LIVECell dataset provided by Sartorius on Kaggle. The model classifies individual neuronal cells, obtains boundary boxes and generates a separated mask for each of them. We reimplemented the Starter Torch Mask R-CNN code public available on Kaggle. An accuracy of 27.3% Intersection over Union (IoU) was reached in testing. The pixel ranges of a single neuronal cell can be recognized automatically and printed out as output. However, compared to the ground truth, there is still a large room for the accuracy to be improved.

2. Related Work: (1 or 2 paragraphs)

Based on this survey paper on image segmentation, the regional convolutional network (R-CNN) based models(Fast R-CNN, Faster R-CNN, Masked-RCNN) have proven successful in instance segmentation. One extension of this model, the Faster R-CNN employs a region proposal network (RPN) to propose bounding boxes candidates as a Region of Interest(RoI). The RoIPool layer calculates features inside boxes to predict the class of object and bounding box coordinates.

He et al. developed a Mask R-CNN model, which is basically a Faster R-CNN with 3 output branches: the bounding box coordinates, the associated classes, and the binary segmented masks. This method is simple to train and perform well in Usiigaci: Instance-aware cell tracking in stain-free phase contrast microscopy enabled by machine learning. In this project, we need to detect and classify the neuronal cells and generate a separate mask for each cell, so the Mask R-CNN model can satisfy all these requirements.

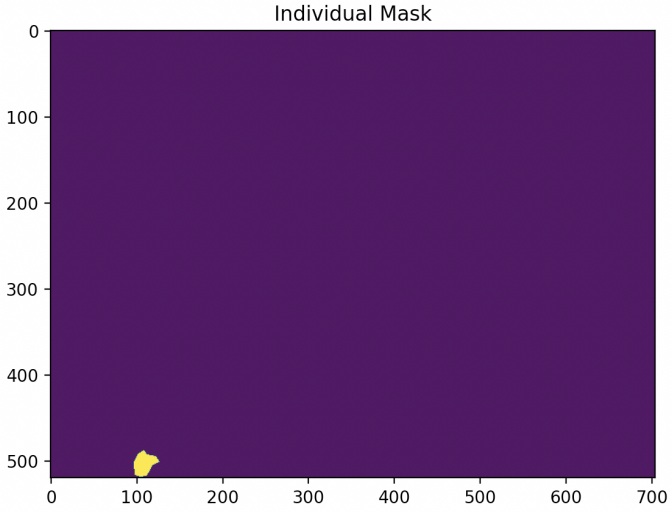
3. Data Sets [at least 1 page]

This project will use the Mask RCNN model to distinguish neuronal cells in biological microscopy images using the kaggle LIVECell_dataset_2021 dataset, which includes images and corresponding annotations.

At its core, the dataset consists of a series of PNG files that show cells under a microscope. These images are 520 x 704. The objective is to create annotations, called targets, that effectively describe the cells shown in the PNG images. The dataset is divided into a training set, containing 3 images and a test set, containing 303 images. The random flip function in the data augmentation class multiplies it so there are 606.

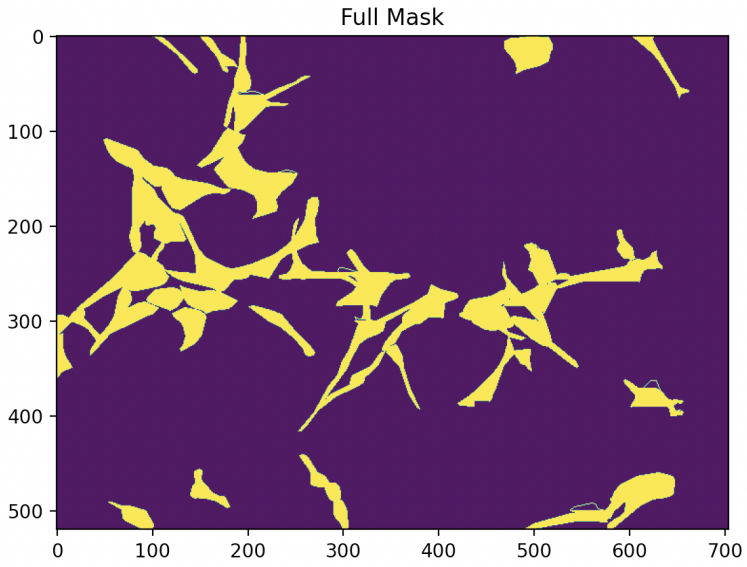
The provided annotations, also known as targets in the code, consist of masks, boxes, labels, image id's, and a boolean called iscrowd which is always 0. They describe the location and nature of the detected neuronal cells. There is one annotation per image, identified in the output table by its corresponding image id. It is these targets, specifically masks, boxes, and labels, that are also emulated by the MASKRCNN_RESNET50_FPN pretrained network.

The most important outputs are masks, which are 520 x 704 maps that overlay the images to distinguish where the neuronal cells are. Each mask carries markings in a box. Boxes are 4d vectors [min_x, min_y, max_x, max_y] that bound a detected cell. Each box represents a certain type of cell, which is classified by using labels which are vectors whose dimension of the total number of boxes. Each box also has an area measurement corresponding to the size of the feature it encloses.



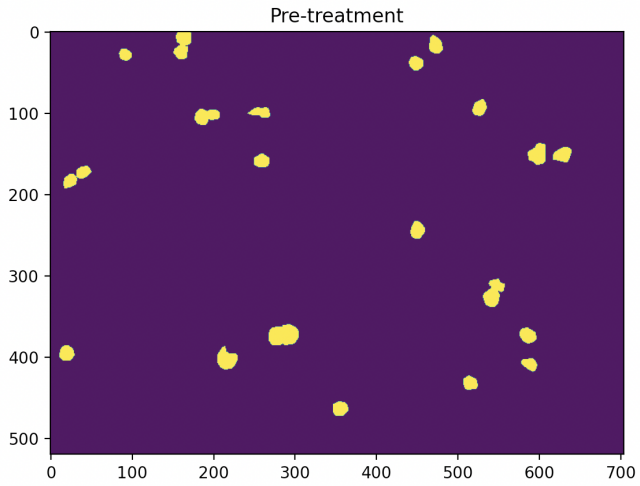
An individual mask shown above.

There is one box per mask, and there are many mask-box pairs per image which each display different parts of the pattern. Since there is an array of boxes and an array of masks, the arrays can be combined using logical or to show the larger pattern. When the masks are put together, this creates the full mask, a large group of green blobs that cover parts of the image, which is ideally the parts where the cells are.



A full mask shown above.

Some masks are from different parts of the brain, for instance this one is from a healthy hippocampus, whereas this one below is from a patient with pre-treatment Motor Neuron Disease.



A pre-treatment mask shown above.

This dataset is part of the larger LIVECell dataset, which is publically available on GitHub

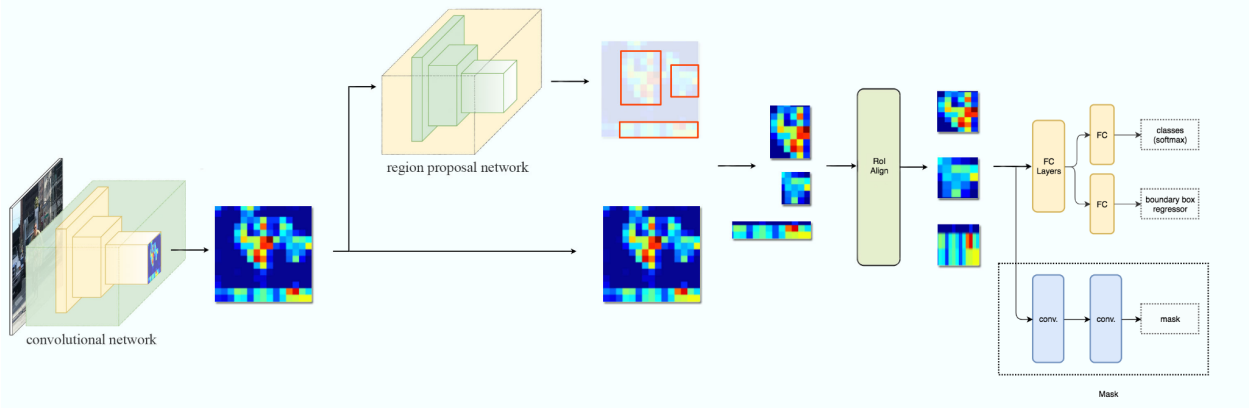
here.

It's worth mentioning that the train.csv file is a central part of the dataset because it contains encoded annotations, which are critical to the formation of all previously mentioned target values. Before being readable as targets, the annotations are in unreadable form in the csv file under the Annotations column. They are decoded from this form in the function rle_decode.

4. Description of Technical Approach [at least 1 page]

The main system we used for this project is Mask R-CNN. It has similar process as the Fast R-CNN method and can be briefly summarized into 3 steps like the pipeline in figure 1:

1. Pass the image through selective search and generate region proposals(about 1K-2K).
2. Calculate IoU (intersection over union, IoU = Area of the intersection / Area of the union) on proposed regions with ground truth data and add labels(TP/FP/FN) to the proposed regions.
3. Pass the test image to selective search and then predict the regions' class. Then refines the bounding box and generates a mask at the pixel level of the object based on the first stage proposal



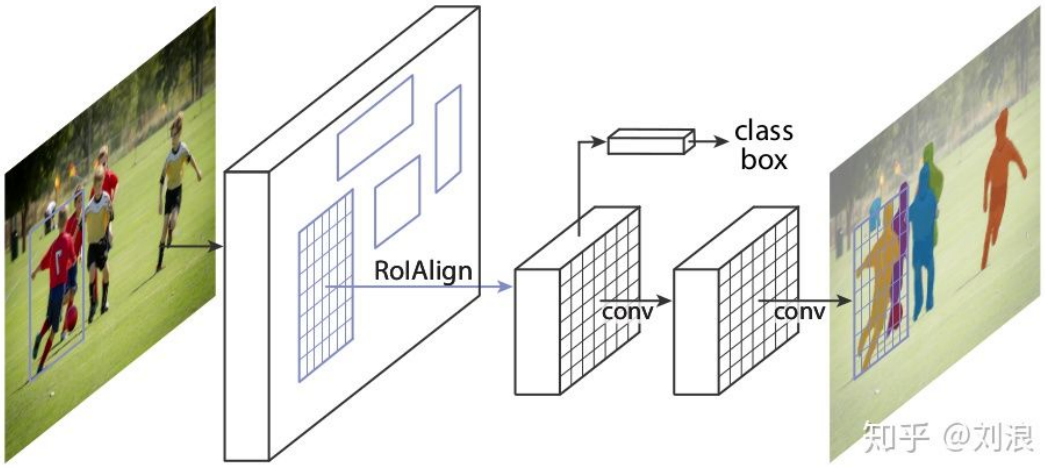
(figure 1: R-CNN process pipeline[1])

The method of Mask R-CNN extends the Faster R-CNN by adding a branch for predicting the target Mask on each RoI. [2]This branch is executed in parallel with the existing branch for bounding box recognition, as shown in figure 2:

---

[1] Jonathan Hui, Image segmentation with Mask R-CNN

[2] He, Kaiming et al. "Mask R-CNN." 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 2980-2988.



(figure 2: Mask R-CNN framework used for target segmentation)

Since the mask branch does segmentation mask predicted pixel to pixel on each RoI, which can ensure more accuracy than Fast R-CNN method, it can be the best choice for image recognition processing. Even though they have similar steps, the mask directly separated by Fast R-CNN is misaligned since its core operation can only perform rough spatial quantization feature extraction.

Both RoIPooling and RoIAlign are standard operations for extracting feature maps from each RoI. Compared to RoIPooling which needs to use a quantization algorithm twice, RoIAlign is a layer independent of quantization and uses linear interpolation algorithm instead, which does not digitalize the boundary of the cells and make every target cell to have the same size. It also has the interpolation algorithm to calculate the feature map values within the cell better. Since Mask R-CNN with RoIAlign improves mask accuracy by 10% to 50%. So we can conclude that Mask R-CNN has significantly more accuracy than Fast R-CNN.

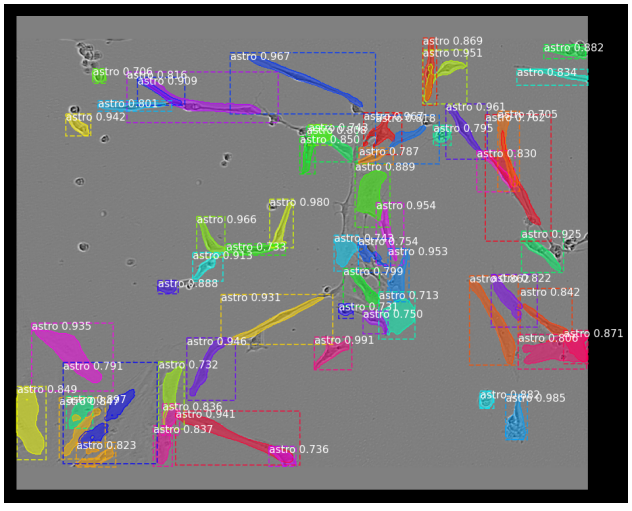


Figure 3 is the visualization of Mask R-CNN. The dotted rectangle areas are the RoI proposals. And the small text is the type of cell it predicts. The final classification should be solid lines instead of dashed and called “boundary box refinements.”

(figure 3: Visualization of Mask R-CNN)[3]

[3] Mark Wijkhuizen, Sartorius MASK-RCNN EfficientNetV2 Train < 2H

Also, to eliminate the overlapping boxes, we apply the method called "*Non Maximum Suppression* (NMS)." It is a technique that filters the duplicate predictions except the most confident one.

For code implementation, we used a kaggle code[4] for the main reference and did some changes and optimizations. Since the raw data amount is small, we did data augmentation with DegreeFlip, scaling, and AWS. We optimized the GPU algorithm and let it cost less memory to run as many batches as it could. But even though it did work, the total memory of the GPU of our best device cannot afford this amount of data and we have to transfer it back to the CPU. To help in Evaluation, we also added a csv writer within the training loop to collect training loss during each batch in each epoch and plotted all of them.

5. Software [at least ½ a page]

We mainly finished most of the coding part on Pycharm. It is a programming integrated development environment (IDE) that focuses on Python. PyCharm is a code analyzer, graphical debugger, integration tester, integrated version control system, and support for Django web development tool created by Czech firm JetBrains(Wikipedia.)

Codes and python files we created were listed in the README.md file in our GitHub repository (zjsscyx222/CS184) with detailed introductions.

The libraries and third-party softwares we used in our project were listed as follows:

Library/Package	Introduction
Matplotlib	A comprehensive library for data visualization and graphical plotting in Python
Numpy	The basic Python package for scientific computing.
Sklearn	The tool for predictive data analysis.

[4] dataista0 (Julián Peller), Sartorius - Starter Torch Mask R-CNN [LB=0.273]

Pytorch	An open source machine learning framework
Torchvision	A library for Computer Vision that goes hand in hand with PyTorch. (But it needs to be installed separately from pytorch.) It has utilities for efficient Image transformations,which we used in our project here.

6. Experiments and Evaluation [at least 1 page, preferably 2 or 3]

The accuracy metric for the project is the mean average precision at different Intersections over Union (IoU) threshold. i.e. the area of intersection divided by the area of union between the predicted segmentation map and the ground truth:

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The ground truth is denoted by A, and the predicted map is denoted by B.

The metric sweeps over a range of IoU thresholds, at each point calculating an average precision value. The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). In other words, at a threshold of 0.5, a predicted object is considered a "hit" if its intersection over union with a ground truth object is greater than 0.5.

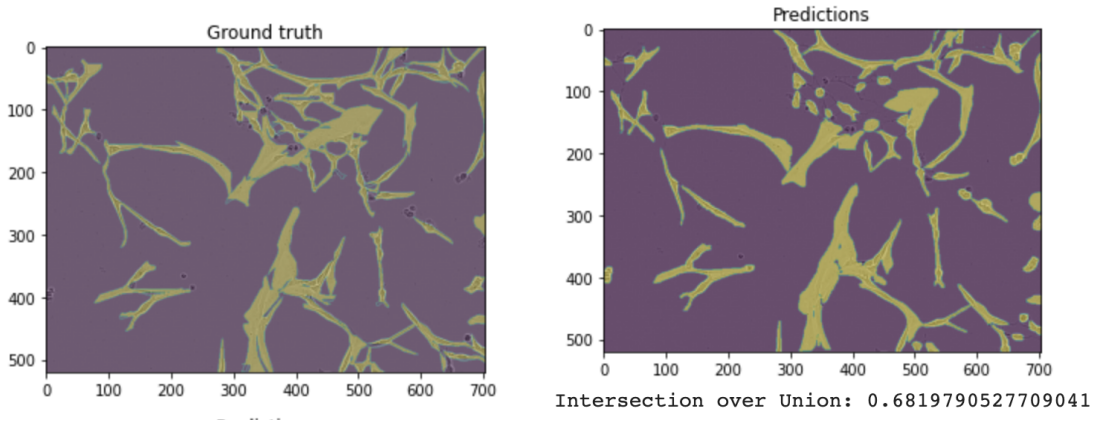
At each threshold value t, a precision value is calculated based on the number of true positives (TP), false negatives (FN), and false positives (FP) resulting from comparing the predicted object to all ground-truth objects:

$$\frac{TP(t)}{TP(t)+FP(t)+FN(t)}.$$

A true positive is counted when a single predicted object matches a ground truth object with an IoU above the threshold. A false positive indicates a predicted object had no associated ground truth object. A false negative indicates a ground truth object had no associated predicted object. The average precision of a single image is then calculated as the mean of the above precision values at each IoU threshold:

$$\frac{1}{|thresholds|}\sum_{t}\frac{TP(t)}{TP(t)+FP(t)+FN(t)}$$

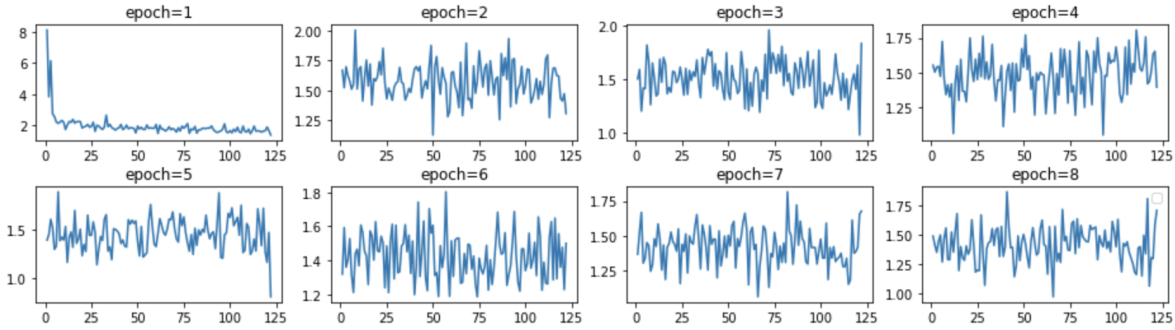
Lastly, the score returned by the competition metric is the mean taken over the individual average precisions of each image in the test dataset.



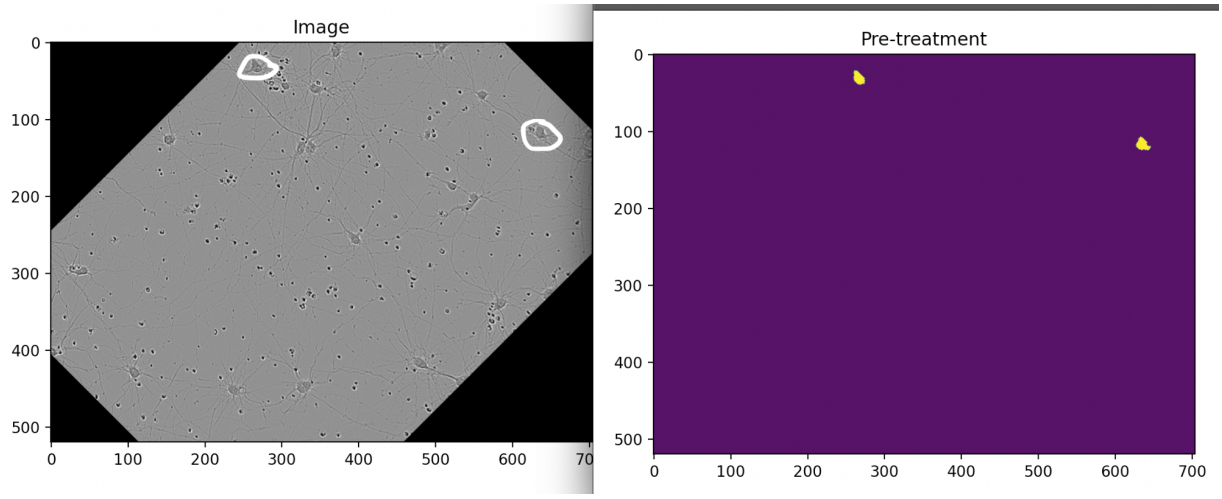
The Jupyter notebook ends with an example calculation of intersection over union algorithm on a sample image to show the accuracy of the model. In order to demonstrate that a model is better than the **Mask-RCNN** benchmark, IoU=0.273, we would need to calculate IoU for every image in a test set and average them together to get a greater value.

Mask-RCNN Training Loss

During the training of the Mask RCNN, the batch training loss function dropped considerably during the first iteration, moderately during the second iteration, and then fluctuated thereafter between 1 and 2, centered at 1.5.

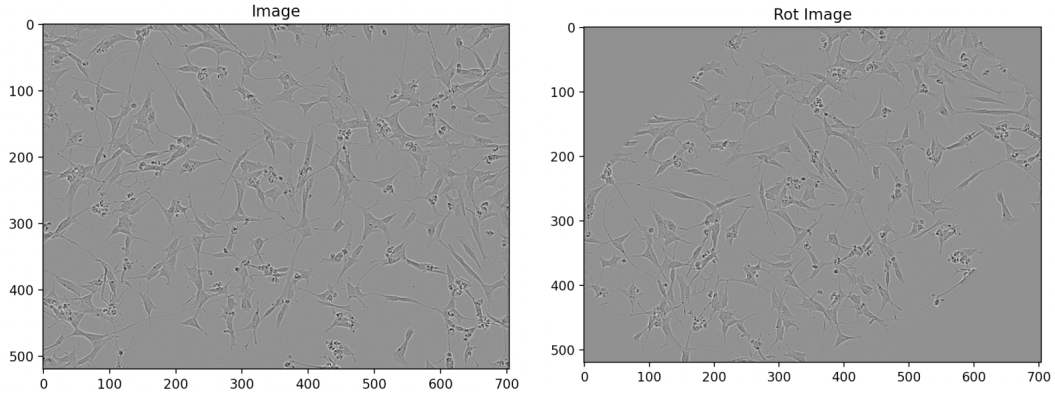


The data suggests that after the second iteration, training didn't significantly lower the loss, suggesting that the performance of the model after the second epoch may be similar to the performance of subsequent iterations. Furthermore, the loss variance appears to be abnormally high, suggesting that the learning rate has stayed too high for this point in the training. This lines up with the fact that the algorithm uses SGD, which tends to take a long time to converge, leading to a slower decline in learning rate. Thus, it is possible that this graph may be more smoothly convergent if the experiment was repeated with an optimizer that converges more quickly by virtue of a fast decline in learning rate.



Data Augmentation

A critical problem in biology arises when training models with small datasets. It is very difficult to capture the important patterns without overfitting to the particular samples available. If the image could be rotated an unlimited number of degrees around, the problem would be alleviated at least for visual data. This is an example of an attempt to implement this generalized rotation scheme. We can see that the mask and the image have been successfully rotated in tandem. However, black boxes have appeared on each corner, leading to erroneous input. This could be fixed by zooming in to avoid the outside shadow, but that would make the picture smaller, which makes it incompatible with the rest of the data. It could be fixed by selectively filling in the black with the background color of the microscope, but this still creates an unnatural diagonal as well as damages any black pixels in the center of the image, as seen below. Nonetheless, it is an improvement.



Code for this is also available *here*.

The main obstacle to implementing generalized rotation in practice is the batch function gets an error much later that the targets have no function ".to()".

7. Discussion and Conclusion [at least ½ a page]

We found a number of interesting properties and idiosyncrasies of R-CNNs, as well as of the dataset itself. The dataset and the choice of network are inexorably related, and in this case a major problem with R-CNN's is their ability to memorize a small dataset. We were nearly able to address this problem using custom degree flipping, but implementation issues remain. The R-CNN seems tailor made for the dataset, given that the training data already contains multiple masks, suggesting that the authors may have intended this approach to be used. As far as overfitting, the loss drop in epoch 1 suggests that good learning is occuring, but later stagnation suggests that overfitting may have started. Nonetheless, there appears to be no accuracy cost to allowing all 8 epochs to run their course, rather it is only a time cost.

In this project, we discovered numerous hurdles that make achieving high accuracy on this data very difficult. Firstly, the image resolution is quite high, making it difficult to run it through a deep learning model at all on a normal computer. This is also due to the dual nature of the pretrained model where it runs an initial CNN, plus a connected layer for every bounded box. It took 9 hours to run on Google Colab, although it took 25 minutes on an Amazon g4 server, which uses a fast P4 graphics card. However, using Amazon servers can get quite expensive, especially as demand for graphics cards rises.

Secondly, the number of images is quite low, making it difficult to encapsulate the full pattern relating the microscope visuals to the masks. Compared to a standard 60000-sample mnist dataset, it is difficult to avoid overfitting and dataset memorization, something the CNN's are well known for. In this respect, the CNN is not optimal for this project, although it's visual properties, specifically with respect to kernels, are still formidable. A natural inclination is to augment the data by zooming in, rotating, and otherwise manipulating it in some combination, but this has thus far proved operationally difficult as seen in section 6 concerning DegreeFlip. Nonetheless, there is still possibility for further improvements in this respect. In addition to improvement from data augmentation, there is also the possibility of improvement in accuracy from a smoother convergence in loss. Because the loss function was so chaotic, a smoother optimizer such as Adam may create better conditions for optimal accuracy.

Thirdly, as the complexity of a machine learning model grows, the number of dependencies in the chain of logic that can break as a result of seemingly unrelated modification seems to be exponential. As an example, after substituting between two rotate functions acting on the tensor target['masks'], DegreeFlip returned tensors of the correct type and shape, yet lacking a ".to" function which is required in another file, leading to an error. The source of the error, the batch algorithm, contains complexities such as one line for loops and could be simplified to a more transparent flow of logic. A further example of this is in the way that annotations are encoded on the train.csv file. Instead of a JSON object or other familiar notation, they are described in an unfamiliar "rle" encoding that reads "112290 15 112993 18 113696 21…." If there were ever a bug in this section, it would be very complex to fix. Usually, a "for file in directory statement" suffices to bring in data from a folder rather than the complex csv scheme, so it would be strategic to simplify in the future.

As far as future direction, there are three main areas that we can look to, augmentation, reformulation, and combination. A highly sophisticated rotation algorithm that zooms in to compensate for visible edges and adds enhanced pixels to stay the same size could enable the neural network to get a better view of the data from different perspectives, overcoming the small sample size problem in the future.

Further, it would be helpful to reformulate the code into a more user-friendly design in the future. It is plain that the preprocessing section is rife with tangled webs of function calls, making it much more difficult to figure out the basics. Now that we have a better idea of the logic flow, it could make it easier to improve targeted segments of the code without having to worry about unnecessary complexity elsewhere.

Lastly, we can implement a combination of the R-CNN with other models, such as a Variational Auto Encoder (VAE) or UNET. While R-CNN is very useful in capturing features, it has a tendency to complexify and overfit, potentially missing the larger picture. By the 8th epoch of training, it could be argued that the network was overfitted, and the 2-dimensional latent space of a VAE could help simplify the model, while UNET would harness the power of CNNs from a different method of computation to gain perspective.

8. A separate page on *Individual Contributions*

Shu Nie: Supported in modifying code. Refined and finished the jupyter notebook and readme file. Added visualization for training loss. Wrote introduction, related work and part of evaluation.

Yuxuan Chen: Implemented the whole code(mainly modified the training loop part), optimized the algorithm to solve the GPU memory problem(but still not big enough), created csv writer to collect loss data during training loop, imported trained model for analysis part. Finished the 1st edition of the jupyter notebook file and the readme file. Wrote the final report section 4 and 5.

Robert Armen Missirian: Augmented the data in different ways with DegreeFlip, scaling, and did AWS; found a method that still makes data look real when rotated. Wrote Data Set, Discussion and Conclusion, and part of final report section 6, assisting in 2. Coded IoU algorithm into the analyze train sample function at the end.