CSE 5243
Final Project

Kevin Shen and Kai Wen Liu

Submitted on Apr. 18, 2018

1. Background and Significance

This project is inspired by the competition, 2018 Data Science Bowl on Kaggle. The goal is to create a machine learning model to automate nucleus detection. Identifying nuclei allows researchers to identify each individual cell in a sample, and by measuring how cells react to various treatments, the underlying biological mechanisms can be understood. By automating such process, more efficient drug testing is possible, shortening normally a long time it takes for each new drug to come to market. We selected this specific domain as our final project because it gives us hands-on experience on a real-world/competition dataset as well as the opportunities to explore and learn graph mining techniques, which we both have no experience on.

2. Introduction

Images and masks (nuclei)

Data given in the competition are all biological images. For each image in the training test, individual masks are provided to train the model (each mask contains one nucleus), as shown in Figure 1. **The goal is to correctly predict the masks for the test set images.** Images of individual masks can be easily superimposed as they are binary data (black and white).

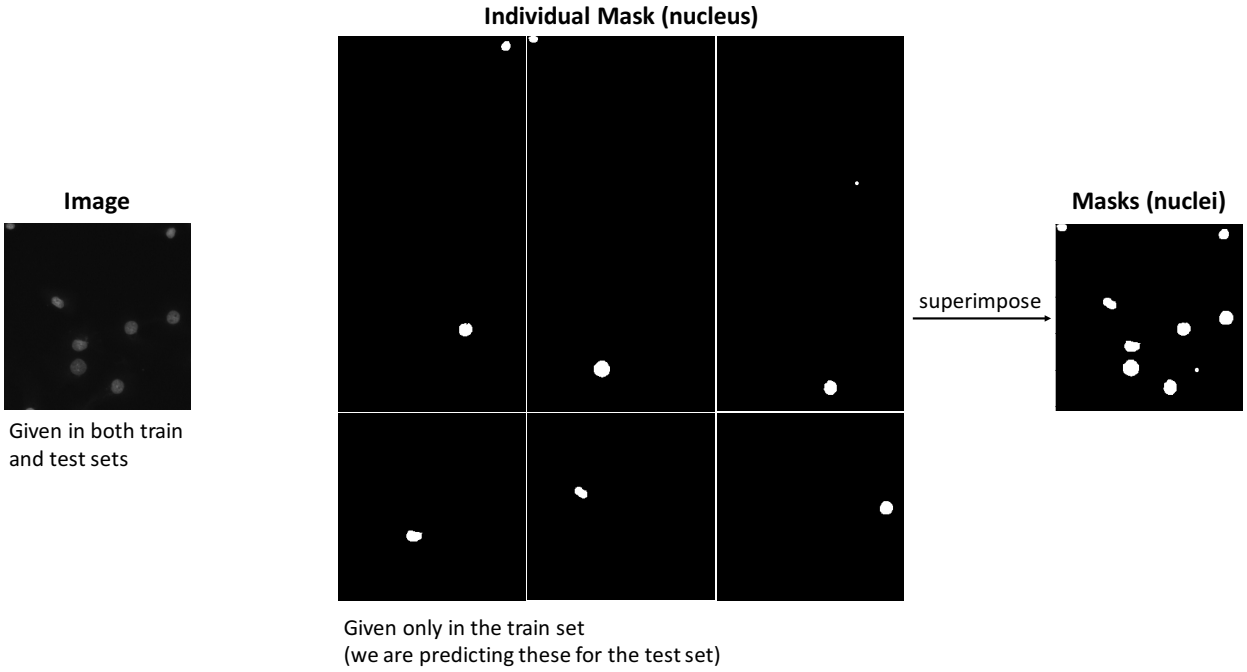


Figure 1. Schematic showing an example of image data given in the training and test set: (left) the image, (middle) individual masks provided in the training set, and (right) image of all the masks superimposed.

This dataset contains a large number of segmented nuclei images. The images were acquired under a variety of conditions and vary in the cell type, magnification, and imaging modality such as bright-field, histological and fluorescence, as shown in Figure 2. The main challenge is to create an algorithm with the ability to generalize across these variations.

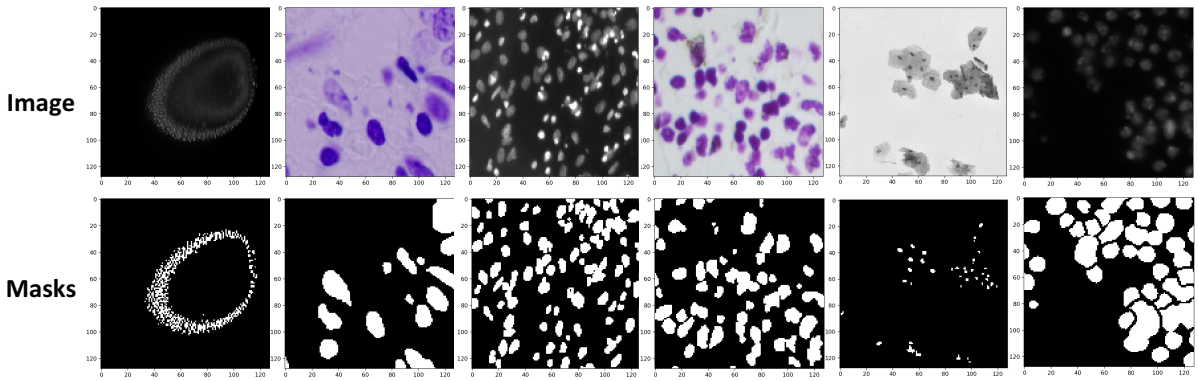


Figure 2. Examples of image data in the training set: (top) the images and (bottom) the superimposed masks showing that a variety of cell types, magnifications and image types is contained.

Prediction evaluation

Two important questions should be asked while evaluating the prediction: (1) whether the number of masks are correct, and (2) whether the predicted masks matches the true ones. For instance, it should be penalized when a predicted mask represents multiple adjacent true masks although they are quite overlapped. As a result, the predicted results are evaluated on the **mean average precision at different intersection over union (*IoU*) thresholds**. The *IoU* of a predicted set of object pixels (*A*) and a set of true object pixels (*B*) is calculated as the following equation:

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

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). 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 true objects:

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

TP is equal to the number of predictions that matches a true object with an *IoU* above the threshold. For example, at a threshold of 0.5, a predicted object is counted as a TP if the *IoU* is greater than 0.5. FP indicates the number of predictions that had no associated true object. On the other hand, FN indicates the number of true object that 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:

$$Average\ precision = \frac{1}{n_{threshold}} \sum_{t=1}^{n} precision(t)$$

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

3. Exploratory Data Analysis

Image and nuclei statistics

Table 1 shows the statistical descriptions of the image height, image width, and nuclei count of the training set. The image dimensions do not appear to be equally distributed and have quite large standard deviations. Figure 3 shows their probability distributions. The smallest image is 256x256 and the largest is 1040x1388 pixels. We could resize all the images to squares, however, it would cause images to be squashed either vertically or horizontally and would result in a loss of information. We will neglect this potential issue as most of the images are 256x256 (the squashing/stretching is not an issue for these).

Table 1. Image and nuclei statistics

	Image height	Image width	Nuclei count
Mean	333.991	378.500	44.972
Std	149.475	204.839	47.963
Min	256	256	2
25%	256	256	16
50%	256	320	28
75%	360	360	55
Max	1040	1388	376

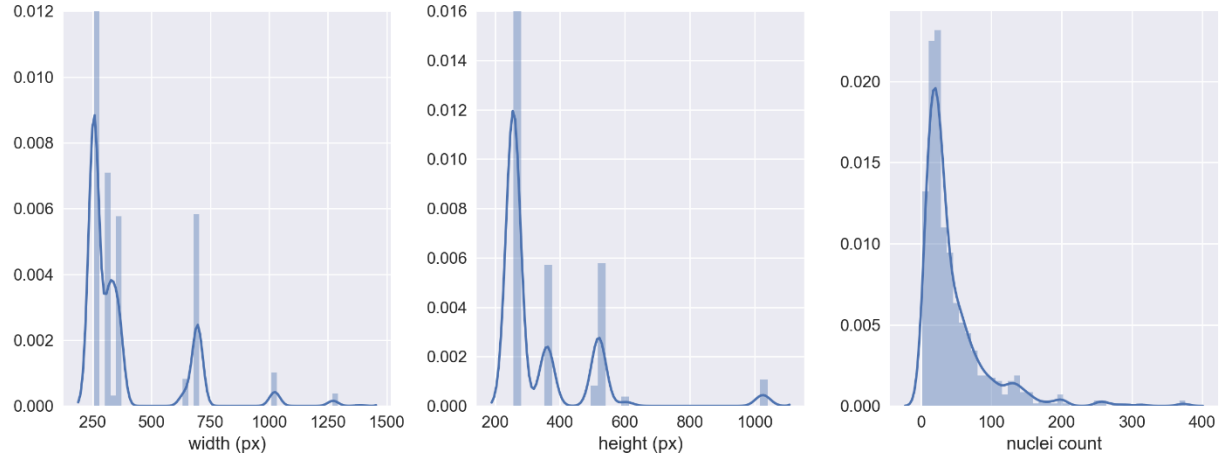


Figure 3. The probability distributions of (left) image width, (middle) image height and (right) nuclei count of the training set.

The count of nuclei also has a wide distribution, ranging from 2 to 376, and images that have the maximum and minimum amount of nucleus are shown in Figure 4. It is obvious that there are diverse types of images and varied sizes of nucleus; therefore, we also calculate the size of the nucleus (by calculating the number of white pixels). The smallest nucleus is only a few pixels in size, and the largest can be hundreds of pixels, as shown in Figure 5. The size of nuclei varies a lot throughout the images in the training set and is likely to make detection more challenging.

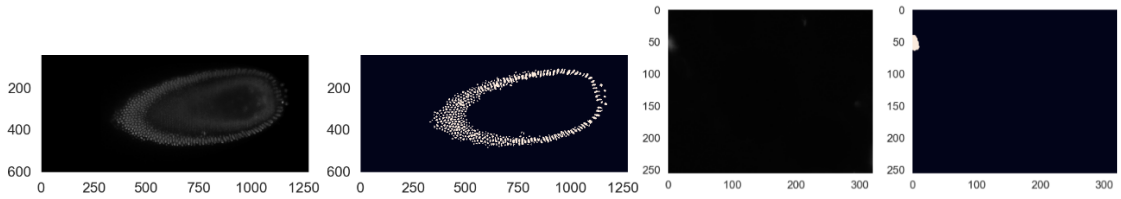


Figure 4. The images and masks (nucleus) of the data that has (left) the maximum and (right) the minimum amount of nucleus.

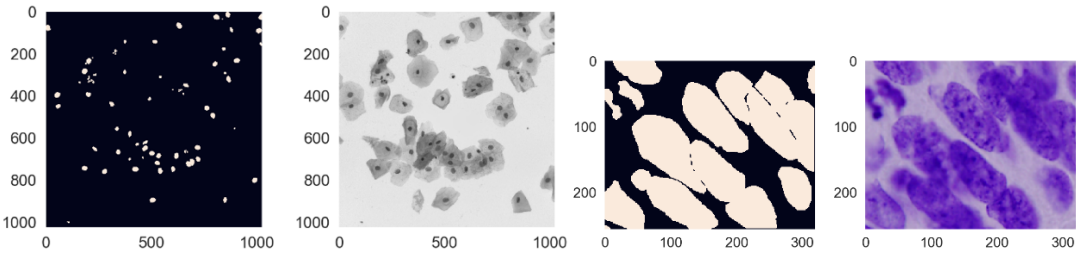


Figure 5. The images and masks (nucleus) of the data that has (left) the smallest and (right) the largest nucleus.

RGB intensity distribution

Since images are loaded in as Red-Green-Blue (RGB) matrices, we can look at the distribution of the RGB intensity, as shown in Figure 6. Gray indicates the mean of the total RGB values. Obviously, most of the image data locate on the diagonal, implying they are somewhere between black and white, whose RGB are (0,0,0) and (255,255,255), respectively. This is consistent with visual observation as a lot of images showed above are either with white nucleus and black background or the other way around. However, we did notice some clusters of data that are off-diagonal, as marked with red circles in Figure 6. Interestingly, some of them have certain red intensity; while some have a slope less or greater than 1. This indicates there are definitely varied types of images, such as fluorescence with strong features in RGB matrices that can be captured by clustering.

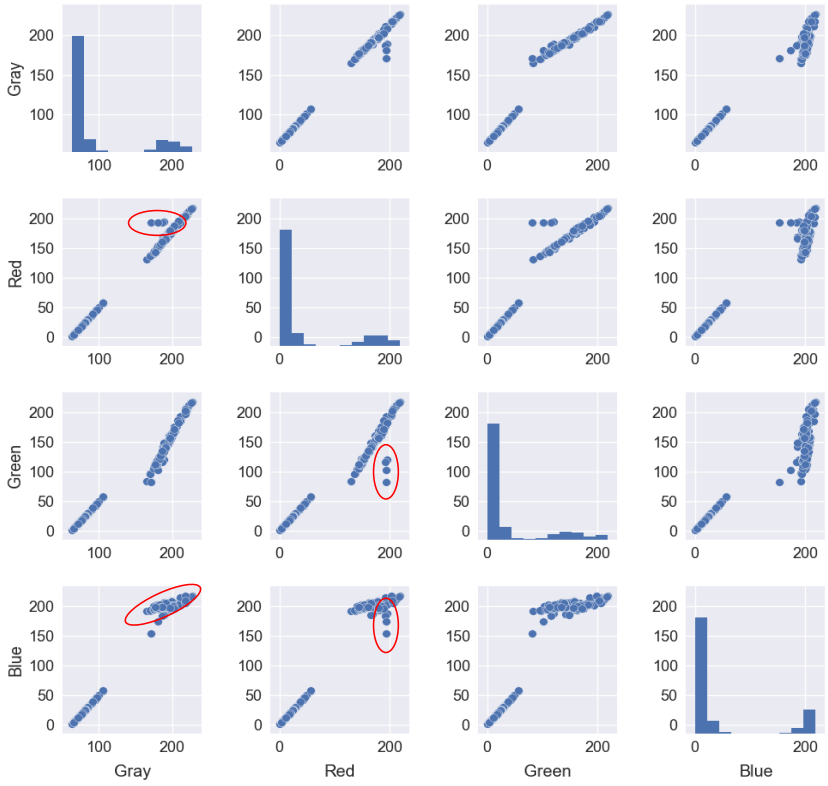


Figure 6. The distribution of the gray (average of the total RGB), red, green and blue intensities in the training set.

To better understand, we looked at the differences between RGB intensities as well as their standard deviations to focus on images that are not black and white. We found the most interesting results in the distribution of the difference between red and green intensities and its standard deviation (denoted as Red-Green and Red-Green-Sd, respectively), as shown in Figure 7. Both values would be non-zero for

colorful images. Multiple data clusters are revealed, showing that it could be a decent metric for data clustering that we aim to do (details are in the following section).

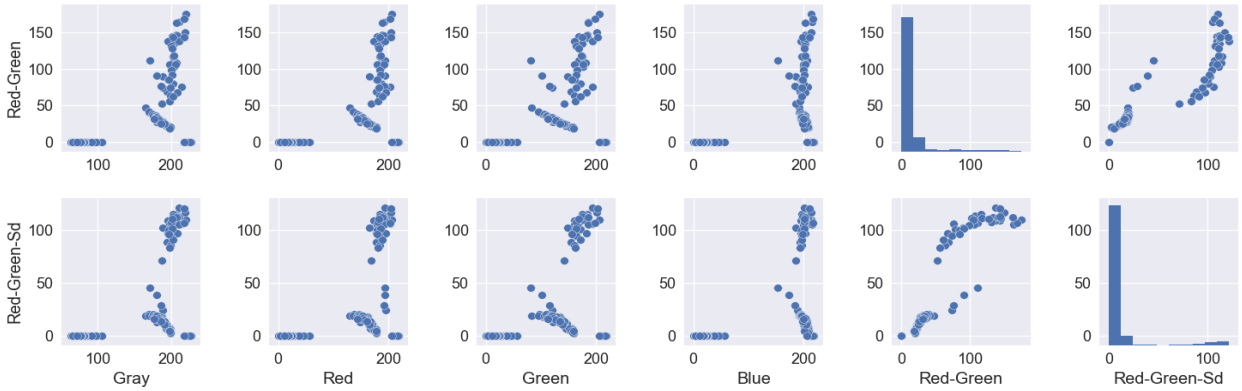


Figure 7. The intensity distribution of the difference between red and green intensities and their standard deviations.

4. Data Preprocessing: Clustering

Since there are multiple types of images, we aim to build individual model for each image type instead of having just one model trained with all the data. As a result, we did clustering for the training set and "predicted" which cluster each test data belongs to by finding the nearest centroid (from the training set). We ended up having 4 clusters in the training set, so we built 4 models in total (but having a model not used because are only three "predicted" clusters in the test set). Each test data is predicted by the model of the cluster that it belongs to. The results of all clusters were then combined and compared with that from a model trained with the raw data (not clustered). The clustering details are as follow:

Training set

As shown earlier, it seems promising to cluster the dataset by certain RGB intensities or their differences/standard deviation; therefore, we implemented the K-means algorithm by using the scikit-learn package with different values of k. Based on three variables (the green, difference between red and green and its standard deviation), we find that k=4 best clusters the training set, as shown in Figure 8. Sample images clearly show that we successfully distinguished various image type by simple K-means on RGB values. However, there seem to be more than 4 image types because (1) one can argue there are 5 clusters based on the relationship between Red-Green and Green and (2) the second image in cluster 1 is clearly different from the other two as it is more red. Nevertheless, we stick with k=4 as there are not enough samples in every cluster if we have k=5.

Test set

After getting the centroid location of each cluster from training set, we "predict" which cluster each test data belongs to by finding the nearest centroid. As there are only 65 test images, we found there are no cluster 3 images in the test set, as shown in Figure 9. Although cluster 0 in the training and test sets match well, cluster 1 and cluster 2 do not. Especially, the cluster 2 images in the test set look completely different from those in the training set. Also, based on their image sizes, we see that there are definitely some types of images in the dataset which are different between the training and test. However, since we did not look for external datasets to further train our models, we stick with the clustered results shown here.

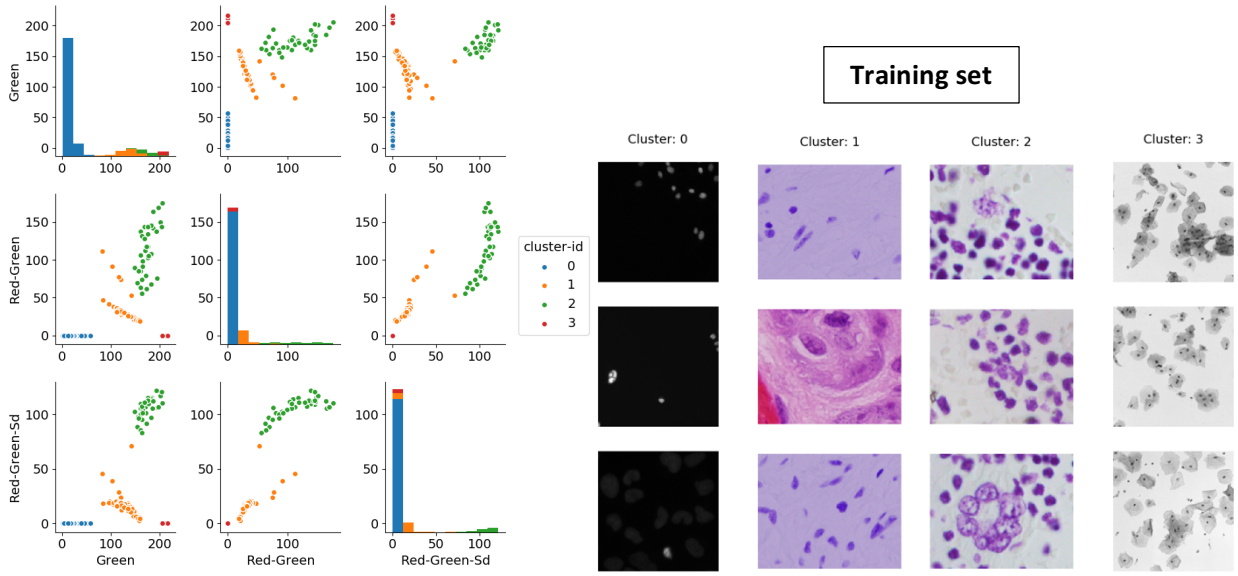


Figure 8. (left) The intensity distribution and (right) sample images of each cluster in the training set.

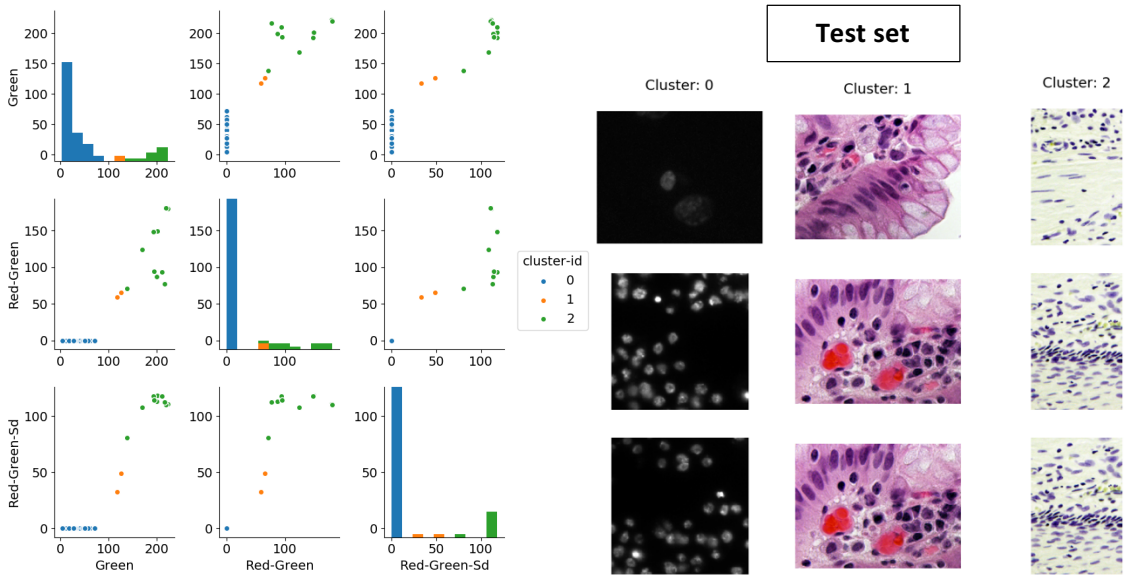


Figure 9. (left) The intensity distribution and (right) sample images of each cluster in the test set.

5. Model Building and Training

Data transformation

Before we start implementing algorithms to the images, we need to preprocess the images first. Every pixel in the image has its own RGB value (r, g, b) and it would be ambiguous for the computer to detect that the pixel should belong to one nucleus or the background. Therefore, we would normalize the RGB value to 0 or 1, which represent black (0, 0, 0) and white (255, 255, 255). As you can see, the processed images below would look like the image Y_train in Figure 10. In addition, the size of images is different. The width and length of images are in the range from 128 to 1024. To fit the model, we need to resize the images and set the size of images to 128*128. However, this might lead to information loss if the image is compressed.

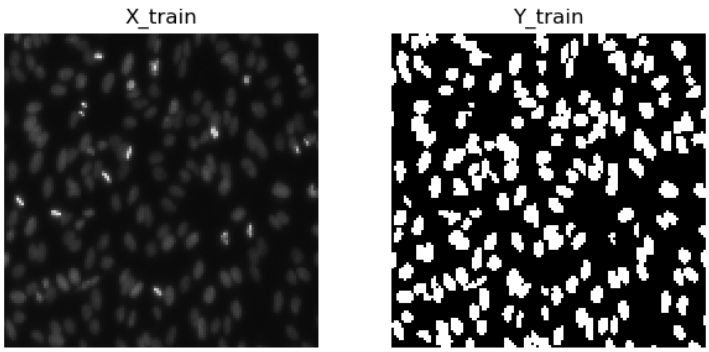


Figure 10. Resized images showed by pixels that represented with RGB matrices (X_train) and represented with True and False matrices (Y_train).

Simple Convolution Neural Network (CNN)

Firstly, we apply simple Convolution Neural Network (CNN) algorithm on this problem. The CNN algorithm performs the process called Convolution to select a small section of pixels (3*3 selected in this work) in the images as a feature and use the feature to see how many sections in the other image are similar to the feature. The following process is named Pooling. It means with the values of matrices we get from the Convolution process, we should select the maximum value in each matrix (the matrix should be a square matrix) and build new matrices as neural network layers. Before the algorithm runs on the next neural network layers, we could use the activation function like ReLU to make the new matrices has no negative value. To build the model, we tested several combinations of ReLu and sigmoid activation functions. However, the result turned out only implementing a sigmoid activation function at the last process would get a better accuracy. The other combinations of activation functions showed the accuracy dropped dramatically and the model loss increased.

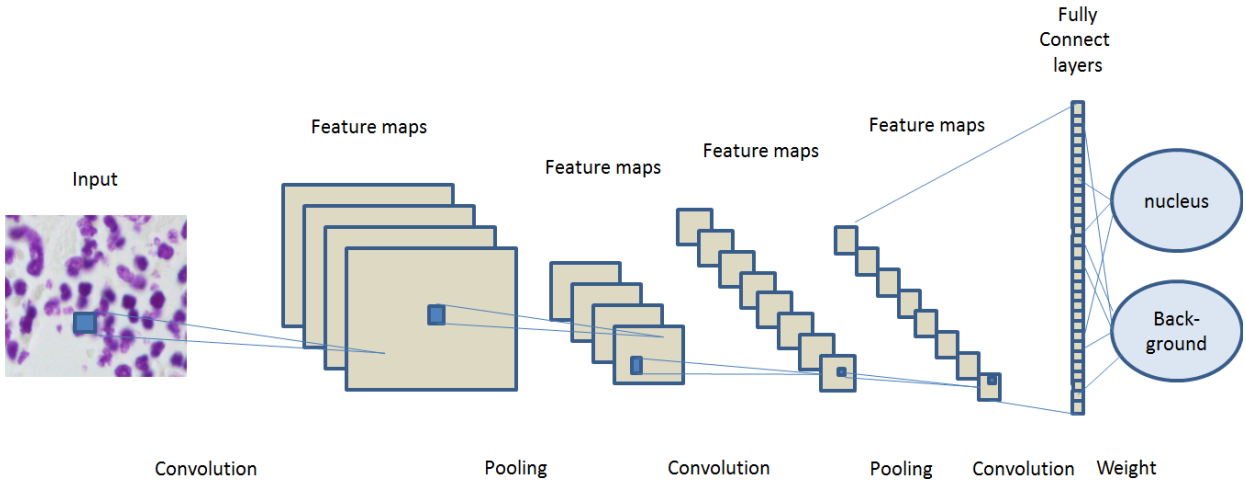


Figure 11. Structure of simple Convolution Neural Network.

U-Net Convolutional Networks

The U-Net model aims to deal with biomedical image segmentation problem. The algorithm structure is shown in Figure 12. There are four down-sampling and four up-sampling processes. Within these processes, we use the ReLU activation function to do the convolution.

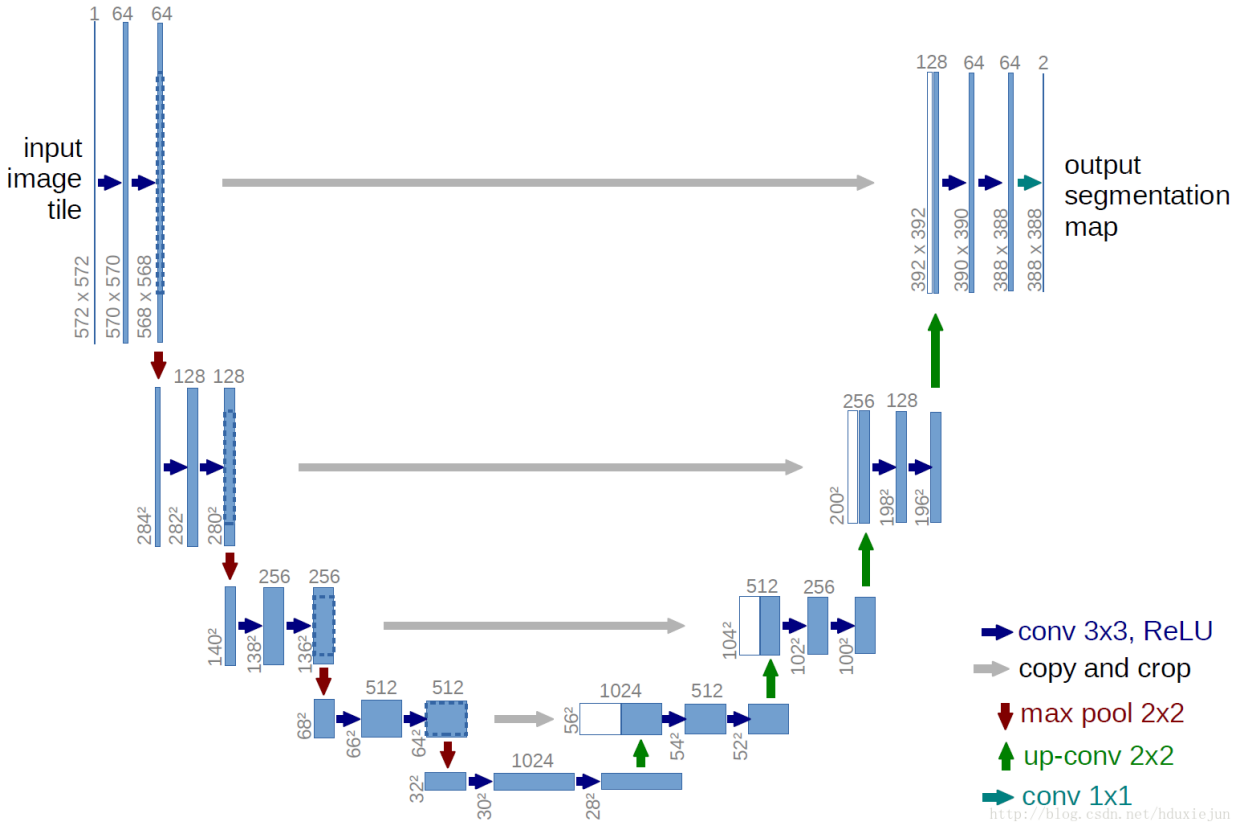


Figure 12. The structure of U-Net Convolutional Networks. The image is from ref 1.

6. Model Validation

Simple Convolution Neural Network (CNN)

We partitioned 20% of the training set as the test set to validate the model. When we implemented simple CNN algorithm on the data, the model accuracy would be around 0.6. If we keep training the model for 30 epochs, we could reach the accuracy about 0.9. Figure 13 shows the track of model accuracy and model loss in 30 epochs. The model loss is defined as:

$$\text{loss} = -\frac{2 * TP + smooth}{TP + FP + FN + smooth}$$

where $smooth = 1$ is added to keep the loss greater than zero. The model loss is related to the IoU value and it equals to the negative dice coefficient. We keep track of the model loss for 30 epochs. In Figure 4, we can see the curves of simple CNN of accuracy and loss end up well. In addition, we found that some of the image types of test set are different from the train set when we were doing clustering. As a result, the curves of test set are a bit unstable most of the time.

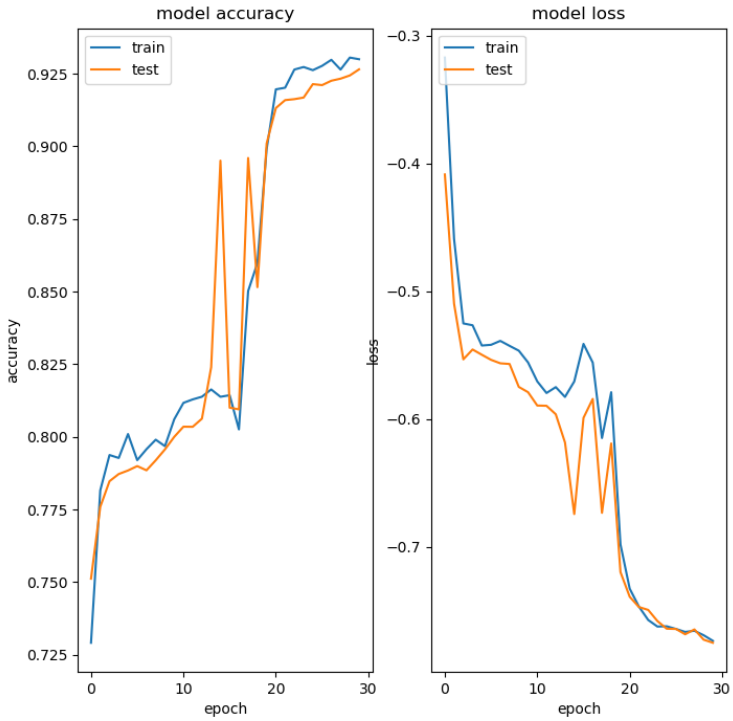


Figure 13. Model accuracy and loss of model applied simple CNN on train set and test set.

In Figure 14, prediction is the result of using simple CNN and put all the masks in the figure. However, we can see some small patches in the figure that should not be there. These patches would lead to the model loss. We can eliminate these kinds of error by setting a threshold for patches being recognized as nuclei. However, since the size of nuclei different from figure to figure, we decide not to set a threshold to filter the noise.

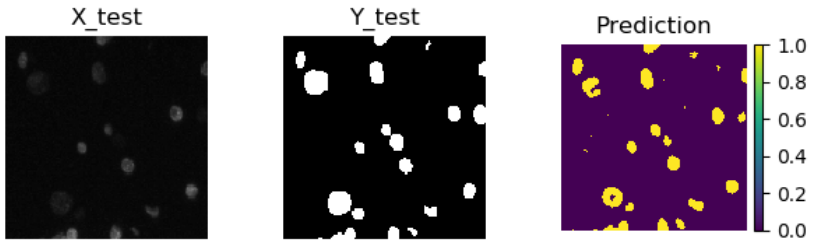


Figure 14. Original image (X_test), normalized image (Y_test), and Prediction of the image run by simple CNN (Prediction).

U-Net Convolutional Networks

When we applied U-net on the train set and test set, it turned out the accuracy was slightly higher than the result of implementing the simple CNN model and the model loss was also lower (as we can see in Figure 14). Figure 15 shows the prediction image of nuclei distribution and it has no obviously unrelated small patches or noise compared to the original image and the prediction of simple CNN.

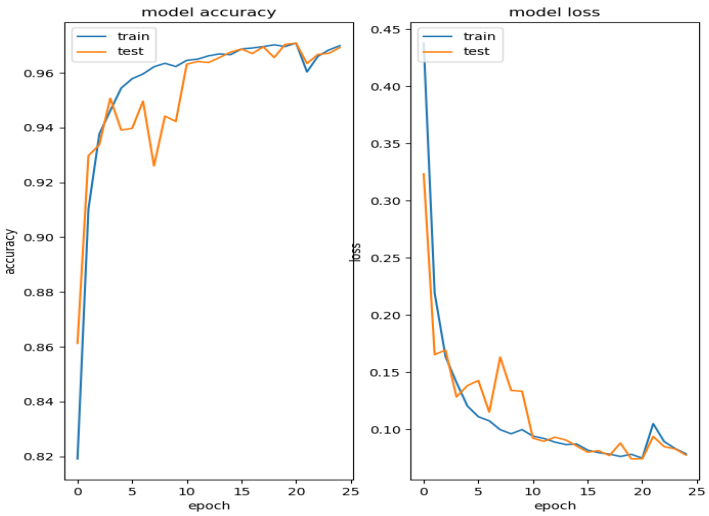


Figure 15. Model accuracy and loss of model applied U-Net on train set and test set.

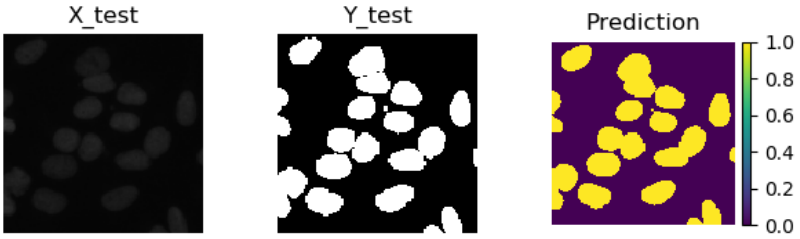


Figure 16. Original image (X_test), normalized image (Y_test), and Prediction of the image run by U-Net (Prediction).

Because we think the type of images would affect the result, we applied simple CNN and U-Net on the clustered train set and test set. Each cluster in test set should correspond to one cluster in the train set. However, there were only three clusters in the test set and four clusters in the train set. Therefore we must discard one cluster in train set which was not highly related to the test set. The result turned out both simple CNN and U-Net improved the accuracy by using the clustered dataset. The curve of model accuracy rose up quicker, higher, and even smoother in figure 17.

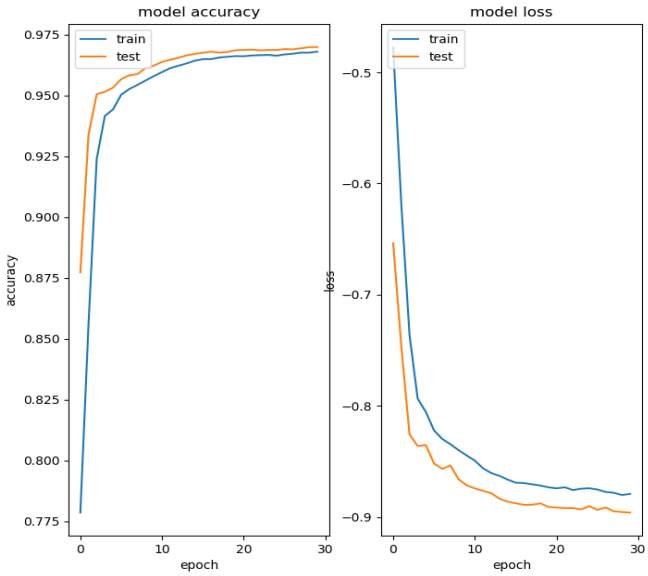


Figure 17. Model accuracy and loss of model applied U-Net on clustered train set and test set.

7. Predicted Result Evaluation

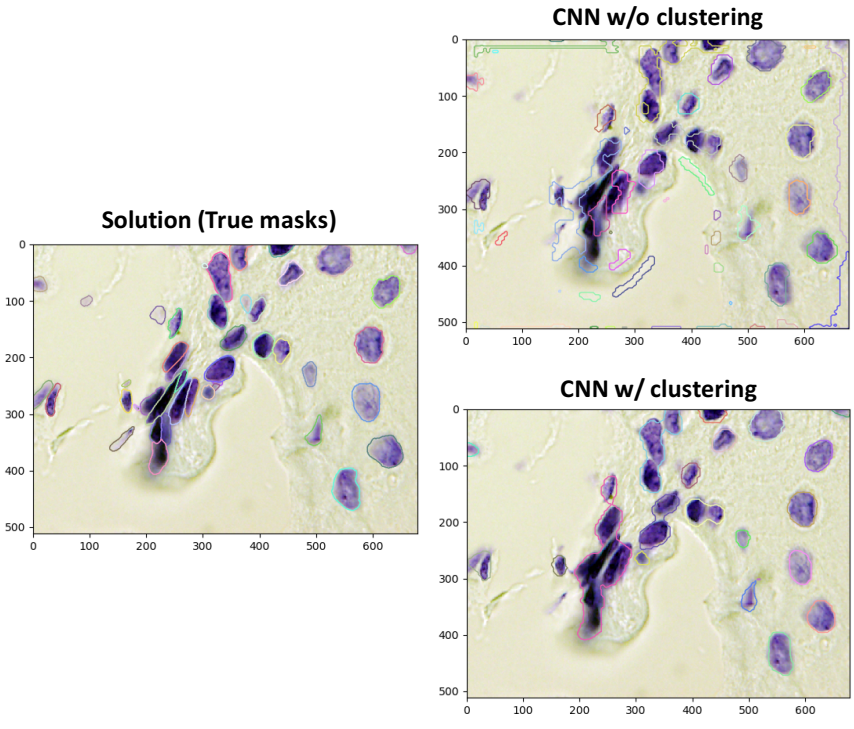
Table 2. Mean average precisions

Avg. precision	w/o clustering	w/ clustering
CNN	0.238	0.284
U-NET	0.364	0.427

After training the model, we apply our models to predict the "true" test set (the test set in the previous model validation part is partitioned from the training set). Note that the predicted results are evaluated on the mean average precision at different intersection over union (*IoU*) thresholds, as defined in the Introduction. The overall result is shown in Table 2. Overall, U-Net performs better than CNN with or without clustering. Most importantly, clustering the data during the pre-process indeed improves the performances of both CNN and U-Net. Let us break it down by visualizing the images and calculating the precision matrices of each case (gladly, the solution of the test set has been posted so we can calculate the precisions over different *IoU*).

CNN

Although the mean average precision over all images is not high for the CNN result, it is improved while having data clustering as preprocessing. Figure 18 shows an example image that the average precision improved a lot, where different masks are represented with different colors. As most of the training images are black and white, it is expected that the model improves a lot for specific types of images that are colorful after clustering. The one without clustering performs very badly (the precision at *IoU* threshold of 0.5 is already very low) because of a large number of false positives (*FP*). It can be attributed to the fact that the model is trained by a majority of images with a black background and white masks. After clustering, the model is able to capture the most essential feature of masks that locates at the dark purple pixels in this type of images. However, the model is not able to distinguish separate masks when multiple masks are very close to each other.



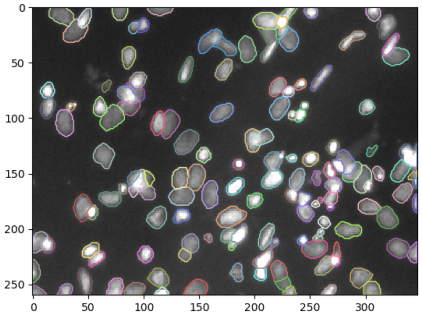
Threshold	TP	FP	FN	Precision
0.5	13	40	12	0.2
0.55	13	40	12	0.2
0.6	8	45	17	0.114
0.65	6	47	19	0.083
0.7	4	49	21	0.054
0.75	4	49	21	0.054
0.8	2	51	23	0.026
0.85	0	53	25	0
0.9	0	53	25	0
0.95	0	53	25	0
Avg. precision				0.073

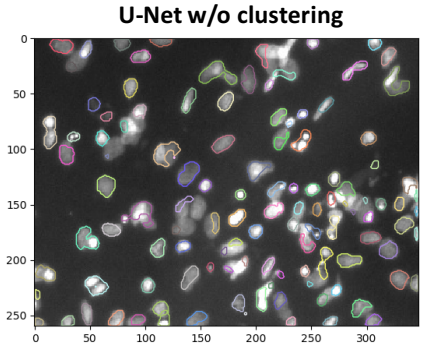
Threshold	TP	FP	FN	Precision
0.5	18	3	7	0.643
0.55	18	3	7	0.643
0.6	17	4	8	0.586
0.65	16	5	9	0.533
0.7	10	11	15	0.278
0.75	10	11	15	0.278
0.8	5	16	20	0.122
0.85	4	17	21	0.095
0.9	0	21	25	0
0.95	0	21	25	0
Avg. precision				0.318

Figure 18. The test images and the true or predicted masks superimposed. Different masks are represented with various colors. The tables on the right show the process of calculating average precisions for both CNN cases.

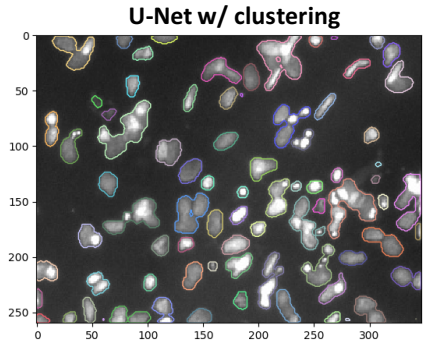
U-Net

It is also observed that clustering improves U-net's prediction on masks. An example image with superior average precision after clustering is shown in Figure 19. The original U-Net model predicted masks only when the color contrast is large enough, which could be due to the features learned from other types of images or possibly overfitting; therefore, *FP* is high at all *IoU* threshold above 0.5 (most predicted masks did not overlap over 50% with the true ones). The predictions from the U-Net model with clustering match the solution nicely. However, the same issue occurs that the algorithm was not able to distinguish masks that clumps up (also occurs in the CNN algorithm). The mask images that we fed in were the superimposed ones, so the fact that there could be clustered masks is not learned by the algorithm. We could have treated this problem with image post-processing, but we focused more on the preprocessing.





Threshold	TP	FP	FN	Precision
0.5	48	62	18	0.375
0.55	44	66	22	0.333
0.6	41	69	25	0.304
0.65	35	75	31	0.248
0.7	33	77	33	0.231
0.75	27	83	39	0.181
0.8	8	102	58	0.048
0.85	6	104	60	0.035
0.9	0	110	66	0
0.95	0	110	66	0
Avg. precision				0.176



Threshold	TP	FP	FN	Precision
0.5	58	16	8	0.707
0.55	57	17	9	0.687
0.6	54	20	12	0.628
0.65	52	22	14	0.591
0.7	49	25	17	0.538
0.75	47	27	19	0.505
0.8	46	28	20	0.489
0.85	34	40	32	0.321
0.9	16	58	50	0.129
0.95	1	73	65	0.007
Avg. precision				0.460

Figure 19. The test images and the true or predicted masks superimposed. Different masks are represented with various colors. The tables on the right show the process of calculating average precisions for both U-Net cases.

CNN vs U-Net

As shown in Table 2, U-Net performs much better than CNN on predicting masks (higher mean average precision). Figure 20 shows images where U-Net is able to make better predictions than CNN while both algorithms are trained with clustered data. Although U-Net is a more sophisticated algorithm, it is found that CNN is more sensitive to noise. Predictions of U-Net are with more well-defined and smoother boundaries. It is difficult for us to probe into the reasons of why U-Net is better, but we noticed that most of the contenders in the competition use more complicated models to get a higher score.

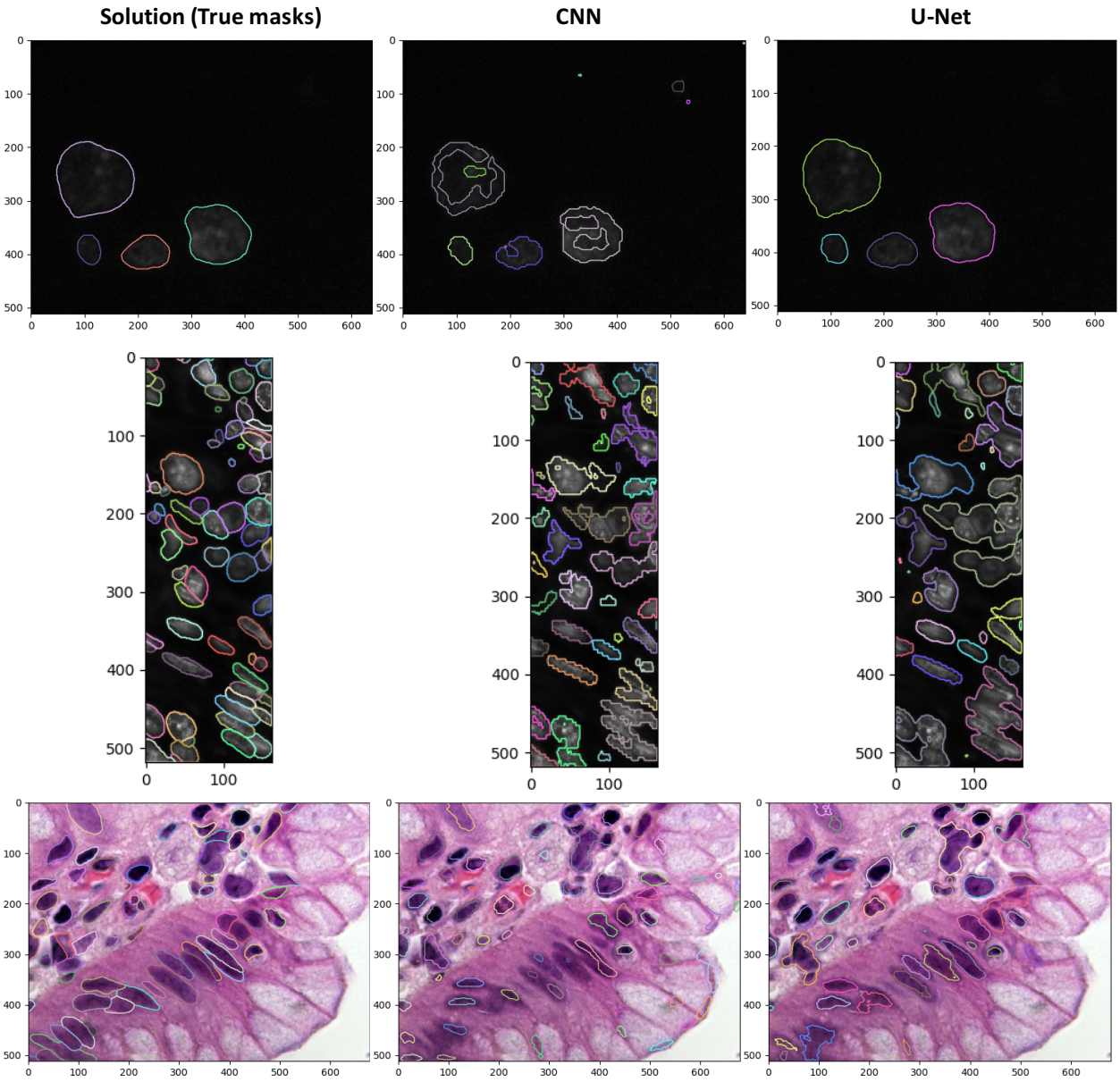


Figure 20. Comparison of masks between the solution, CNN and U-Net (both are trained with clustered data).

Success vs complete failure

To better understand the performance of the models, images where both CNN and U-Net are successful in predicting masks are shown in Figure 21. While the color of masks is in strong contrast to the background, both algorithms are able to capture the feature of masks. However, in some cases, both algorithms fail to obtain predictions that are close to the true ones, as shown in Figure 22. It is found that features of masks are hard to be learned while images are with gradient colors or objectives that are in the same color as the masks.

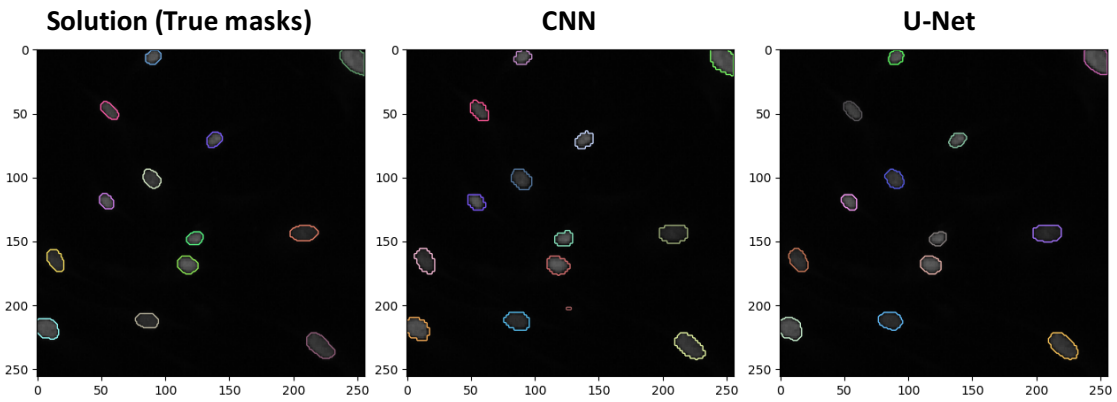


Figure 21. Images that both CNN and U-Net algorithms made excellent predictions.

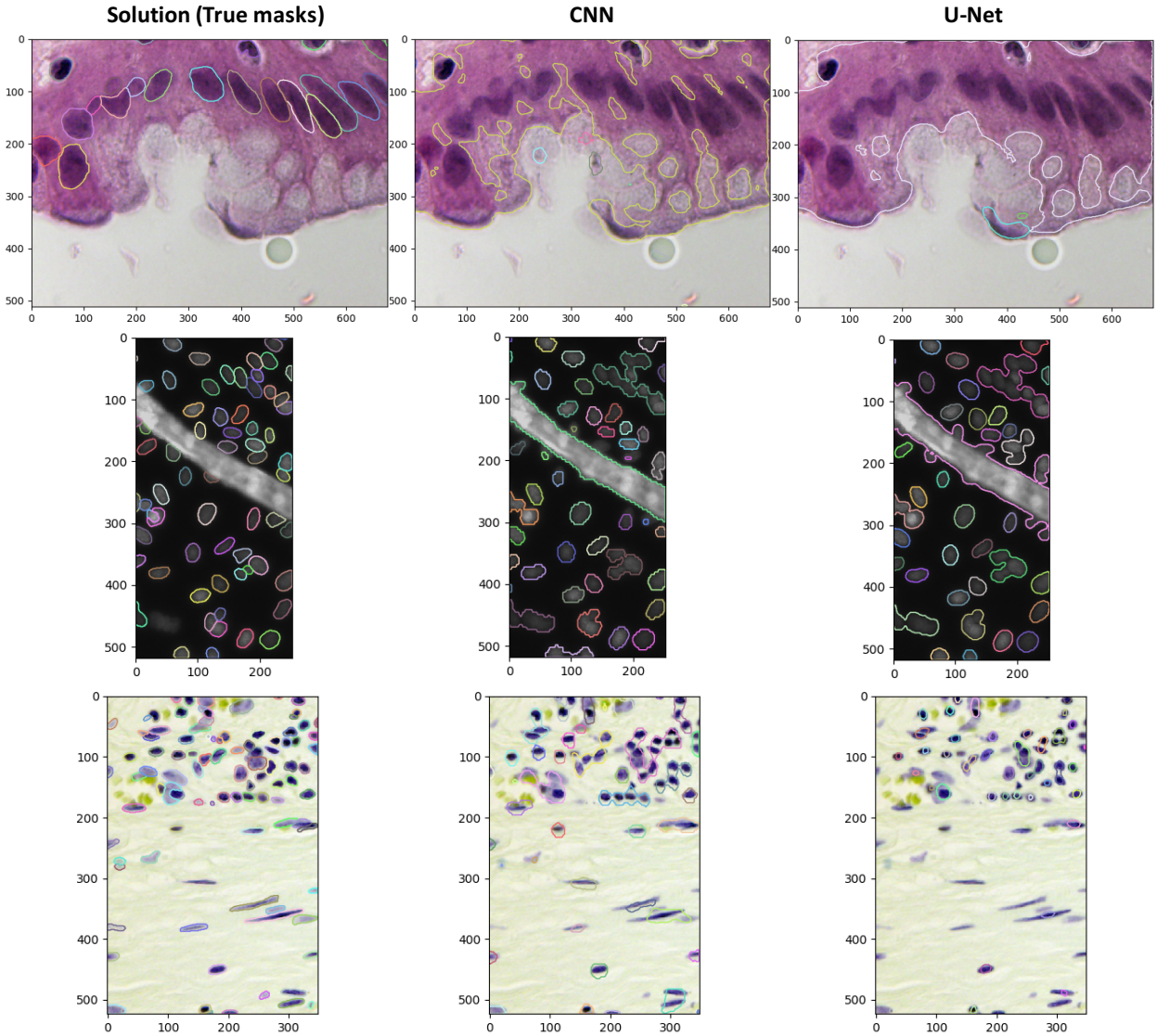


Figure 22. Images that both CNN and U-Net algorithms made poor predictions.

8. Conclusions

We implemented two models to predict nucleus from biological images. We came up with ideas of clustering preprocessing to deal with the data with various cell types, magnifications, and imaging modalities. We have shown that clustering does improve predictions in both models. Although our best model has definitely yet reached the competitive level as we did not have experience in image processing, this work gives insights on how to better train a machine learning model with data mining techniques.

9. Contributions

Report

Kevin Shen: Background, introduction, exploratory data analysis, data preprocessing: clustering, predicted result evaluation, and conclusion.
Kai Wen Liu: Model building and training, and model validation.

Coding

Kevin Shen:
1. Exploratory data analysis (get data statistics and analyze RGB distributions)
2. Clustering (apply K-means on the training set based on RGB, and "predict" the cluster of the test set)
3. Result evaluation (decode RLE-encoding prediction files and superimpose predictions onto original images; calculate the precisions at different IoU for each prediction)

Kai Wen Liu:
1. Built the simple CNN and U-Net models and tried different IMG_WIDTH, IMG_IMG_HEIGHT, and several combinations of activation functions to find the best for the convolution step.
2. Analyzed the result of accuracy, loss, and IoU of models.
3. Convert the predict result to a RLE file.

References

1. Ronneberger O., Fischer P., Brox T. (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI