Development of Medical Image Analysis and Machine Learning Solution to Solve Histo-Pathological and Cyto-Genetic Problems

Internship Report

by Onur Basci
onur.basci@etu.u-paris.fr

Internship done at Argenit
from 30th of May to 5th of August

Supervisor - Argenit
Abdulkerim CAPAR
kerim@argenit.com

Academic tutor
Elise BONZON
elise.bonzon@u-paris.f

Université Paris Cité
Licence 2 - Parcours Informatique
2021-2022

Summary

1 Acknowledgment

I would like to express my deepest gratitude towards Abulkerim Capar for accepting me as an intern at Argenit, supervising and sharing his knowledge during my Internship. I also want to thank Dursun Ali Ekinci for helping me to adopt to the working environment, answering all of my questions and guiding me. Additionally, I want to thank Ramazan Cagac and Ahmed Can Ozcan for helping me with technical problems. It was a pleasure to working with you. I want to thank Élise Bozon for being my Academic tutor and answering my questions during internship. Finally I would like to thank Nelize Dorta for the internship option.

2 Summary Sheet

2.1 Subject

Development of medical image analysis and machine learning solution to solve histo-pathological and cyto-genetic problems. I study mainly about network models for chromosome detection and classification of images obtained from Argenit's microscope systems. I also studied about Tumor cell segmentation which is useful for the breast cancer diagnosis. During the internship I often work remotely with some meetings at the company. The enterprise provided me a computer to work with computationally demanding deep learning models. I generally used Python and its library such as OpenCV, Pytorch, Numpy.

2.2 About Argenit[1]

- **Company Adress**: ITU Teknokent ARI 1 Nr:31/32 34469 Istanbul/Sariyer Turkey
- **The person in charge of the internship in the company**: Mr CAPAR Abdulkerim
- **Title**: Board Member CTO

ARGENIT aims to help physicians make the best diagnosis in the fastest way with its advanced biomedical imaging systems and develops new diagnostic solutions tailored to laboratory needs that deliver high-quality clinical value and user experience for the clinical and research communities. Digital systems from ARGENIT provides the essential tools that laboratories today need to scale and streamline their exponentially increasing number of diagnostics and demands.

2.3 Works Done During the Internship

- Preparation of cytogenetic images and their labels to work on network models.
- Using Python and OpenCV for all kind of data manipulation to work with neural network models.
- Determining the network models and parameters that will detect and classify chromosomes, with the highest success rate possible, on the PyTorch platform.
- Achieve 99% success rate for the chromosome detection.
- Achieve 96.3% mean average precision for the chromosome classification.
- Segmentation of tumor cells containing KI-67 proteins to get all the necessary label data for training neural network models for the classifications.

3 Introduction

3.1 Chromosome Analyses System

The advancement of artificial intelligence and particularly neural networks in recent years brings new solution to computer vision problems such as object detection, segmentation and classifications. Even though there were different methods for these problems before, Artificial Neural Networks (ANN) provides generally way faster and successful results when enough data is provided. Therefore lots of domains increased the usage of ANN to solve their problems, like in biomedical. Argenit uses neural network methods to provides diagnosis from microscopic images. The Chromosome Analyses System software of Argenit, given an image of human cell chromosomes taken during the mitoses or meiosis, detects and classifies the chromosomes and presents them.

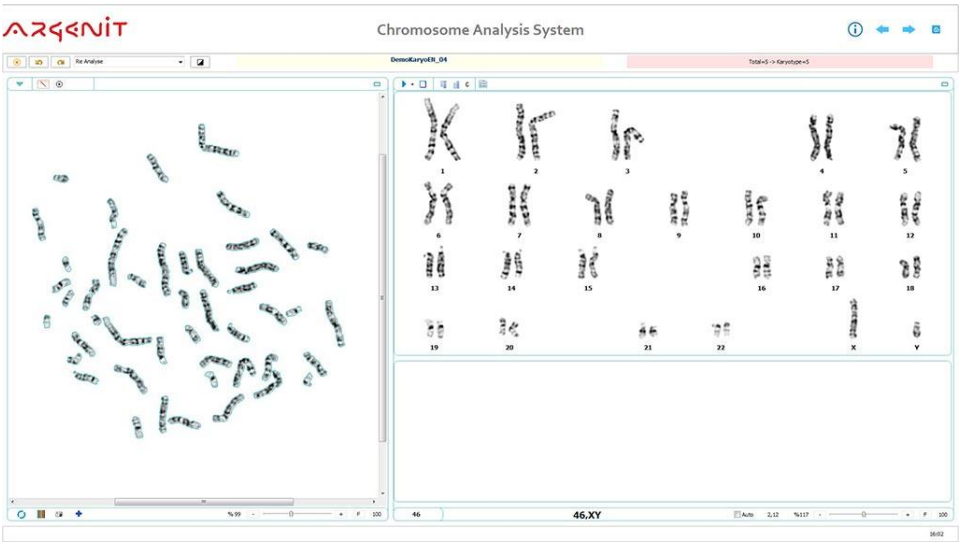

Figure 1: An example of the usage of Chromosome Analyses System

3.2 How Does It Work

Chromosome Analyses System basically works in two different stages: detection and classification. The software takes an image of chromosomes and uses some segmentation techniques to detect all chromosomes. Then it uses that information to surround all chromosomes and cut them into smaller images. Thus, it analyses every chromosome images separately and can easily classify them. Finally it presents all chromosomes with their labels in another window.

3.3 What Was My Job

The problem with this approach is the execution time. Because of the 2-stage classification, the software works slowly. The segmentation and the classification with an ANN for each of the chromosome images increases the execution time considerably. During the internship I worked on the artificial intelligence development and not the software development of the Chromosome Analyses System. My main task was to try different methods using only one CNN to directly classify chromosomes thus decrease the execution time hopefully without decreasing the success rate.

4 Chromosome Detection

4.1 Yolo

Yolo [2] (you only look once) is a real time object recognition algorithm invented in 2015, proved that it is faster than other object recognition algorithms like Sliding Window Object Detection, RCNN, Fast RCNN etc. Also Yolo does not only classify images in an object, it localizes them with a bounding box(i.e. it finds boxes surrounding objects). That is why, my first task was to try this new algorithm for our problem. I used Yolo version 5, released by Glen Jocher in 2020, which uses pytorch deep learning framework. Yolo uses bboxes labels to indicate objects in an image. A bbox contains the information of class, coordinates of box (x, y), width and length.

Figure 2: example of a yolo bbox label

For instance in this example the person's bbox is $\begin{bmatrix}0 & 0.48 & 0.63 & 0.69 & 0.71\end{bmatrix}$ if we suppose that the person class is 0.

4.2 Detection Results With Yolo

For the detection task, Argenit provided me a large quantity of image and label data (≈ 7000). After I prepared the label data in Yolo format, I trained the Neural Network with different parameters. Yolov5 accepts various parameters such as epochs (the number of passes of the entire training dataset), batch size (the number of training examples utilized in one iteration), image size (the size of input images), optimizer (used for the gradient descent) etc. which can effect considerably the obtained results. I have done 3 training with these parameters for the chromosome detection task.

image size	batch	epoch	optimizer	detection rate
1280	4	25	Adam	99%
512	4	25	Adam	98%
512	4	25	SGD	99%

As you can see we get 99% with the first and third row parameters but 98% with the second row parameters. 99% proves us the efficiency of Yolo5 so we can start trying Yolov5 for classification.

5 Chromosome Classification

5.1 Performance Metrics

Before talking about Yolov5's classification results, I would like to explain some performance metrics. Because of multiple classes the analysis of success of an ANN can be quite challenging. These performance metrics will help us to analyse the success of an ANN.

5.1.1 Confusion Matrix

A confusion matrix is an $n \times n$ matrix (with n = number of classes) for classification with actual values on one axis and predicted on another.

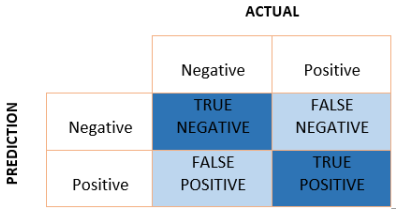

Figure 3: Confusion Matrix[3]

Let's explain some metrics learned by a confusion matrix.

- **True Positive** (TP): model correctly predicts the positive class (prediction and actual both are positive).
- **True Negative** (TN) — model correctly predicts the negative class (prediction and actual both are negative).
- **False Positive** (FP) — model gives the wrong prediction of the negative class (predicted-positive, actual-negative).
- **False Negative** (FN) — model wrongly predicts the positive class (predicted-negative, actual-positive)

These values are good to see how well the ANN prediction is, but we would like to have a percentage value to better understand the success rate for each class. That's why we use precision and recall.

5.1.2 Precision and Recall

$$precision = \frac{TP}{TP + FP} \quad (1) \qquad Recall = \frac{TP}{TP + FN} \quad (2)$$

Precision determines out of all the positive predicted, what percentage is truly positive. And Recall determines out of the total positive, what percentage are predicted positive. These values lie between 0 and 1 therefore give us a better understanding for the succes of the ANN. But in our case, for a 24 classes neural network, It would still be difficult to analyse the ANN with 24 different precision and recall values. That is why, to compare different ANN's confusion matrices, I calculate the average of precision per classes. There is also another numerical metrics called mean Average Precision (mAP) between 0 and 1, that indicates how successful an ANN is.

5.1.3 Mean Average Precision

Before explaining what is mean Average Precision, let's explain what is Average Precision (AP). AP is a popular metric in measuring the accuracy of object detectors. Basically it is defined as the area under precision-recall curve, or mathematically defined:

$$\int_0^1 p(r)dr \tag{3}$$

precision and recall are always between 0 and 1. Therefore, AP falls within 0 and 1 also[4]. But AP is calculated for only one classes. That is why we use mean Average Precision. As you can understand by its name, mean Average Precision refers to average of APs for each classes. Now that we've explained the performance metrics to compare different ANN, we can understand better how well Yolov5 is for the chromosome classification.

5.2 Chromosome Classification

In humans, each cell normally contains 23 pairs of chromosomes, for a total of 46. With allosomes (sex chromosomes) we have 24 classes to classify. Once I finished prepared the new data provided by my supervisor, and the configuration file for 24 classes, I trained Yolov5's CNN with the same parameter as the detection but with higher epochs (= 50). Here is the confusion matrix obtained by the first trained model.

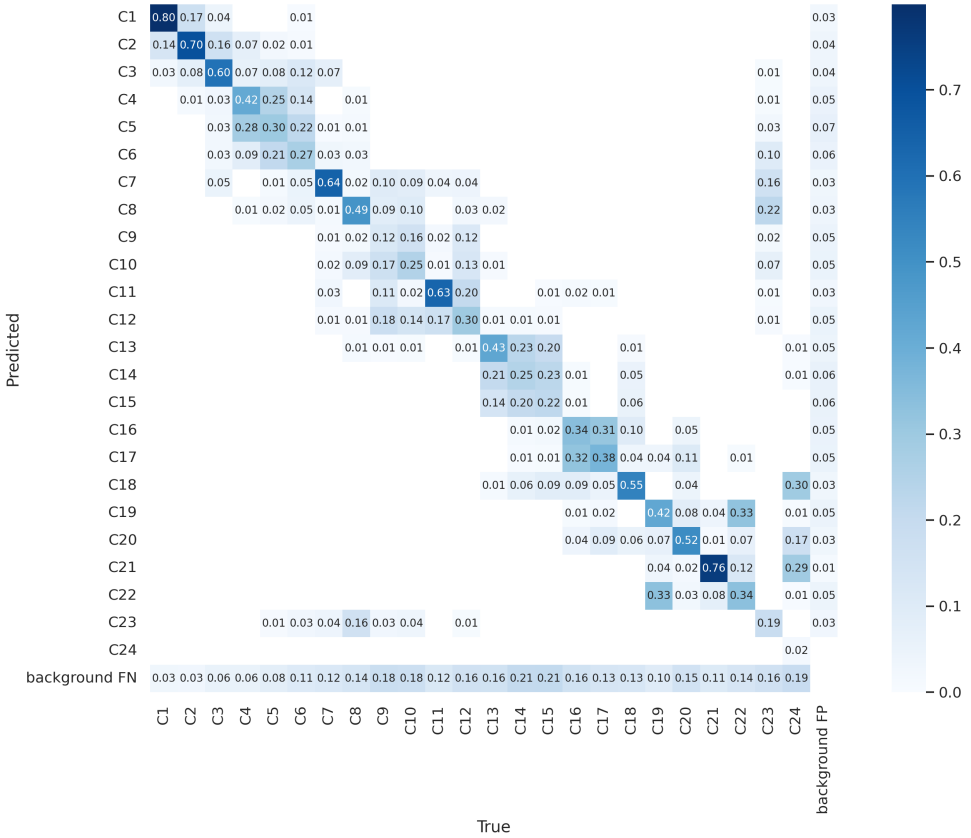

Figure 4: Confusion Matrix (image size = 512, batch = 4, epoch = 50, optimizer = SGD

This matrix contains 24 classes for predictions and ground truths. It also contains background FN row and a background FP. The background FN indicates the percentage of background predictions where the ground truth was actually a chromosome. And the background

FP column indicates the percentage of chromosome predictions where the ground truth was actually background. With that in mind we can say that the first model did a pretty bad job. We can see that the best right prediction is 80% for the 1th class and the worst is for 24th class with 2%! If we want to get one numerical value explaining the confusion we can calculate the average of precisions per classes (average of sum of diagonal ligne) and than we get approximately 41%. So now we should find the right training parameters to get the maximum success for the model.

5.2.1 Hill Climbing Algorithm

At this point, we can actually see this as an optimization problem. We can consider the Yolov5's CNN as a function with inputs are parameters and output is the average precision per classes. Thus, the problem of finding the best parameters to get the most successful CNN as possible, becomes a problem of finding the local maximum of a multiple variable function. Therefore I used one of the classic optimization algorithm called Hill Climbing Algorithm to find out the local maximum of our "function". We can define the algorithm with this pseudo code.

Algorithm 1 Hill Climbing

```
0: begin
   i ← initial solution
   repeat
      generate an s ∈ Neighbors(i);
      if f(s) > f(i) then
         i ← s;
      end if
   until f(s) ≤ f(i) for all s ∈ Neighbors(i);
```

So this algorithm simply consists of starting from the initial configuration, to evaluate the neighboring solutions, and to choose the best of these, and to repeat the operation until arrive at a better position than the neighboring positions. In our case, we can apply this algorithm by only changing one parameter and stabilizing others and then repeat this process for other parameters too. For example here is an application oh Hill Climbing for batch size.

Paramaters:	
Image size_Batches_Epochs_Optimizer_LearningRate	
Batches	
512_2_100_Adam_0.01	0.68125
512_4_100_Adam_0.01	0.720833333
512_8_100_Adam_0.01	0.718333333
512_16_100_Adam_0.01	0.731666667
512_32_100_Adam_0.01	0.723333333

Figure 5: Application of Hill Climbing for batch size

As you can see it appears that batch size = 16 gives us the local maximum. By repeating this approach for each parameter we find that these parameters gives us local maximums.

- Image size = 1100
- Batch size = 16
- Optimizer = SGD (Stochastic gradient descent)

And intuitively, since epoch means, how many times a model is trained by the data, as it increases the model should be more successful. But it also increases the training time so we cannot increase it as high as we want. So I trained the model again with these new parameters and epoch = 350. After a long training we got approximately 91% average precision! It is 50% higher than the first training. Here is the confusion matrix.

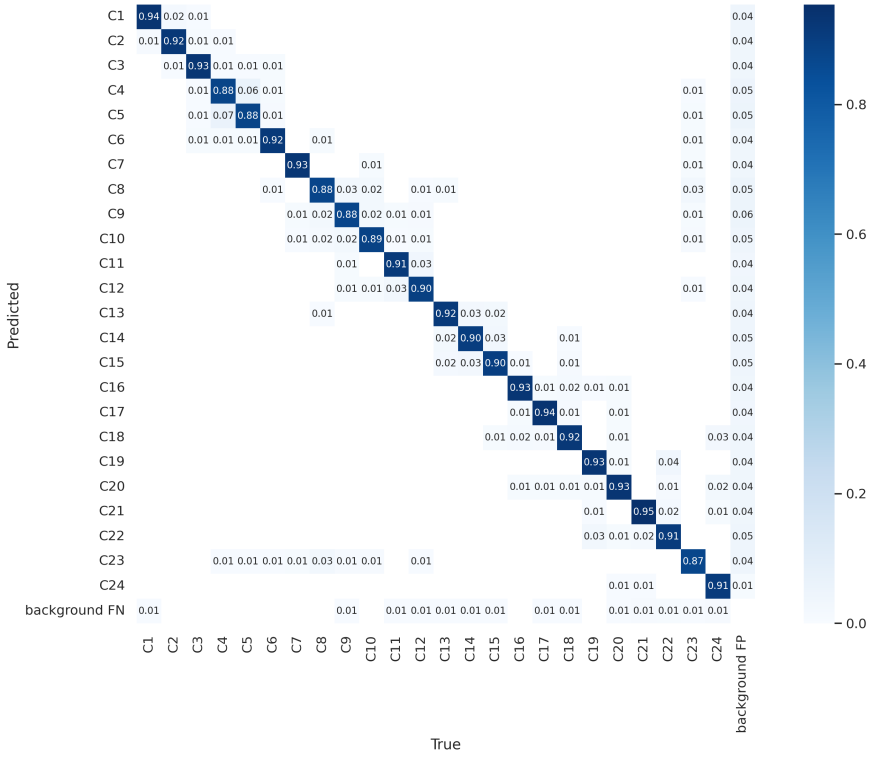

Figure 6: Confusion Matrix (image size = 1100, batch = 16, epoch = 350, optimizer = SGD)

As you can see, we get a clear diagonal line as wanted. But, even though 91% is good, it can be better. That is why after optimizing our model I started to work on the data.

5.2.2 Chromosome Data and Yolo

Data is one of the most important thing in depp learning. An ANN basically learns from the data that we provide. Therefore I decided to do some addition to my data with some augmentation techniques. Augmentation techniques in deep learning are basically ways to increase the data number without having new data. One of the most popular ones are rotating, flipping and scaling. In my case I wrote a python code to read all my data and flip half of my them horizontally and the other half vertically. With that way I doubled my data. With this new data set I trained a new model with same parameters (except I decreased epoch to 200 because as the data number increases, the training time also increases) and I got 92.7% average precision. It is 1.7% better than the last time !

Figure 7: Example of a vertically flipped image

After that, while I was searching some advise to train more successful models with Yolov5, I saw that adding background images to the dataset reduces Background F P therefore increases TP values. As I've already the bbox labels for each chromosome, it is easy to obtain background images. I wrote a python script that reads all the data, cut all the chromosomes from the image according to bbox labels and fills these areas with white pixels (because the data background is white).

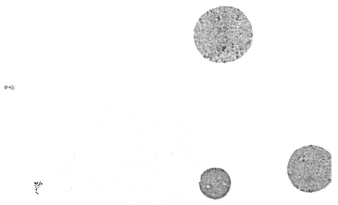

Figure 8: Example of background image

With these adjustment the average precision of the trained model becomes 93.2%.

5.2.3 No Enhanced Data

The final adjustment that I made for the data was completely changing it. The current data that I was using was in fact not directly taken from microscopes. It has been enhanced by whiten the background. This, in fact, adds an additional stage to the software and increases the execution time. That is why my supervisor asked me do the same training with no enhanced data. So I prepared the data, added background images, added augmentations and finally trained the model with same parameters. As a result we get 93.6%. This result is 0.4% better than the last result, but more importantly it takes off an additional process. I find the increase of the result quite intriguing because it means that even though the enhanced image looks more comprehensible to human eye, it has also less information than the original image which makes the difference.

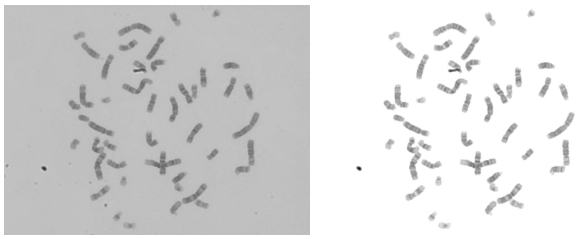

Figure 9: Enhanced no Enhanced comparison

5.2.4 Confidence Threshold

Like most of the object recognition models, Yolov5 also uses a confidence score for its predictions. For each of the prediction yolov5 gives us also a value between 0 and 1 determining how sure the model is for that prediction. If this value is 1 it means that the model is certain and if it is 0 it means that it has no idea. The confidence Threshold is the limit deciding at which point do we consider the model's predictions. By default, this value is 0.25 so, for instance if the model makes a prediction by 0.2 certainty yolov5 does not consider it as a prediction. Before looking for the advantages and disadvantages of changing the confidence threshold, I would like to explain two types of detection error.

- **False Positive detection error**: The model predicted chromosome but the image was background.
- **False Negative detection error**: The model did not detect that the image contains a chromosome

We can intuitively guess that the variation of the confidence threshold would effect these two values. In fact if we increase the threshold, the model should do more "certain" guesses therefore the false positive detection error should decrease and for the same reason the model could detect less chromosomes. At this point I realized that, even though the confusion matrix gives us the background false negative numbers, there is no indication for the false positive detection error. That is because of how the confusion matrix is implemented in yolov5's metrics.py script. This script plots every block of the confusion matrix by the block it self divided by the sum of the column. So I modified the script to print False Negative detection error (each block of the lost column / the sum of the row). And then I compared our last model with confidence threshold = 0.25 (default) and confidence threshold = 0.50. Here is the result.

Model NoEnhanced_1100_16_200_SGD_0.01	**confidence threshold = 0.5**
	Average	**Classes**
False Positive detection error	0.01624543	
False negative detection error	0.01333104	
Sum	0.02957647	
Model NoEnhanced_1100_16_200_SGD_0.01	**confidence threshold = Default (0.25)**
False Positive detection error	0.10862375	
False negative detection error	0.00388618	
Sum	0.11250993	

Figure 10: False Postive/negative detection errors for confidence = 0.25 and 0.5

So, firstly we realize that, with the default confidence score there is a huge "over prediction" problem for the model. It seems that 10% of the model predictions are actually backgrounds. On the other hand, it seems that the missed detections are pretty low (0.3%). And we can also see that our hypothesis worked in this case. When we increased the threshold to 0.5, the False Positive detection error decreased and False Positive negative detection error increased.

But the good things while the FP detection error increased greatly, the increase of the FN detection stays negligible. In fact we can simply compare the sum of the detection errors and see the difference. In addition, due to the more certain model predictions, the increase of the confidence threshold increased also the average precision to 94.7%. At this point we had no more ideas to increase the success of the model so we decided to search for new models to study our problem.

5.3 Other Models

5.3.1 Yolov6[5]

There are other object detection frameworks besides yolov5. In fact, a non official new version of yolo was released in June. They claim that Yolov6 achieves higher mAP than Yolov5 on coco val2017 dataset. So it is worth a try. I trained Yolov6's model with the same parameters as my last Yolov5 model parameters as a reference point.

YOLOV5/YOLOV6 COMPARISON						
	parameters:	batch: 16	img_sz:1100	epoch: 50	Opt: SGD	lr: 0.01
		Yolov5	**Yolov6**			
mAP_0.5:		0.957	0.964			
mAP_0.5:0.95		0.873	0.793			
recall		0.907	0.859			
	parameters:	batch: 16	img_sz:1100	epoch: 200	Opt: SGD	lr: 0.01
		Yolov5	**Yolov6**			
mAP_0.5:		0.969	0.972			
mAP_0.5:0.95		0.898	0.822			
recall		0.943	0.897			

Figure 11: Yolov5-Yolov6 comparison

We can see some metrics that yolov6 provides us and their correspondence for yolov5. mAP0.5 corresponds to mean average precision for confidence threshold = 0.5. And mAP0.5:0.95 refers to mean Average Precision from 0.5 to 0.95 with a step size of 0.05. We can see that Yolov6 does a pretty good job while confidence threshold is equal to 0.5. But for the other cases, yolov5 does a better job. But as we can choose which confidence score to use for the inference, it is not a problem. We can also see that yolov5 did a better job for recall. But we can surely search other parameters to increase these value too. The problem with yolov6 is it's evaluation metrics. Yolov6, unlike yolov5, uses cocoAPI's evaluation script which gives us much less information about the model's predictions. There are no prediction per classes, background detection values, average precision per classes etc. So even though it is slightly better in mAP and it shows potential we decided to stay with yolov5.

5.3.2 Training With Cutted Chromosomes

In the Introduction part of my report I talked about how the Chromosome Analyses System worked. Unlike yolo, it doesn't study the totality of the image but it cuts the chromosome and classify them separately. This surely increases the execution time but analysing only one chromosome instead of the whole image seems easier and could give us better results. Chromomse Analyses System currently uses this technique, but while implementing it they

didn't train a lot of models. So my supervisor asked me to train different models to compare their results. And if the success is high enough we can use it instead of yolov5 model. So I wrote a python script to cut all chromosomes from my image data and write all the label in a csv file. Then by using pythorch, I wrote a basic pyhton script that reads the data, and trained several models from torchvision library with these data. I used googlenet, resnet, densenet, efficient, resnext with their different size (renset101, rensnet152 etc.) as models. Finally, I got 94.78% as the best result from densenet201 model(optimizer= SGD, batch size = 64, learning rate = 0.05, epoch = 4). Even though the result is pretty high, we got almost the same result as yolov5. So it would be more reasonable to use yolov5 model as it eliminate one stage and directly classifies images form the whole image.

6 KI-67 Stained Tumor Cell Segmentation

6.1 About KI-67

Ki-67 is a nuclear antigen that is an excellent marker of active cell proliferation in the normal and tumor cell populations. It has been proposed as a useful clinical marker for breast cancer subtype classification, prognosis, and prediction of therapeutic response[6]. In the last week of my internship a medical firm asked us to develop a system which will detect and classify KI-67 containing tumor cell and other tumor cells. The proportion of tumor cells containing KI-67 proteins to other tumor cells can be use to classify the type of breast cancer and maybe in the future to choose which kind of treatment to use.

6.2 Classification

From the result of the chromosome classification we were satisfy with Yolov5. So we decided to use it again for the tumor cell classification. But the problem is, unlike the chromosome classification we don't have annotated data. Therefore we used Shiraz Histopathological Imaging Data Center's (SHIDC)[9] KI-67 dataset. This dataset contains 3356 images with annotated labels. The problem with these labels is they contain only center coordinates for each Tumor cells. But we know that yolov5 requires bbox label. It means that we need the length and height for each cell too.

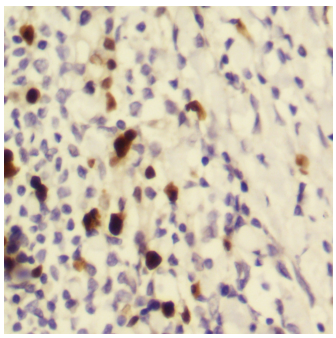

Figure 12: An image from SHIDC data set

So my supervisor asked me to segmentate SHIDC's images thus, if we can separate each of the cells successfully, from their contours we can obtain the bbox information for each cell and train yolov5 model.

6.3 Segmentation

6.3.1 Watershed Algorithm

I started by implementing Watershed Algorithm for tumor cell segmentation. Watershed is a classic algorithm used for segmentation and is especially useful when extracting touching or overlapping objects in images. It separates objects by their Euclidean distance to the closest background pixel. So I wrote a python script, using openCV[7], that firstly converts the input image to grayscale then applies thresholding to separate background and objects. Then it applies watershed algorithm from the distance map obtained by using Euclidean distance. And finally it draws green circles surrounding objects found and puts also red annotation dots got from SHIDC dataset to compare results.

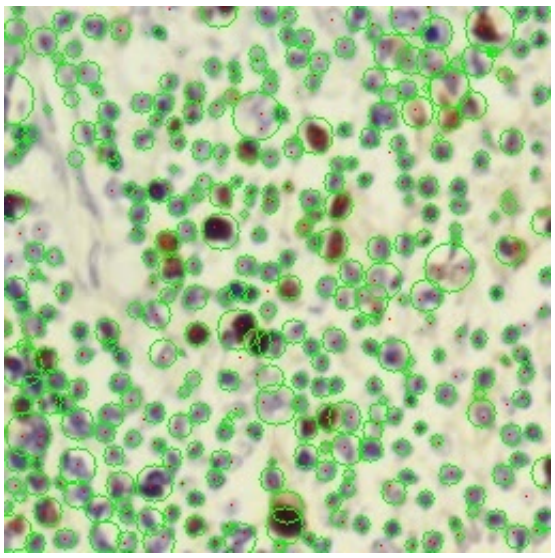

Figure 13: Application of watershed algorithm

As you can see it generally detects cells, however there are also some circles surroundings multiple dots, dots not detected and circles without dots. So we tried some different approaches.

6.3.2 Flood Fill

One of the problem of the application of the watershed algorithm is that we weren't using the annotation labels at all. We have all the center coordinates for each cell. Better to use them! Therefore firstly I tried Flood Fill. Flood fill is an algorithm to identify adjacent values in an image based on their similarity to an initial seed point[8]. So I wrote a python script which reads annotation labels and uses all the coordinates as initial points to apply Flood fill for all annotated cells. The problem with this script is, because there are some cells so similar to background, it also consider the background as a cell. Therefore I modified the program to apply flood fill while the detected object is smaller than a given pixel size. I also modify the script to draw contours instead of circles.

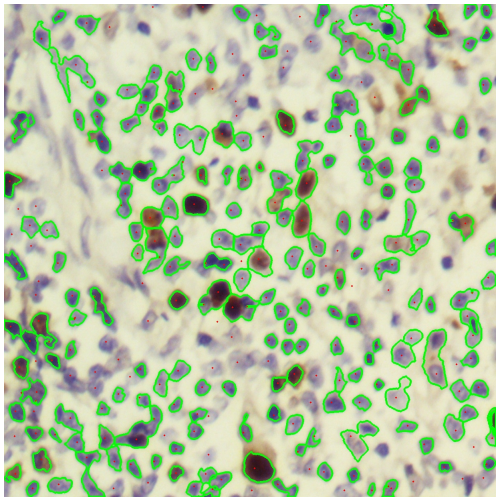

Figure 14: Application of Flood Fill

As you can see there are no more contours without dots. But there are still lots of non detected cells.

6.3.3 Watershed With Cell Coordinates, A Hybrid Solution

Watershed algorithm was "over-detecting", the flood fill was "under-detecting" the cells. So we found a hybrid solution. By default, Watershed algorithm uses local maximums of the distance map obtained by Euclidean distance. So instead of using Euclidean distance we directly use the coordinates as local maximum points. With that way the result should depends to the cell coordinates.

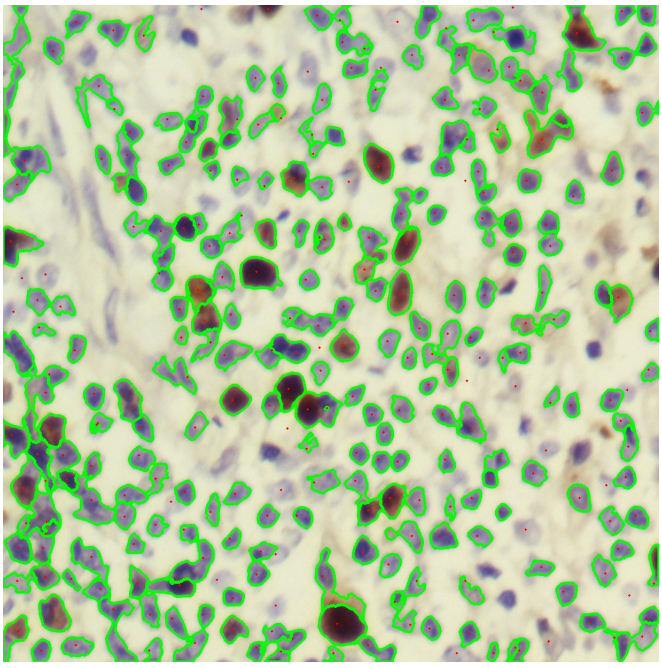

Figure 15: Application of Watershed version 2

There are no contours with multiple dots, no contours without dots and less undetected cells! Even though there is room to progress, unfortunately I finished my internship at this point. So hopefully they continue my work from here and get good classification results !

References

[1] Argenit. https://argenit.com/en/home/.

[2] Yolov5. https://github.com/ultralytics/yolov5.

[3] Confusion matrix, precision, recall. https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262.

[4] map. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-

[5] Yolov6. https://github.com/meituan/YOLOv6.

[6] Ki-67. https://www.frontiersin.org/articles/10.3389/fendo.2021.687244/full#:~:text=Ki67%20is%20a%20nuclear%20antigen,response%20(2%E2%80%934).

[7] Watershed. https://pyimagesearch.com/2015/11/02/watershed-opencv/.

[8] Flood fill. https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_floodfill.html.

[9] Ki-67 dataset. https://shiraz-hidc.com/service/ki-67-dataset/.