Micro-Net-508 for Gland Segmentation in Microscopic Images

Abhishek Bhatt
Department of Computer Science
Rutgers University
Piscataway, NJ 08854
ab2083@scarletmail.rutgers.edu

Ashish Podduturi
MBS Department
Rutgers University
Piscataway, NJ 08854
ap1822@scarletmail.rutgers.edu

Brian Yi
MBS Department
Rutgers University
Piscataway, NJ 08854
bfy2@scarletmail.rutgers.edu

Abstract

For automated microscopic image analysis, segmentation and localization of key structures like tumours, glands, and cells is a crucial step to improve the accuracy of downstream tasks. Deep CNN architectures have been found to be really effective for this task, while also allowing re-usability to segment various structures across different modalities through retraining and hyper-parameter tuning. In this work, we present the details of our implementation of the Micro-Net-508 architecture, which was proposed as the state-of-the-art for gland segmentation from microscopic images. With no implementation available currently and lack of large annotated training data, we found this an interesting problem to contribute towards. Our goal is to compare our implementation of Micro-Net-508 to the model proposed in the paper, and try to reproduce the observed results.

1 Introduction

Micro-Net was proposed as a deep convolutional architecture for segmenting microscopic images by Raza et al. (2019). It is setup to be a unified framework for segmentation of various types of objects (nuclei, cells, glands) in two different types of image modalities (histology and fluorescence microscopy). Micro-Net-252 is proposed for segmentation of individual cells in multiplexed fluorescence images using nuclear and membrane markers, as well as the segmentation of nuclei in histology images. These are significant components in the studies to understand the tumour microenvironment. For gland segmentation, Micro-Net-508 (refer Figure 4) is proposed in order to train on a bigger input image patch size. Compared to 252, MicroNet-508 leverages doubled input size and a modified architecture for group 2 blocks.

For our implementation, we only focus on the Micro-Net-508 model for gland segmentation since the training and evaluation datasets for Micro-Net-252 (for cell and nuclei segmentation) could not be found in the public domain at the time of this submission. Segmented glands in histology images are used for grading the severity of colon cancers. However, gland segmentation in practice is a difficult problem since it requires manual labeling by highly-trained pathologists. In the absence of a unique algorithmic approach, the results vary across different human experts and are hard to reproduce. Through the proposed Micro-Net-508 model, this problem can be addressed.

The main characteristic of this network is that it has additional connected intermediate layers on its downsampling path, and bypasses the max-pooling operation to learn additional parameters for segmentation. This architectural adjustment provides the network with main two benefits: 1) it trains at multiple resolutions of the input image such that it can learn parameters for different gland sizes and shapes at different intensities and textures 2) it retains important information that would have otherwise been lost through the max-pooling operation. In summary, the model can learn context at multiple resolutions and is more robust to noisy data. This proves particularly useful in gland microscopy, in which automatic segmentation is challenging due to the very high variance in size, texture, and structure of glands.

When compared Micro-Net^{-}, the same remaining architecture with the removal of the multi-resolution input and the bypass layers, the proposed Micro-Net network is observed to result in higher training and validation accuracy, as seen in Figure 1.

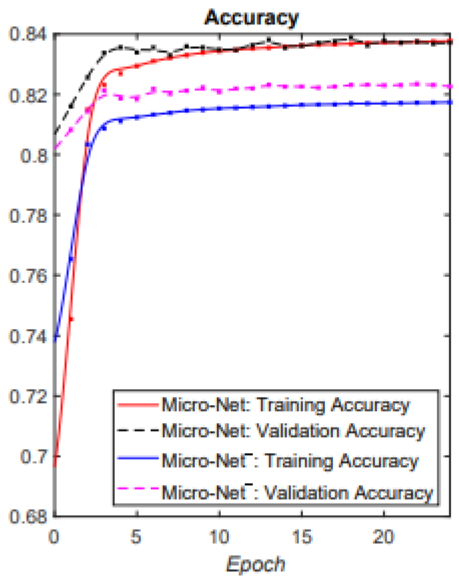


Figure 1: Micro-Net vs Micro-Net^{-}

2 Related Works

There are two main avenues of approach to image segmentation currently available: custom feature engineering, and deep learning.

Most custom feature engineering approaches to cell/nuclear segmentation use some combination of image thresholding, filtering, morphological operations, region accumulation, marker controlled watershed (Yang et al. (2006)); Veta et al. (2013), deformable model fitting (Bergeest and Rohr (2012)), graph cut (Dimopoulos et al. (2014)) and feature classification (Li et al. (2015)). For gland segmentation, Wu et al. (2005) identified initial seed regions based on large vacant lumen regions and expanded the seed to a surrounding chain of epithelial nuclei. Farjam et al. (2007) proposed segmentation by clustering texture features calculated using a variance filter. However, robust segmentation with feature engineering requires more domain knowledge and texture features, and does not generalize very well across different use cases.

With the advent of deep Convolutional Neural Networks, feature learning has replaced feature engineering based approaches. Most of the recent state-of-the-art for vision tasks use CNN based architectures, like fully convolutional network (FCN) Shelhamer et al. (2017), which serves as the benchmark for segmentation tasks using deep learning. U-Net (Ronneberger et al. (2015)) was built as an extension to FCN to conserve the context information. Recently, Sadanandan et al. (2017) employed a variation of fully convolutional network inspired by the improvements in U-Net (Ronneberger et al. (2015)) and residual network architecture (He et al. (2016)) for cell segmentation. Kraus et al. (2016) use multiple instance learning (MIL) to simultaneously segment and classify cells in microscopy images. DCAN (Chen et al. (2017)) employs a modified FCN that simultaneously segments both the objects and contours to assist separating clustered object instance. Manivannan et al. (2018) combined handcrafted features with deep learning for segmentation, but this approach is computationally expensive.

The proposed MicroNet is an attempt to unify and extend the previous works on cell (Raza et al. (2017a)) and gland segmentation (Raza et al. (2017b)).

3 Data Set & Implementation

3.1 Data

Out of the three data sets used in the paper, only one of them is accessible publicly. Both the Multiplexed Fluorescence Imaging Data and the Computational Precision Medicine (CPM) data set for nuclear segmentation were unavailable. So we used the public Warwick-QU dataset from the Gland Segmentation (GLaS) Challenge for training and evaluating our implementation of Micro-Net-508. The dataset consists of 85 training images along with their respective ground truth images that were annotated by pathologists. Figure 2 shows one training example, with a histology image on the right along with its segmented ground truth on the left. The segmented image is modeled as pixel-wise probabilities by Micro-Net-508, indicating whether a pixel is part of a gland or not, which is accordingly set as bright or dark in the output.

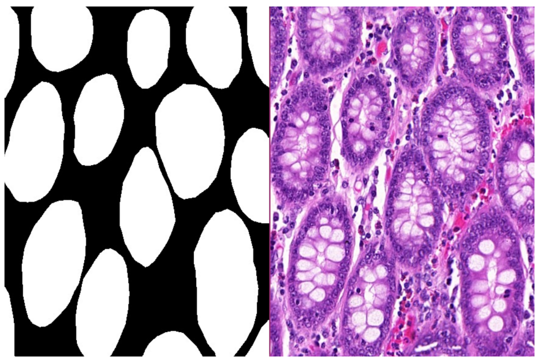


Figure 2: Ground Truth vs Training Image

There is a separate held-out test set of 60 images for final evaluation, after the model training and tuning on the validation set is completed.

3.2 Data Augmentation

Due to expensive and labor intensive preparation and annotation of microscopic images, we have only 85 training images, that is too small for deep learning models. As proposed in the original paper, we use augmentation techniques to setup a larger training set. We applied radial distortions (barrel, pincushion and mustache) on the original images, using the Wand module in Python that is built on top of the ImageMagick library. Each of these three distortions added an additional 85 images to our training set. Next, adding a Gaussian blur on the original images, with a filter size of 12 x 12 and σ ranging from 0.2 to 2; added an additional 1,615 images. We also rotated the images by 90, 180, and 270 degrees to add another 255 images. Finally, we flipped the images across the vertical, horizontal, and both axes to add the final 255 images to the training data. We use OpenCV Python library for these transformations. After this data augmentation, our training dataset increased in size to 2,465 from the original 85 examples. The paper does not mention any information about the training size after data augmentation.

3.3 Training

We did a 90-10 training-validation split with our augmented dataset of 2,380 images. Following the training approach described in the paper, we extract 600 x 600 patches from the training images, where images are padded accordingly if smaller than the stated dimensions. During each epoch while training, the network would pick one of these patches randomly and crop it to a new size of 508 x 508 to create the input. We start with a learning rate of 0.001 and update it according to $lr = lr/10^{\frac{epoch}{5}}$, where $epoch > 0$.

The training loss is defined as the combination of auxiliary losses from Group 4 blocks and the output loss from Group 5 (refer Figure 4)). We compute the log-probabilites corresponding the output probabilities p_{a1}, p_{a2} and p_{a3} from Group 4 as well as p_o from Group 5. The paper presents a weighted cross entropy loss formulation for each of these outputs as:

$$l_k = \sum_{x \in \Omega} w(x) \log(p_{kx}(x))$$

where $k \in a1, a2, a3, o$. $w(x)$ is a weighing function as proposed in (Ronneberger et al. (2015)), which computes a weight for every pixel. Finally the total training loss is computed as:

$$l = l_o + \frac{(l_{a1} + l_{a2} + l_{a3})}{epoch}$$

where $epoch > 0$.

For our implementation, we use the same loss formulation as the paper except for the weight function $w(x)$. We wanted to avoid the additional weight computation given that our implementation is already very computationally expensive. As a result, our implementation gives equal weight for every log-probability across all pixels.

Our code implementation is done using the PyTorch library in Python for deep learning. Refer A for more.

3.4 Evaluation

In the paper, the authors have used various evaluation metrics for each of the three datasets as well as model variants, like F1 score, Object Dice and Object Hausdorff. For our implementation, we focus on the F1 score as the evaluation metric. As shown in Figure 3, the proposed model is observed to beat previous architectures with an F1 score of 0.913 on Test set A, which is the same test set that we use for our implementation.

For any given image, we compare feature maps with pixels values 0 or 1, between the predicted and ground truth image. The F1 score score is then computed as follows.

$$F1score = \frac{2 * precision * recall}{precision + recall}$$

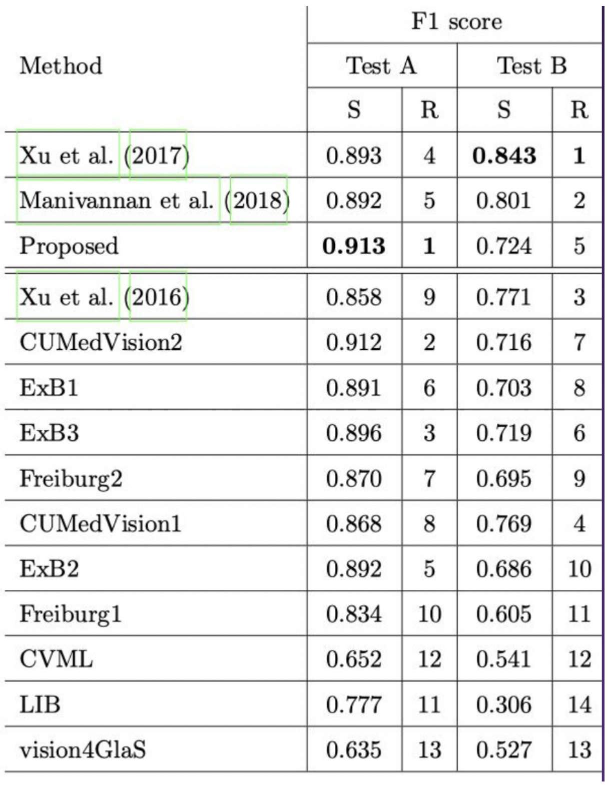
where

$$precision = \frac{count(TruePositives)}{count(TruePositives) + count(FalsePositives)}$$

and

$$recall = \frac{count(TruePositives)}{count(TruePositives) + count(FalseNegatives)}$$

Finally the F1 scores for all images are averaged to get the final F1 score for the model prediction.



Method	F1 score			
	Test A		Test B	
	S	R	S	R
Xu et al. (2017)	0.893	4	**0.843**	**1**
Manivannan et al. (2018)	0.892	5	0.801	2
Proposed	**0.913**	**1**	0.724	5
Xu et al. (2016)	0.858	9	0.771	3
CUMedVision2	0.912	2	0.716	7
ExB1	0.891	6	0.703	8
ExB3	0.896	3	0.719	6
Freiburg2	0.870	7	0.695	9
CUMedVision1	0.868	8	0.769	4
ExB2	0.892	5	0.686	10
Freiburg1	0.834	10	0.605	11
CVML	0.652	12	0.541	12
LIB	0.777	11	0.306	14
vision4GlaS	0.635	13	0.527	13

Figure 3: Benchmark result

4 Architecture

Our implementation of Micro-Net-508 consists of five main blocks as shown in Figure 4. The paper only gives out intermediate feature map dimensions after a set of layers. We use these as our reference to configure individual layers within each set, for which no configuration can be inferred from the given architecture. The complete layer configuration for our implementation can be found in our associated code.

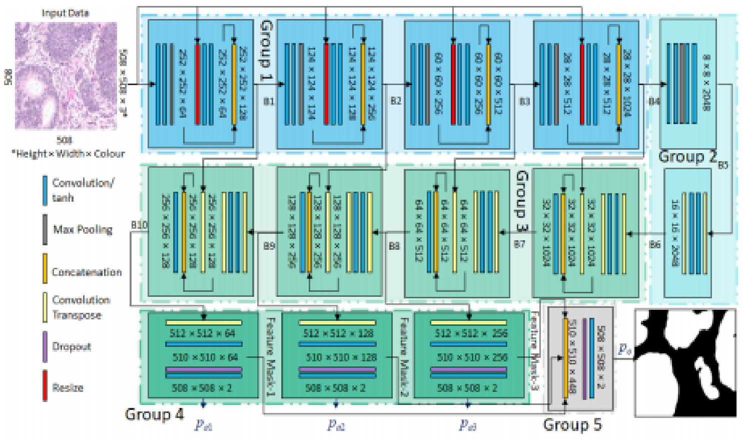


Figure 4: Micro-Net-508 Architecture

4.1 Group 1: Downsampling

Group 1 consists of four blocks (B1-B4) that make up the downsampling path. In each block, the input first goes through two convolution layer layers with a tanh activation after each layer; for the remaining of this architecture section assume that there is a tanh activation after each convolution layer.

This is followed by a max-pooling layer. Afterwards, there is a resizing operation that resizes the image through bicubic interpolation, such that the max-pooling output matches in spatial dimensions to the resized image. This resized, lower resolution input is added in order to keep the information from pixels that are close to a noisy neighborhood that prevents them from having a maximum response. In other words, this lower resolution image allows us to retain smaller feature details such as cell boundaries with extreme intensities that would have been lost during the max-pooling operation. This resizing operation is also useful for training the network on different size glands as mentioned previously in Section 1.

After the resizing operation, the input goes through two more convolution layers before going through a concatenation on the channels dimension, which combines the current output with the earlier output from the max-pooling layer. The output from this concatenation then proceeds to the next block in Group 1. Note that while the feature dimensions (height and width) shrink from size 256 x 256 to 28 x 28 going from B1-B4, the feature depth is doubled at each block and overall increases from size 128 to 1024 by the end of the path.

4.2 Group 2: Bridge

Group 2 consists of two blocks, B5 and B6, that serve as a bridge between the downsampling path (B1-B4) and upsampling path (B7-B10).

B5 consists of two convolution layers, followed by a max-pooling layer, and then two convolution layers again. B6 has a different setup with two transposed convolution (de-convolution) layers, with two convolutional layers in between them. The first block has an output of 8 x 8 x 2048, which then becomes a size of 16 x 16 x 2048 after the second block.

4.3 Group 3: Upsampling

Group three consists of blocks B7-B10 that make up the upsampling path. The output of each block doubles in height and width while the feature depth is halved; an input of 16 x 16 x 2048 becomes 256 x 256 x 128 by the end of this path. Each of these blocks receive one input from the previous block in the upsampling path as well as another input from it's nearest block in the downsampling path. The additional input from the downsampling path is added for the purpose of better localization that better captures context information.

The first input from the previous block in the upsample path goes through a deconvolution layer, two convolution layers, and another deconvolution layer to provide the first output. The second input from the downsampling path goes through a deconvolution layer before being concatenated with the first output. The deconvolution layer serves to produce a segmentation map with the same dimensions as the input image without the need of an overlap-tile strategy. Furthermore, it decreases computational load by reducing the number of necessary patches needed for the segmenation map. This concatenated output goes through a final convolution layer before proceeding to the next block within the upsampling group.

4.4 Group 4: Auxiliary

Group 4 serves to generate auxiliary feature masks that are then fed into Group 5 to produce the final output. The probabilities computed from the Group 4 are used for calculating the auxiliary loss which is one component of the total training loss. Group 4 consists of three blocks, with each block taking it's input from the nearest block in Group 3 (B7-B10). Each block in group 4 creates an auxiliary feature mask by feeding the input into a deconvolution layer, followed by a convolution layer.

For computing the probabilities, this mask is passed through a dropout layer with a drop rate of 0.5, followed by a convolution layer and finally a 2D softmax operation.

4.5 Group 5: Main Output

The three auxiliary feature mask outputs from Group 4 are then concatenated, followed by 50% dropout, a convolution layer, and another 2D softmax operation to get the final output.

5 Implementation Challenges

The authors have not made their code public. We could not find any other publicly available PyTorch implementation for this paper either, until the time of this submission.

When trying to reproduce the results presented by the paper for gland segmentation, lack of a large training set is definitely a challenge. It is not known how many training examples after augmentation were used by the authors to achieve the observed results. So we apply the augmentation techniques mentioned on every image in the original dataset and expect that the resulting training size would be sufficient to match the benchmark results.

Similarly, the paper demonstrates intermediate feature map dimensions after a set of layers within each group and how the blocks of each group are connected. We use these as our reference for configuring individual convolution, de-convolution and max-pooling layers, such that the corresponding feature map dimensions match between the paper and our implementation. Consider Figure 5, where we do not know the output dimensions after the first de-convolution layer (yellow) until after the second convolution layer (blue), going right to left. Lack of information on individual layer configurations may result in our implemented model being different from the authors'. However, we do not consider this a key factor for differences in the results that may arise.

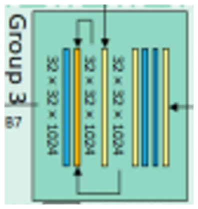
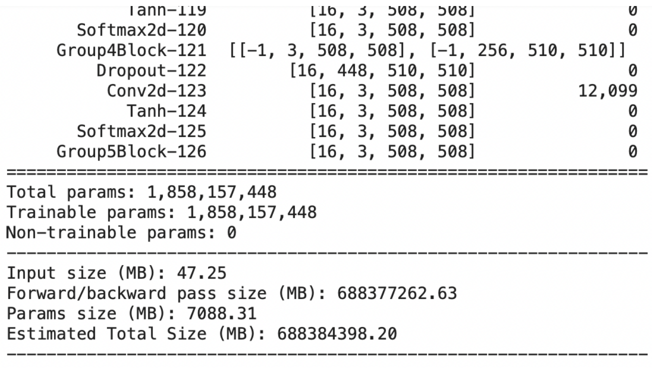


Figure 5: Unknown Layer Configurations

5.1 Computational Challenges

We focused more on finishing the code implementation for the model architecture, data processing and the training setup as described in the paper. However, the greatest challenge that we faced after this process was the ability to train our model with the computing capacity available for our use, which we did not evaluate earlier. For a batch size of 16 and input image size of 508x508x3 as used in the paper, Figure 6 shows our model summary. The complete summary of layer-wise parameters can be found in our associated code.



```
           Tanh-119          [16, 3, 508, 508]               0
      Softmax2d-120          [16, 3, 508, 508]               0
    Group4Block-121  [[-1, 3, 508, 508], [-1, 256, 510, 510]]
        Dropout-122        [16, 448, 510, 510]               0
         Conv2d-123          [16, 3, 508, 508]          12,099
           Tanh-124          [16, 3, 508, 508]               0
      Softmax2d-125          [16, 3, 508, 508]               0
    Group5Block-126          [16, 3, 508, 508]               0
================================================================
Total params: 1,858,157,448
Trainable params: 1,858,157,448
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 47.25
Forward/backward pass size (MB): 688377262.63
Params size (MB): 7088.31
Estimated Total Size (MB): 688384398.20
----------------------------------------------------------------
```

Figure 6: Implemented model summary for batch size = 16 and input size = 508x508x3

The available RAM on Google Colaboratory Pro GPU with High-RAM configuration is 27.4 GB. Our model fails to train on this infrastructure with the given memory availability. We get CUDA out of memory exception either when moving the model instance created on CPU to the GPU, or during the very first epoch of training.

The paper describes the setup used for training as a Windows 10 machine with Intel E5-2670 v2 CPU, TitanX Maxwell GPU and 96 GB RAM. With this configuration, that authors' model takes around 4.5 days to train over 25 epochs. As a result, our implementation could not be trained with the available compute available on Colab Pro. We tried a few approaches to try and fix this problem as described below.

- Moving the setup to the CS student iLab cluster : The original implementation still failed with the CUDA out of memory exception on the iLab setup. The iLab being a shared cluster and busy with other workloads does not guarantee compute to full capacity as we had expected.
- Reducing the batch size : We reduced the batch size for training from 128 originally to 16. However, this does not help since the model is still computationally expensive as seen in Figure 6.
- Using torch.nn.DataParallel : We tried to implement data parallelism at the module level using the DataParallel container in PyTorch. This does not help on Colab where there is access to only one GPU. On iLab, though getting access to 3 devices with this addition, the training still failed with CUDA out of memory exception due to multiple workloads taking up the shared space. Due to time constraint, we could not test this further at sometime when the cluster might be relatively free.
- Resizing the input image / Reducing model size : In an attempt to train a model with smaller training load (at the cost of accuracy), we downsized the input images to 254 x 254 x 3. We reconfigured few layers to adjust the model for changes in the intermediate dimensions due to this. This still does not resolve the memory constraint issue. We believe the model might start training for 64 x 64 x 3 input but that will result in losing out a lot of information useful for learning the segmentation. In the interest of time, we did not pursue further downsizing. Another direction of our thinking was to change the model architecture by removing as many layers as possible to bring down the number of model parameters. Again due to time constraints we could not try this in implementation, since it will lead to the need for re-configuring all the individual layers. Also, we do not have any reference to guide us in this case, since the model might behave completely different from the expected results.

6 Results & Discussion

At the time of this submission, we are still struggling to train our model and collect the evaluation results with the computational capacity available to us. However, we do believe that when trained with sufficient RAM, our associated code implementation would be able to achieve evaluation scores comparable to the original paper with little or no changes.

7 Conclusion & Future work

In this work, we demonstrated one of the state-of-the-art approaches for segmentation of glands from histology images, which is a crucial component of automated processing of microscopic images for digital pathology. Though our current implementation does not reproduce the paper results, with dedicated compute available we can check it's effectiveness as a future extension of this project. Alternatively, we can try implementing a computationally less expensive version of the architecture to start with, and then scale up to the proposed model as an iterative implementation.

Nevertheless, this whole exercise gave us a good insight into the microscopic image segmentation problem and relevant state-of-the-art. We explored implementing deep learning models in a resource-constrained as well as data-constrained setting. We also learnt how to do academic literature survey and approach implementing the code for proposed neural net architectures. The project has given us a good starting point to dig deeper into this area of active research and interest.

8 Contribution

Abhishek contributed mostly to the coding implementation of the Micro-Net-508 with respect to data preparation, augmentation, and model setup.

Ashish's contribution was towards trying out different approaches to make the model memory efficient and train it with the available compute.

Brian contributed to coding the training and evaluation setup and debugging the implementation.

There was equal contribution from all the members towards the project proposal, literature survey, presentation and the final report.

References

Bergeest J, & Rohr K. Efficient globally optimal segmentation of cells in fluorescence microscopy images using level sets and convex energy functionals. In Medical image analysis 2012;16(7):1436-44. doi:10.1016/j.media.2012.05.012.

Chen H, Qi X, Yu L, Dou Q, Qin J, Heng PA. Dcan: Deep contour-aware networks for object instance segmentation from histology images. In Medical Image Analysis 2017;36(Supplement C):135-46. doi:10.1016/j.media.2016.11.004.

Dimopoulos S, Mayer CE, Rudolf F, Stelling J. Accurate cell segmentation in microscopy images using membrane patterns. In Bioinformatics 2014;30(18):2644-51. doi:10.1093/bioinformatics/btu302.

Farjam R, Soltanian-Zadeh H, Jafari-Khouzani K, Zoroofi RA. An image analysis approach for automatic malignancy determination of prostate pathological images. In Cytometry Part B: Clinical Cytometry 2007;72B(4):227-40. doi:10.1002/cyto.b.20162.

He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770-8. doi:10.1109/CVPR.2016.90.

Kraus OZ, Ba JL, Frey BJ. Classifying and segmenting microscopy images with deep multiple instance learning. In Bioinformatics 2016;32(12):i52-9. doi:10.1093/bioinformatics/btw252.

Li G, Sanchez V, Nagaraj P, Khan S, Rajpoot N. A novel multitarget tracking algorithm for myosin vi protein molecules on actin filaments in tirfm sequences. In Journal of Microscopy 2015;260(3):312-25. doi:10.1111/jmi.12299.

Manivannan S, Li W, Zhang J, Trucco E, McKenna SJ. Structure prediction for gland segmentation with hand-crafted and deep convolutional features. In IEEE Transactions on Medical Imaging 2018;37(1):210-21. doi:10.1109/TMI.2017.2750210.

Raza SEA, Cheung L, Epstein D, Pelengaris S, Khan M, Rajpoot NM. Mimonet: A multi-input multi-output convolutional neural network for cell segmentation in fluorescence microscopy images. In 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017). p. 337-40. doi:10.1109/ISBI.2017.7950532.

Raza SEA, Cheung L, Epstein D, Pelengaris S, Khan M, Rajpoot NM. MIMO-Net: Gland Segmentation Using Multi-Input-Multi-Output Convolutional Neural Network. In Cham: Springer International Publishing 2017b. p. 698-706. doi:10.1007/978-3-319-60964-5_61.

Raza SEA, Cheung L, Shaban M, Graham S, Epistein D, Pelengaris S, Khan M, Rajpoot NM. Micro-Net: A unified model for segmentation of various objects in microscopy images. In Medical Image Analysis (Feb. 2019), pp. 160–173. ISSN:1361-8415. doi:10.1016/j.media.2018.12.003.

Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention. 2015. p. 234-41. doi:10.1007/978-3-319-24574-4_28.

Sadanandan SK, Ranefall P, Le Guyader S, Wahlby C. Automated training of deep convolutional neural networks for cell segmentation. In Scientific reports 2017;7(1):7860. doi:10.1038/s41598-017-07599-6.

Shelhamer E, Long J, Darrell T. Fully convolutional networks for semantic segmentation. In IEEE Transactions on Pattern Analysis and Machine Intelligence 2017;39(4):640-51. doi:10.1109/TPAMI.2016.2572683.

Veta M, Van Diest PJ, Kornegoor R, Huisman A, Viergever MA, Pluim JP. Automatic Nuclei Segmentation in H&E Stained Breast Cancer Histopathology Images. In PLoS ONE 2013;8(7);e70221. doi:10.1371/journal.pone.0070221.

Wu HS, Xu R, Harpaz N, Burstein D, Gil J. Segmentation of intestinal gland images with iterative region growing. In Journal of Microscopy 2005;220(3):190-204. doi:10.1111/j.1365-2818.2005.01531.x.

Yang X, Li H, & Zhou X. Nuclei segmentation using marker-controlled watershed, tracking using mean-shift, and kalman filter in time-lapse microsopy. In IEEE Transactions on Circuits and Systems I: Regular Papers 2006;53(11):2405-14. doi:10.1109/TCSI.2006.884469.

A Appendix

Our code implementation and the dataset used are also part of the submission. They can be referred for fine details not covered in this report due to space constraints.