SEMANTIC SEGMENTATION OF CELLS FROM CLINICAL IMAGES USING DEEP LEARNING

Mikkel Lehmann (S153048), Sameer Agarwal (S192274), Nicklas Lund (S145129)

Project 9: Segmentation of cell images
Supervised by: Peter Jensen, CTO of Cellari.io

ABSTRACT

As part of the course 02456 Deep Learning at DTU, this project has set out to build and evaluate two deep learning models with supporting functions and architecture for semantic segmentation of cells from clinical images. The two models implemented were (1) U-Net (2015) and (2) Mask R-CNN (2018). Significant support functions include a data pre-processing pipeline and a memory-efficient batch generator. The dataset used is from the *GlaS@MICCAI'2015: Gland Segmentation Challenge Contest* and evaluation was done using the official challenge evaluation metrics and scoreboard. Only limited tuning and performance improving work was done for the two models. Despite this, our DTU-UNET was found to perform similar to peers of its time, finishing at a 7th-place out of 12 peers, while the DTU-MASKRCNN was found to outclass all peers, finishing at 1st-place out of the 12, with a clear win in each of the 6 performance metrics. The findings show that the performance of semantic segmentation algorithms has significantly improved from 2015 to 2018. Based on this, it can be assumed that future work might see new segmentation algorithms implemented in many real world settings in just a few years from now, as robustness and performance is closing in on a level where the algorithm can work in concert with humans, and maybe even work autonomously.

GitHub link: `https://github.com/s153048/02456DL_Finalproject`

1. INTRODUCTION

1.1. Introduction

From basic biological questions and diagnosis to clinical trials, cell morphology derived from microscopic images of the cells provides key quantitative feedback that helps in diagnosing various diseases (especially cancer) and drives further research. Unfortunately, this is still done manually and thus can be time intensive as well as prone to human error. The task is made even more difficult due to overlapping cells and poor imaging quality among other factors [1].

Henceforth, there has been an ever increasing demand for effective computational strategies to analyze large-scale, image-based data. To this end, computer vision approaches have been applied to cell segmentation and feature extraction, whereas machine-learning approaches have been developed to aid in phenotypic classification and clustering of data acquired from biological images [2]. However, what seems to have really been a game changer is the recent advent and application of deep learning approaches, especially convolutional neural networks (CNNs) in image processing. [3]

One such task is the accurate segmentation of gland cell images to obtain reliable morphological statistics for cancer research. In this paper a deep learning approach comparing two state of the art (at that time) CNN architectures – U-Net [4] (2015) and Mask R-CNN [5] (2018) is demonstrated to automate this process of segmenting cells in a gland image.

In summary, the following tasks has been performed using the PyTorch library and the code has been made available on GitHub `https://github.com/s153048/02456DL_Finalproject`.

- Build a memory efficient batch generator from scratch
- Build a data augmenter to expand the training set to a given size and make the model more robust
- Implemented CNN architectures viz. U-Net and Mask-RCNN for solving the pixel-wise image segmentation problem
- Benchmarked and compared the results with other implementations against the same problem across different measurement metrics

2. DATASET

The dataset used is from the Gland Segmentation Challenge Contest, Munich, 2015, [6]. The challenge concerns gland segmentation in histology images. The data has images of Hematoxylin and Eosin (H and E) stained slides, consisting of a variety of histologic grades. The images are of sizes (width, height): (574×433), (589×453), (775×522), $(567 \times$

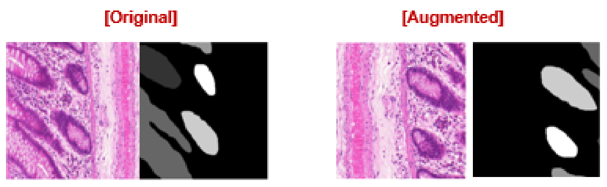


Fig. 1. Above is an example of an original cell-image and of an augmented image which involves a vertical flip, a horizontal flip as well as random cropping

430), (578×433) and (581×442). The dataset contains three subsets, one set of training data (85 images) and two sets (A and B) of test data, 60 and 20 images, respectively. Along with the original images ground truth segmentation masks annotated by expert pathologists are provided. An example can be seen in figure 1. Originally, the data has been used in the detection of Colorectal cancer.

3. PRE-PROCESSING

Since the dataset has been provided as part of a challenge, data pre-processing in terms of cleansing, formatting etc. are minimal. However, *data augmentation* is used to make the model more robust and invariant to the change in location and shapes of cells as well as also to expand the dataset as we only have 85 training samples. A composite pipeline of vertical flip, horizontal flip and random cropping, each with a probability of 50 percent is created for augmentation. An example of an augmented image and its corresponding mask can be seen in figure 1. It is necessary for the U-Net that the input images are of fixed size and divisible by four, hence all images are rescaled to a size of (512×512). Rescaling is performed such that the original aspect ratio is kept by using aliasing while rescaling. For the Mask R-CNN a fixed sized input is not necessary, the varying input size is handled internally in the framework implementation.

4. BATCH GENERATOR

Working with image data, efficient use of memory becomes a challenge. Designing an efficient batch generator for the U-Net became significantly important to be able to effectively train the model on a limited amount of compute. The batch generator is designed such that it only loads the images needed during training. E.g. if the batch size is four, four images are loaded and stored in the batch one at a time until the batch is full (the same goes for the corresponding masks needed for evaluation). At any other time only image paths are stored in order to get access to the images. However, the memory efficiency comes at a cost of computational runtime, we have a memory vs. speed trade-off. Every time an image is needed, the image has to be loaded as a tensor and this operation may be performed thousands of times during training. As the amount of memory is limited, we will have to settle with a bit slower computational times. While training the batch generator randomly picks a pre-specified amount of samples from the pre-specified training data. It generates a random index, calls the data loader class, and adds the image to the current batch. This proceeds until the batch is full, the batch then undergoes a full iteration, the batch is cleared and the process runs until the number of desired epochs is reached. This keeps memory uses at a minimum and thus leaves free memory for saving weight updates during back propagation during training.

5. MODEL 1: U-NET

5.1. Introduction

The U-Net model was created in 2015 by Ronneberger et. al. to solve biomedical image segmentation tasks [7]. The model was designed to learn more efficiently from fewer samples of data, and to run faster, than current models of its time. The inventors succeeded, and the model proved itself as state of the art after winning a number of competitions in 2015 [4].

We chose to implement this model to solve the segmentation task for three reasons: (1) Because the GlaS dataset is small, (2) Because we have limited compute and (3) Because it was invented at the same time as the models competing in the GlaS@MICCAI'2015 challenge [6]. Thus, it is expected to both be a good performer, and serve as a good baseline to which we can compare more recent models.

5.2. Architecture

The U-Net architecture consists of two parts. A contracting layer where the image is down-sampled, and an expanding layer where the image is up-sampled. These parts are denoted "Encoder" and "Decoder" on figure 2.

In the encoder, the model takes a $(512, 512, 3)$ (width, height, channels) input which is then converted through 4 *horizontal convolutions*. A horizontal convolution consists of two zero-padded (to preserve image size) 3×3 2D convolutions, each followed by a ReLU and then a 2×2 max-pooling. Each move doubles the amount of feature channels (starting at 64) and halves the dimensions (starting at 512). At the bottom layer, two more 3×3 2D convolutions, each followed by a ReLU is conducted, resulting in the $(512, 512, 3)$ image now having size $(32, 32, 1024)$.

In the decoder, the model takes the new $(32, 32, 1024)$ image and converts it through 4 *reverse horizontal convolutions*. A reverse horizontal convolution consists of a 2×2 2D convolutional transpose, followed by a concatenation of the features from the same level of the encoding part, and finally two zero-padded 3×3 2D convolutions, each followed by ReLU. Each move halves the amount of feature channels (starting at

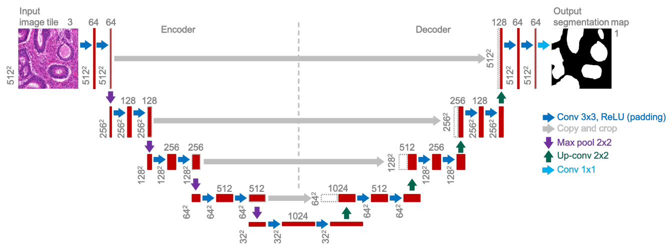


Fig. 2. U-Net architecture for (512 × 512) input with 3 channels ((32 × 32) in the lowest resolution). Each red box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. Grey-dotted boxes represent copied feature maps. The arrows denote the different operations as specified on the right.

1024) and doubles the dimensions (starting at 32). At the final layer, a single 3×3 2D convolution and a ReLU converts the $(512, 512, 64)$ image to a $(512, 512, 1)$ mask using Sigmoid, where each pixel is a binary prediction of background, (0-0.5), or cell (0.5-1).

The models works as follows. In the encoder, at each level, a number of features are learned according to the level of abstraction the current resolution allows. This lets the model encode both specific cell-oriented features, as well as more high-level general image-related features. Reaching the decoder part, the model then starts to up-sample the features to a higher resolution space for accurate pixel-wise predictions. The up-convolution alone retains features, but in a low resolution. By concatenating the layers from the encoder, the resolution increases and the pixel-wise predictions thus becomes better.

5.3. Tuning and training

The models hyper-parameters were tuned using a three-step approach. Firstly, a broad random search was initiated for 50 combinations at 20 epochs each. Secondly, a more narrow random search was initiated, selecting the most promising space from the first search for another 50 combinations at 20 epochs each. The most promising hyper-parameter combination was then chosen and the model was trained for 900 epochs overnight. This resulted in a reduction in validation loss from 0.69 to 0.42. Thirdly, the model was then loaded with its optimizer reset, and then tuned using babysitting, similarly to how one would tune a pre-trained model in transfer-learning [8]. The most promising hyper-parameters were chosen, and the model was trained for 50 epochs. This allowed the model to overcome the local minima at which it had become stuck, and further reduce its validation loss from 0.42 to 0.29.

The best performing network used normalisation of images but were otherwise kept as described in figure 2 with no dropout layers or similar. The hyperparameters chosen were, for the first training: SGD with learning rate 0.1 and L2-regularization 0.00001 with batch size 6. For the second training, the hyperparameters were: Adam with learning rate 0.00001 and L2-regularization 0.001 with batch size 6. The criterion used was BCELoss.

6. MODEL 2: MASK-RCNN

6.1. Introduction

Mask R-CNN, [5], was developed by Kaiming He et. al from the Facebook AI Research group in 2018 to solve the task of object instance segmentation - the task of pixel-wise classification (semantic segmentation) of objects of different classes while distinguishing between multiple instance of objects (object detection) of the same class. Mask R-CNN is one of the state-of-the-art models available, that is why we choose it for the segmentation task. As the dataset only contain one class of objects, namely cells, we expect it to perform well and out-perform the U-net.

Mask R-CNN is an extension to the framework Faster R-CNN, [9], which was developed by Shaoqing Ren, Kaiming He et. al. in 2016 to solve the task of object detection. Faster R-CNN, in turn, builds on Fast-RCNN, [10], which builds on R-CNN, [11]. Mask R-CNN is different from Faster R-CNN in three major ways: 1) by introduction of a Feature Pyramid Network (FPN), 2) by replacing RoIPool with RoIAlign, and 3) addition of a *mask branch* in order to predict an object mask, thus able to perform object instance segmentation.

6.2. Architecture

On a high level, Mask R-CNN works as follows: it takes an image as input, it then locates relevant regions in the images where it contain objects and based on these it predicts the object class, object location (*boundary box*) and a binary (segmentation) mask of the object. Figure 3 shows a high-level visualization of the Mask R-CNN framework.

On a detailed level, the framework is complex and flexible and thus many variations and versions are available. We will in this section elaborate, to some extend, in detail on the architecture of our implementation.

For clarity, we shall distinguish between two modules in the framework, **1)**, typically referred to as the Region Proposal Network (RPN), and **2)**, typically referred to as the head, the numbers used here matches those seen in figure 3.

The first module, **1)**, is a Region Proposal Network (RPN) which can be further subdivided in to two parts. **1.1)** is referred to as the *convolutonial backbone*. For our implementation we choose a ResNet-50-FPN. I.e. a FPN which uses a ResNet-50, [12], architecture in the bottom-up pathway. Conceptually, the FPN architecture is much like the U-Net architecture. It consists of a bottom-up pathway, a top-down pathway and lateral connections between the two pathways. At each ”level” of the pyramid a feature map is constructed. I.e. at corresponding levels in the two pathways a feature map is constructed by a lateral connection (i.e. element-wise addition) of the two feature maps of each of the pathways. By doing so, information on each level of resolution is preserved and thus it maintains strong semantically features at various resolution scales [13]. This is the main reason for choosing an FPN implementation contrary to the standard Faster R-CNN implementation and it is crucial when an image contains many objects in order to capture the boundaries between objects. The FPN architecture is described in detail in [14]. This feature is important for the task at hand, as we have images with many cells close together and it is importance to detect at clear boundary between cells close to each other.

1.2) is a small fully convolutinal network (FCN) which takes as input the covolutional feature maps of **1.1)**. The network is applied in a *sliding* fashion. It takes as input a $n \times n$ spatial window of a feature map, this window is mapped to a lower-dimensional feature vector, 256-d in our case as we use a Zeiler and Fergus model (ZF) [15], and the lower-dimensional feature then serves as input to two 1×1 convolutional layers, a classifier layer and a regression layer. For each window, k sets of four coordinates are regressed and k objectness scores are predicted. The coordinates indicates the region of the potential objects and the obejctness scores measures the membership to a set of object classes vs. background of the image, respectively. By default $k = 9$, where k is the maximum number of proposals in the window. The k proposals are parameterized relative to k reference boxes which are called *anchors* [9]. These anchors are important as they are *translation-invariant*. This means that if the object in original input image is translated (i.e. its position has changes), the anchors ensures that the same regions are proposed, relative to the images. Furthermore, the anchors leaves the model with less parameters to tune and a lower risk of over-fitting, the method is described in detail in [9].

In essence, module **1)** is designed to propose regions in the image where something interesting is happening, i.e. where the image contains objects.

At this point we have multiple proposed regions of interest (RoI) in our convolutional feature maps. The RoIs need now to be mapped to their respective feature map and serves as input to the second module, **2)**

The second module, **2)**, can be further subdivided in to three parts. The first part, **2.1)** is the RoIAlign layer. The RoIAlign takes as input a feature map and its corresponding proposed RoIs and maps the RoIs to the feature maps after which it extracts fixed size feature vectors from each RoI. The fixed size feature vector serve as further input in the model. As mentioned, one of the main differences of Mask R-CNN to Faster R-CNN is the substitution of RoIPool by RoIAlign. The difference of the two is that RoIPool uses *quantization* (turning a real number to an integer) where RoIAlign does not. In the task of object instance segmentation, using quantization in this mapping process becomes a problem as it yields misalignment of the RoI and the feature map and this is a major problem when performing pixel-wise classification. RoIAlign is described in further detail in [5].

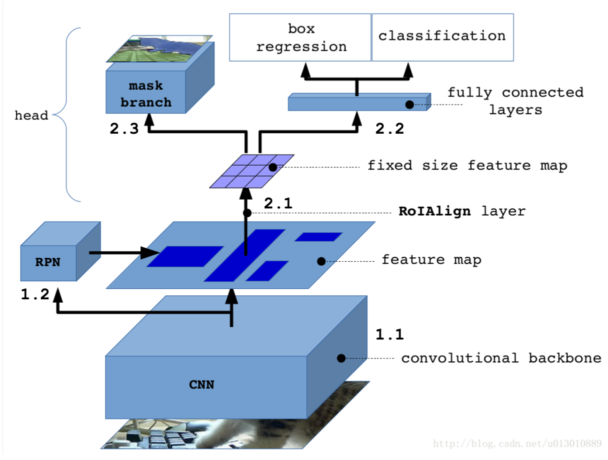


Fig. 3. Visualization [16] of the Mask R-CNN framework

The second part, **2.2)**, performs object classification and *bounding box* regression based on the feature vectors from the RoIAlign layer. Each feature vector is passed through a sequence of fully connected layers and the convoluted features are then passed to two sibling 1×1 fully connected layers, which predicted classification probabilities and bounding box regressions [10].

The third part, **2.3)**, the *mask branch*, is another main difference of Mask R-CNN from Faster R-CNN. It is a small FCN which preforms semantic segmentation, pixel-wise classification, predicting segmentation masks of objects, which operates in parallel to **2.2)**. It takes as input the same feature vectors as **2.3)** and passes it through a sequence of fully connected convolutional layers which for each RoI predicts a binary mask [5].

6.3. Tuning and training

We implemented the Mask R-CNN with a RPN pre-trained on the 2016 COCO dataset [17] and by such utilize the concept of transfer learning. Untrained networks for the classification, **2.2)**, and mask branch, **2.3)**, are initialized and by so only a small amount of training and tuning is needed for the layers in these parts of the network. The hyper-parameters were tuned using a *baby-sitting* approach. Initial parameters values were sat according to [18]. A range of experiments using different configurations of the parameters were then conducted and change in accordance to lowering evaluation loss binary mask predictions. Only the loss of the binary masks were considered during training and evaluation. This is due to the task of the project. The evaluation loss for each binary mask was computed by summing each individual predicted binary mask of an image and ensuring this to be binary by threshold (pixel values $> 0.5 = 1$, otherwise 0), and then computing the Binary Cross Entropy loss of the predicted mask and the true mask of the image. The parameters chosen were: size 256 of the hidden convolutional layer in the mask branch, SDG with momentum, a learning rate of 5e-3 and small L2 regularization of 5e-4. Additionally a small batch size of 2 was used and the model was trained for 500 epochs.

7. POST-PROCESSING

Many post-processing options exists. In the GlaS@MICCAI'2015 challenge, the following post-processing techniques were used by different entrants [19]: (1) Removing small objects, (2) Fill out holes, and (3) disconnect weakly connected objects.

However, in the models implemented here, no performance enhancing post-processing was done. The only work done on the data after model output, was converting the Sigmoid output to binary predictions, $(0, 1)$, as well as converting multi-channel outputs to 1 channel predictions. We chose not to conduct any post-processing as we were interested in seeing the true model performance, without influencing it using our own human judgement. The two models can therefore be seen as *pure*, opposed to the other models compared with in the benchmark in figure 5 which all use some type of post-processing [19].

8. RESULTS

The results of the two models will first be analysed by visual comparison of predictions, and then by benchmarking against the original entries of the GlaS@MICCAI'2015 challenge.

8.0.1. *Visual comparison*

On figure 4 the results from running the U-Net and Mask R-CNN on the test set is compared.

Looking at the best prediction, it can be seen that the U-Net performs very well, however, it does produce some noise, and are quite blurry around the edges[1]. The Mask R-CNN however, provides near perfect results, with little noise and blur around the edges.

Looking at the average prediction, the U-Net performance is unsatisfactory. Many of the cells contains holes, some are not recognized at all, and some cells are predicted that are not really there. Here it becomes clear why the original authors used post-processing to fill in holes. The Mask R-CNN performs very robustly, and it is hard to distinguish the average from the best prediction. There does however seem to be 1 fundamental mistake on average, for instance, recognizing the leftmost cell as 2 instead of 1.

[1]Sharper edges was achieved in the original U-Net with a custom loss-function punishing wrong predictions around the edges harder [7].

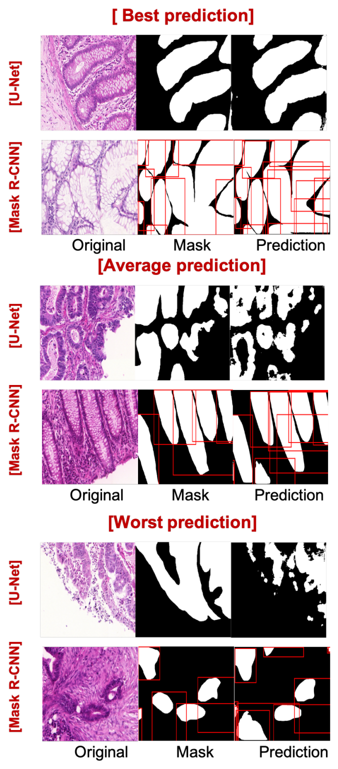


Fig. 4. Sample results from each of the two models (U-Net and Mask R-CNN) compared to the true mask on the test set. For each of the models: The top two rows, [Best prediction], shows the best prediction of the two models from the test set, the middle two rows, [Average prediction], shows an average prediction of the two models from the test set, the bottom two rows, [Worst prediction], shows the worst prediction of the two models from the test set. The loss used is the BCELoss.

Looking at the worst prediction, it can be seen how the U-Net completely fails to correctly recognize the cells, and although it does somehow outline it, the true mask cannot be inferred. The Mask R-CNN however performs quite well, and even though it misses 1 cell and classifies 3 areas as cells that are really background, the prediction looks very human-like when comparing to the original image.

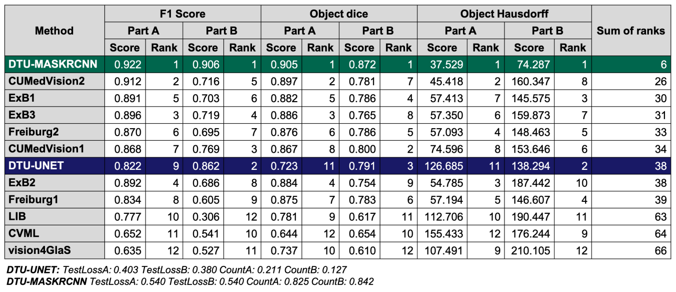
8.0.2. *Benchmarks*

The models are benchmarked by three metrics, each metric evaluated on both test set A and B. The three metrics used are:

The F1 score, used to score *detection*. That is, answering the question "Does the model detect a cell?".

The Object Dice, used to score *segmentation*. That is, answering the question "How well does the model segment the image"

The Object Hausdorff, used to score *shape similarity*. That is, answering the question "How similar are the pre-



Method	F1 Score				Object dice				Object Hausdorff				Sum of ranks
	Part A		Part B		Part A		Part B		Part A		Part B		
	Score	Rank	Score	Rank	Score	Rank	Score	Rank	Score	Rank	Score	Rank	
DTU-MASKRCNN	0.922	1	0.906	1	0.905	1	0.872	1	37.529	1	74.287	1	6
CUMedVision2	0.912	2	0.716	5	0.897	2	0.781	7	45.418	2	160.347	8	26
ExB1	0.891	5	0.703	6	0.882	5	0.786	4	57.413	7	145.575	3	30
ExB3	0.896	3	0.719	4	0.886	3	0.765	8	57.350	6	159.873	7	31
Freiburg2	0.870	6	0.695	7	0.876	6	0.786	5	57.093	4	148.463	5	33
CUMedVision1	0.868	7	0.769	3	0.867	8	0.800	2	74.596	8	153.646	6	34
DTU-UNET	0.822	9	0.862	2	0.723	11	0.791	3	126.685	11	138.294	2	38
ExB2	0.892	4	0.686	8	0.884	4	0.754	9	54.785	3	187.442	10	38
Freiburg1	0.834	8	0.605	9	0.875	7	0.783	6	57.194	5	146.607	4	39
LIB	0.777	10	0.306	12	0.781	9	0.617	11	112.706	10	190.447	11	63
CVML	0.652	11	0.541	10	0.644	12	0.654	10	155.433	12	176.244	9	64
vision4GlaS	0.635	12	0.527	11	0.737	10	0.610	12	107.491	9	210.105	12	66

DTU-UNET: *TestLossA: 0.403 TestLossB: 0.380 CountA: 0.211 CountB: 0.127*
DTU-MASKRCNN *TestLossA: 0.540 TestLossB: 0.540 CountA: 0.825 CountB: 0.842*

Fig. 5. Benchmarks from the GlaS@MICCAI'2015 challenge, including results from our implementation of the U-Net and the Mask R-CNN model. The models are compared using 3 metrics: F1 Score, Object Dice and Object Hausdorff. Additionally, the BCELoss and the Count loss of the two models have been reported at the bottom of the table. All metrics are evaluated on both test set A and B, and the ranking is based on the performance within each metric for each test set.

dicted shapes to the true mask?".

Additionally, the two models are evaluated by the BCELoss and a count metric, used to score how many cells can be counted on the predicted mask, compared to the true mask. The mathematical definition and further explanation of the metrics can be found at [20] and the implementation at [21]. The results can be seen in figure 5.

The U-Net implementation ranks 7 out of 12, performing near-best on each metric on test set B, and near-worst on each metric on test set A. As the results are worse than expected on test set A, it may be possible to achieve better rankings with more tuning, pre-processing or post-processing.

The Mask R-CNN implementation ranks a clear 1 out of 12, finishing on rank 1 across all metrics. It is clear, that just a few years of research can do wonders for models such as these. However, considering that little tuning was done on the Mask R-CNN, no post-processing was done, and more pre-processing could be done, it is expected that even better performance can be achieved with this model.

Additionally, the Mask R-CNN clearly outperforms U-Net on the count metric, getting an average score of 0.834 compared to 0.169. Note that the BCELoss is higher for the Mask R-CNN than the U-Net, this is because we binarize the output before calculating the loss, thus, penalizing wrong predictions harder than the U-Net.

9. FUTURE WORK

Extending the project further in terms of improvements and more tasks can be done at various stages of the project as follows.

9.1. Dataset and Pre-Modeling

In terms of the dataset, to further increase robustness, data augmentations in the form of cell elasticity and cell plasticity can be implemented. It will be interesting to see if they increase the score for U-Net as these are elastic distortions which change the shape and size of the cell. Using a model which is pre-trained exclusively on cell images rather than the COCO dataset might also improve the performance of Mask R-CNN architecture.

9.2. Modeling

As mentioned in [4], a loss function where the pixels on the edges are penalized more than the other can be used for better training. In the case of Mask R-CNN, different convolutional backbone networks can be explored to better detect the regions of interest for the specific task at hand. Also, better hyperparameter tuning techniques like Bayesian and Hyperband optimization can be tried as well [22].

Since, the dataset also has the classification of cell types as malignant or benign for cancer, the model can extended to predict these as well.

10. CONCLUSION

The Mask R-CNN clearly outperforms the U-Net and the other peers of its time. Developed in 2017, just two years after the U-Net and peers of 2015, the Mask R-CNN is a testimony to the speed of which development in the deep learning area is happening [5] [7] [19]. The Mask R-CNN is believed to outperform due to two key reasons: (1) Because of the pre-trained CNN backbone, and (2) Because of the RPN and RoIAlign layer. Although we estimate that it still does not beat human performance[2], the model is much more robust and creates much better and more sensible results than previous models. As such, it is believed that the Mask R-CNN may at least be able to serve as assistant to laboratory workers, for instance by supplying first predictions which the human simply needs to check and fix slightly before approving. Furthermore, assuming that development will continue following the pace seen here, we believe that we should be able to see completely autonomous models taking over, either fully or with only limited oversight, some tasks of laboratory work or similar in the near future.

11. REFERENCES

[1] Moritz Böhland Ralf Mikut Tim Scherr, Katharina Löffler, "Cell segmentation and tracking using cnn-based distance predictions and a graph-based matching strategy," 2020.

[2]Based on inference from table 1 from [7].

[2] Nil Sahin Oren Z Kraus Quaid Morris Charles Boone Brenda J Andrews Ben T Grys, Dara S Lo, "Machine learning and computer vision approaches for phenotypic profiling," 2016.

[3] James Le, "The 5 computer vision techniques that will change how you see the world," 2020.

[4] Department of Computer Science University of Freiburg, "U-net wins two challenges at isbi 2015," 2015.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, "Mask r-cnn," 2018.

[6] Department of Computer Science University of Warwick, "Glas@miccai'2015: Gland segmentation challenge contest," 2015.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[8] Wikipedia, "Transfer learning," 2020.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[10] Ross Girshick, "Fast r-cnn," 2015.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," 2015.

[13] Xiang Zhang, "Simple understanding of mask rcnn," 2018.

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie, "Feature pyramid networks for object detection," 2017.

[15] Matthew D Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," 2013.

[16] "Understand mask-rcnn," 2018.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár, "Microsoft coco: Common objects in context," 2015.

[18] PyTorch, "Torchvision object detection finetuning tutorial," 2017.

[19] Korsuk Sirinukunwattana, Josien P. W. Pluim, Hao Chen, Xiaojuan Qi, Pheng-Ann Heng, Yun Bo Guo, Li Yang Wang, Bogdan J. Matuszewski, Elia Bruni, Urko Sanchez, Anton Böhm, Olaf Ronneberger, Bassem Ben Cheikh, Daniel Racoceanu, Philipp Kainz, Michael Pfeiffer, Martin Urschler, David R. J. Snead, and Nasir M. Rajpoot, "Gland segmentation in colon histology images: The glas challenge contest," 2016.

[20] Department of Computer Science University of Warwick, "Glas@miccai'2015: Gland segmentation challenge contest evaluation metrics," 2015.

[21] Peter Jensen, "Github: Imagemetrics," 2020.

[22] Dan Malowany at Allegro AI, "Accelerate your hyperparameter optimization with pytorch's ecosystem tools," 2020.