Nucleus Segmentation using Deep Learning

Kevin Raji Cherian, Ananth Ravi Narayan, Felipe Rocha

August 2, 2018

1 Introduction

Even though this is a group project, each group member explored different approaches, yet at the end we combined multiple approaches (e.g. using Ensemble). This was, each member was able to work on and try to improve some part of a Deep Learning model, as this is the goal of the proposed project. That being said, the *Related Work* and *Our Contributions* sections are shared among our reports, and each subsection of them has its author indicated on its title.

2 Related Work

2.1 U-Net

U-Net is a deep convolutional neural network architecture, specifically developed for biomedical image segmentation [1].

Different from most convolutional networks, it does not work as a classifier. Instead, it is designed to identify and segment objects on the input image by outputting a mask, where each element indicates the class that the corresponding pixel on the input belongs to (i.e. actual object of interest or background). The structure of U-Net, which I describe below, makes it not only fast but also particularly well suited to image segmentation.

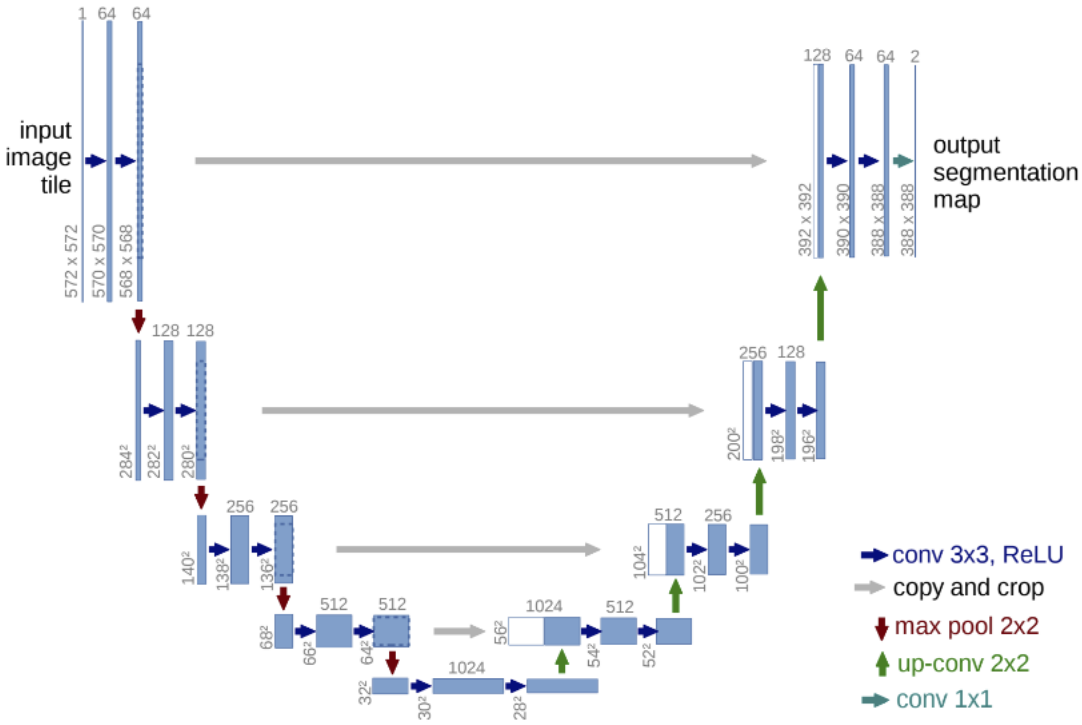

Figure 1: UNet [1]

2.2 Mask-RCNN

Faster R-CNN [2] was a state of the art model for object detection and Mask R-CNN [3] extends this body of work by adding an extra branch to the network to do pixel level image segmentation. To do this, the newly added branch as can be seen in the below figure predicts an object mask in parallel with the existing classification branch. To modify Faster R-CNN, the authors changed the Region Of Interest Pooling layer (ROI-Pool) to Region of Interest

Align (ROI-Align) layer, the reason for this being that pixel level segmentation requires more precision and perfect alignment which ROI-Align offers.

The new branch that is added to the network and through which each ROI is fed, is a fully connected network that is built to take in as input a RoI and output a binary mask (pixel wise). Therefore, now as output of the network, we get candidate object bounding boxes from the Region Proposal Network of the Faster R-CNN, we get class predictions using the features extracted using RoI, and we get a binary mask for each RoI.

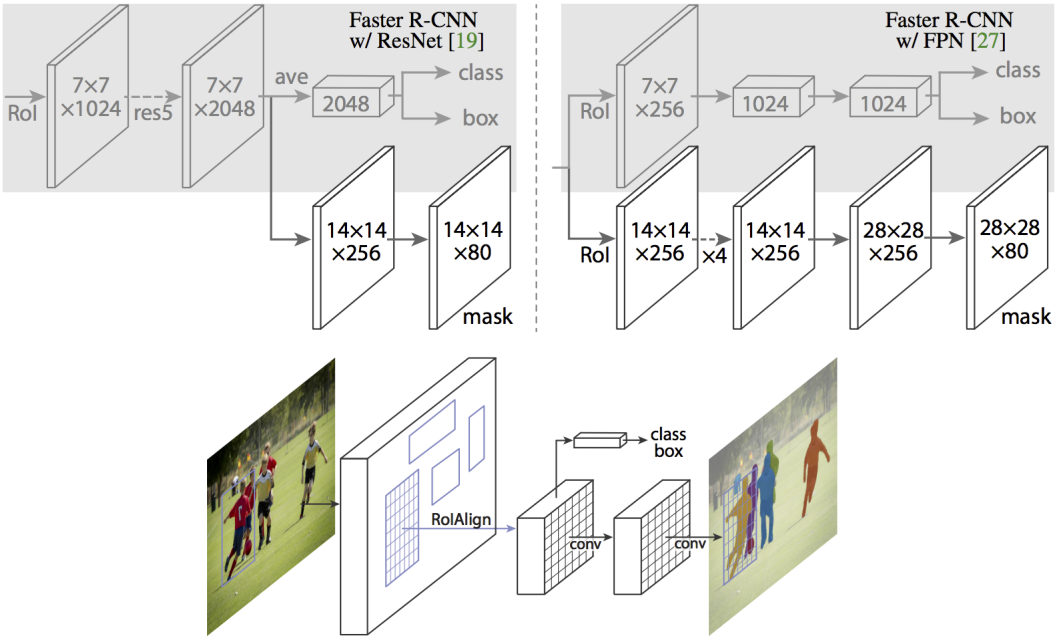

Figure 2: Mask R-CNN [3]

3 Our Contributions

3.1 Weighted U-Net (flt2107)

3.1.1 Preprocessing

For the weighted U-net model, given the diverse dataset that was provided (i.e. different imaging techniques, contrasts, colors), we built a preprocessing pipeline to standardize every image.

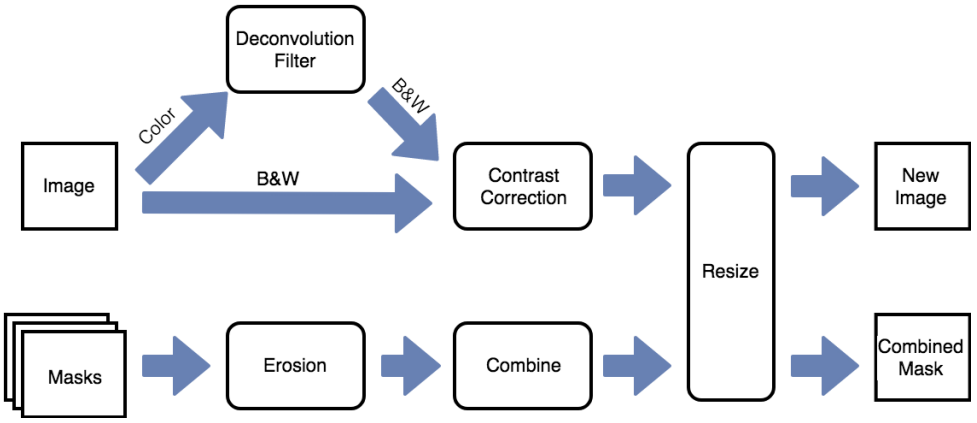

Figure 3: Preprocessing Pipeline Diagram

The pipeline (shown in figure 3) if composed of the following steps:

- **Deconvolution Filter**: This step is executed only for color images. Such filter converts a RGB image into the Haematoxylin-Eosin-DAB colorspace. On fluorescent imaging technique, each one of these compounds emits a different color, and such filter splits their brightness intensities accordingly. Then, empirically, we chose to consider only the Eosin channel, which, extracted alone, is a grayscale image itself;

- **Contrast Correction**: every image reaches this step in a grayscale form. Since several samples have low contrast range, we apply a intensity equalization by linearly "stretching" the intensity histogram, followed by gamma normalization;

- **Mask Erosion & Combination**: since each object in a sample has its own image, we combine all objects for a sample into a single mask. However, since there is some overlapping, before combining them, we first apply binary erosion to each object mask individually, which shrinks it. As a result, we guarantee that all boundaries are present in the combined mask (see figure 4)

- **Resizing**: This step is applied equally to every pair of sample image and its corresponding combined mask, since the resulting pair must match. It consists on resizing every sample into the same fixed size, so they can be fed in batch to our model later. If the original image is larger than such size, we simply scale it down. However, if the original image is smaller, scaling up would lead to blurred images, so instead we mirror the image to reach the desired size.

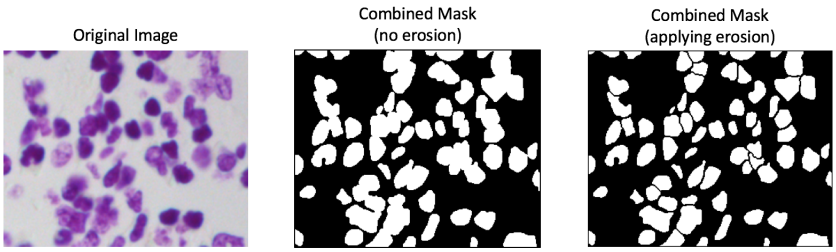

Figure 4: Difference between combinations of raw and eroded object masks

3.1.2 Weight Map Calculation

A core part of the U-net architecture, as proposed on its paper [1], is the weight map, i.e. pixel-wise loss weights that are precomputed for each training sample, as below:

$$w(\mathbf{x}) = w_c(\mathbf{x}) + w_0 \exp\left(- \frac{(d_1(\mathbf{x}) + d_2(\mathbf{x}))^2}{2\sigma^2} \right)$$

For each $\mathbf{x}$ (the coordinates of a pixel), the final weight is sum of two terms:

- the first one, w_c, is the class weight, responsible for addressing class imbalance when training (most of the pixels in the dataset are background pixels);
- the second one increases exponentially as the pixel is close to two different objects (d_1 and d_2 are the distance to the two nearest objects). This way, we force the network to learn such boundaries, as mis-classification of such "boundary" pixels would heavily penalize the loss function.

Lastly, w_0 and σ are hyperparameters. As an example, the weight map of a sample is shown in figure 5.

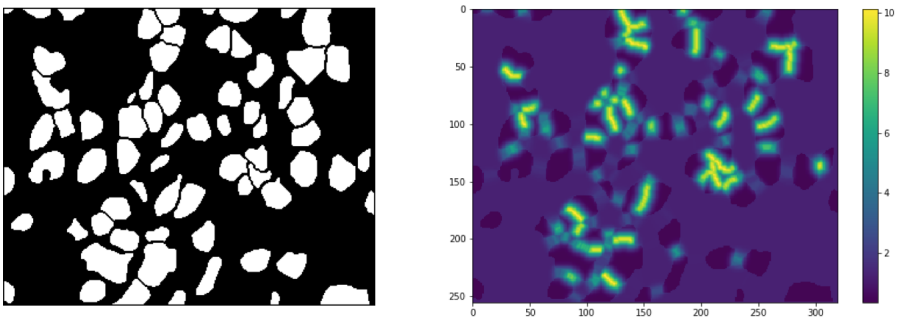

Figure 5: Sample mask and its corresponding weight map

3.1.3 Model Implementation

The preprocessing pipeline was completely designed and implemented by us, using operations from the Scikit-Image package for Python. Since all U-Net implementations that we

found online do not include the weight maps, we implemented the calculation of weight maps on our own. Therefore, our implementation of Weighted U-Net strictly follows the one described in the paper.

The model was trained for a fixed amount of epochs first, and then fine-tuned using early stopping based on the IoU metric, which will be discussed later in this report.

3.2 Ensembling Models (#ar3792, #kr2741)

We thought about the problem at length and realized that while these multiple architectures are performing well on the training and validation set, they tend to overfit because of their inherently powerful nature and large number of hidden neurons. We therefore hypothesized that ensembling two models together could improve performance on the unseen test set.

Our basic motivation behind this was to harness the independence between two weak learners and thereby, reduce error by averaging outputs of both learners. In this body of work, we have implemented both homogeneous and heterogeneous ensembles which will we discussed further in the sections below.

Our idea was simple, combine the outputs of two architectures in some smart way. We followed two basic methods for two combine the outputs. They are listed below.

- The outputs of the two architectures are passed through a final convolutional layer, trained on the ground truth outputs.
- The outputs of the two architectures are passed through a simple averaging layer, which doesn't require training at all.

3.2.1 Ensemble U-Nets (SMU-Net and Savg-Net)

Our first method was to ensemble two similar, but slightly different versions of U-Net. We stuck to our basic principle, that U-Net tends to overfit on the the training data. In order to mitigate the consequeunces of this overfitting, we decided to combine U-Net with a simpler version of itself, which we call Small-U-Net.

The Small-U-Net has half the number of neurons in each Convolutional Layer and also has a smaller number of Convolutional Layer. We theorized that this simplification would result in less overfitting to the training data. We also wanted to retain some of the complexity of the original U-Net.

We ensembled both version using the two different methods we mentioned above to give us two different architectures. They are listed below.

- SMU-Net: This Net is the result of combining the outputs of U-Net and Small-U-Net in a Convolutional Layer (Figure 4).

- SAvg-Net (pronounced 'Savage-Net'): This Net is the result of combining the outputs of U-Net and Small-U-Net using a Simple Average (Hence the name SAvg).

The results for the different models are described in the Results section below.

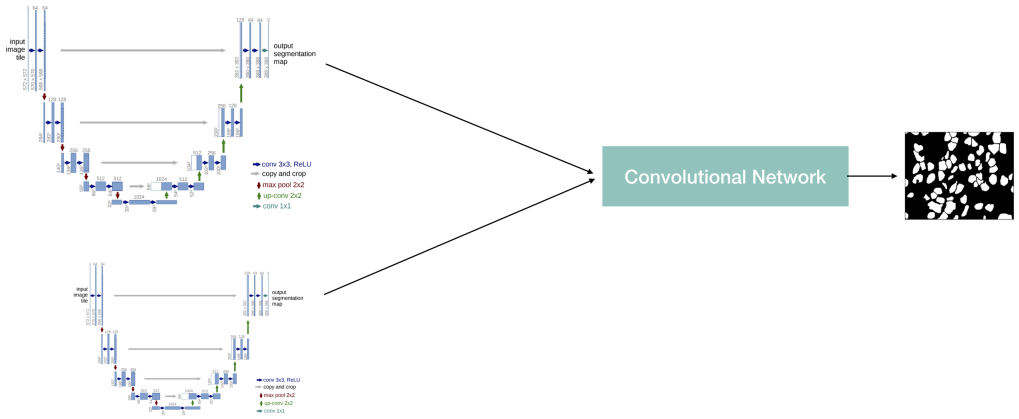

Figure 6: SMU-Net

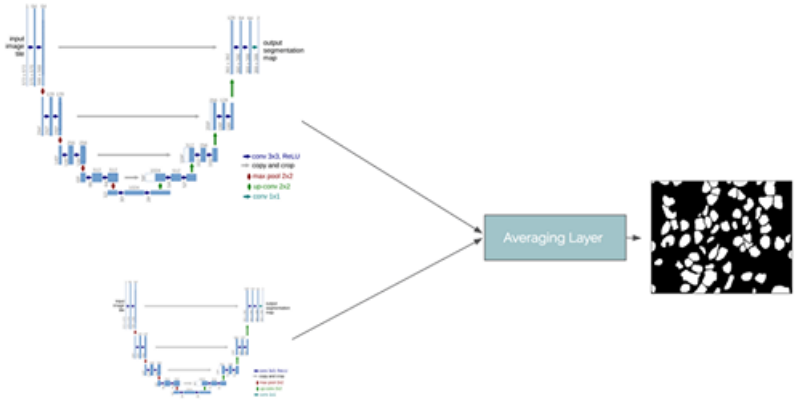

Figure 7: SAvg-Net

3.2.2 Ensemble U-Net and Mask RCNN

Our second method involved ensembling two relatively weak learners, i.e. the Simple U-Net and the Mask R-CNN. To do this, we combined the output of the Mask R-CNN and constructed one single mask by a simple logical or across all output masks. Then, in contrast to what's done in the homogeneous ensemble, we used this mask and the mask outputted by the U-Net as 2 channels of an input to an ensemble network. Our architecture can be seen below :

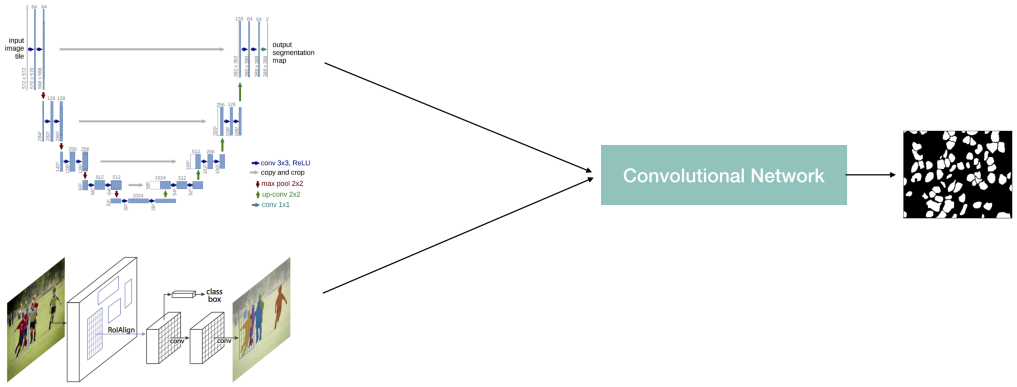

Figure 8: μ-Net

3.3 Data Augmentation (#kr2741)

The fundamental problem with our dataset is the lack of training examples. In order to combat this, we make use of an image augmentation library that allows us to generate new training examples by augmenting the existing ones. This can be done through simple transformations like flipping the cell image and its corresponding masks, increasing exposure/brightness or a combination of several techniques. A few examples of how the augmented images and their masks look are illustrated in the figure below.

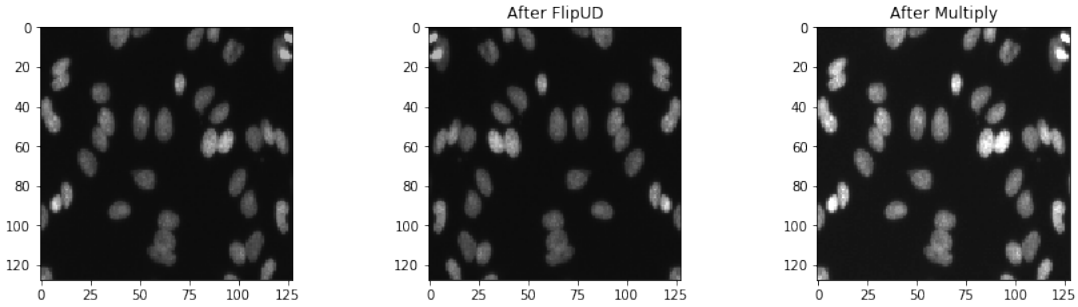

Figure 9: Augmented Input Images

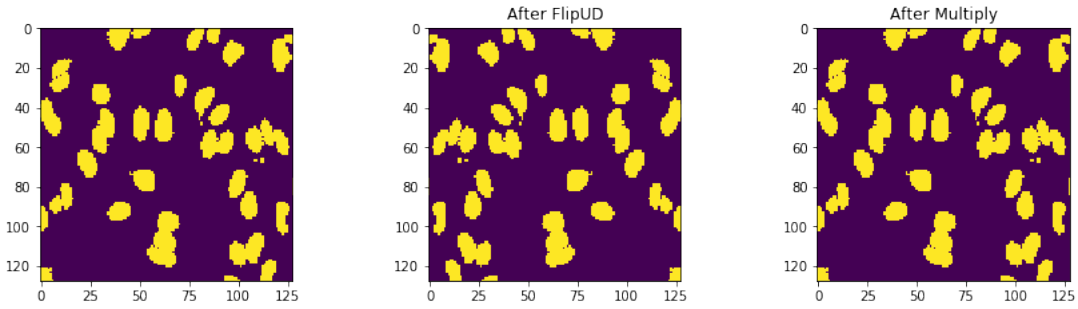

Figure 10: Corresponding Augmented Masks

4 Results

4.1 Evaluation Metric

We followed the evaluation metric described on the Data Science Bowl 2018 challenge on Kaggle. We evaluate our models based on the Mean Average Precison at different Intersection over Union (IoU) thresholds. The IoU of a proposed set of object pixels and a set of true object pixels is calculated as:

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

The metric sweeps over a range of IoU thresholds, at each point calculating an average precision value. The threshold values range from 0.5 to 0.95 with a step size of 0.05. For example, at a threshold of 0.5, a predicted object is considered a "hit" if its intersection over union with a ground truth object is greater than 0.5.

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

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

Finally, the Mean Average Precision (MAP) is calculated as a mean over these thresholds.

$$MAP = \frac{1}{|thresholds|} \sum_{t} Prec(t)$$

Discussion of Results

Caveats

To note, when we computed AP results using Mask RCNN's in-built function, we received results of 0.29 on the Mask R-CNN model. The reason for the discrepancy between our IOU metric and the inbuilt one in Mask RCNN module is the fact that masks can overlap with

Method	IOU Metric
Simple U-Net	0.358
Weighted U-Net	0.417
SMU-Net	**0.427**
Savg-Net	0.401
Mask R-CNN	0.211
μ-Net	0.340
Mask R-CNN + U-Net (Avg)	0.330

Table 1: Results

each other, and for our evaluation metric, this results in an inability to accurately compute the number of objects, hence resulting in a discrepancy in the IOU metric. For the sake of comparability and consistency, we have reported the results using our IOU metric function.

5 Conclusions

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[2] Ross Girshick. "Fast r-cnn". In: *arXiv preprint arXiv:1504.08083* (2015).

[3] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.