Edge Detection Using U-Net Architecture

Onur Tepencelik
PID: A53313861
otepence@eng.ucsd.edu

Ismail Ocak
PID: A53305067
iocak@eng.ucsd.edu

Abstract

Edge detection is one of the most traditional and fundamental tasks, and is seen as the predecessor of the most widely investigated problems in computer vision. Among many approaches, deep learning based ones have shown great success in recent years and became state-of-the-art. We propose a new deep learning based approach to the edge detection task, using an existing and well-known architecture, the U-Net [17]. Although the U-Net architecture has shown a lot of success in semantic segmentation and labeling tasks, especially in the field of bio-medicine, it has not been adopted before for the traditional edge detection task. We train and optimize the U-Net architecture for the edge detection problem. On the BSDS500 benchmark, we obtain the fixed contour threshold (ODS) F-score as 0.703, per image best threshold (OIS) F-score as 0.708 and mean average precision (mAP) as 0.584. We show that the U-Net architecture has a promising performance on this problem, and might be able to stay competitive with other deep learning based methods after some optimization.

1. Introduction

Edge detection is one of the most traditional and fundamental tasks in computer vision. It is seen as the predecessor of the most widely investigated problems in computer vision nowadays, such as semantic segmentation, object detection and object recognition. In this paper, we present our own take on edge detection, using a well-known architecture, the U-Net [17].

Being one of the most historic tasks in computer vision, edge detection has a very rich pool of previous attempts, pioneered by the early developed Canny [3] and Sobel [8] edge detectors. These famous works are followed by systems based on information theory, such as [15] and [1]. Lately, learning based systems such as [4] and [5] took over the edge detection task and became the state-of-the-art at the time. Following the developments in learning based approaches and the rise of the Convolutional Neural Networks (CNN), deep learning based approaches such as [2] and [21] started to be adopted in edge detection tasks and became the current state-of-the-art.

In this paper, we present a somewhat new approach to the edge detection task. We use the well-known U-Net architecture to address the problem. The U-Net architecture is mainly targeting end-to-end image segmentation tasks, developed specifically for biomedical images. Hence, it should be suitable for the edge detection task, by its nature. Our contribution is to experiment with the U-Net architecture on the BSDS500 [1] dataset, a benchmark for the edge detection problem. To the best of our knowledge, the U-Net architecture has not been trained on the BSDS500 dataset and published before.

The paper is organized as follows. *Section 2* presents the history of edge detection and investigates the previous works that address the problem in great detail. *Section 3* presents the U-Net architecture and how it can be used for the edge detection task as well as the training procedure, pre-processing and post-processing stages. The experiments and results are shown and discussed in *Section 4*. The paper is concluded by *Section 5*.

2. Related Work

In this section, we present some of the previous works among many important and respectable ones, on the edge detection task.

Throughout the history of computer vision, many different approaches were adopted for the edge detection task, which is objectively one of the most fundamental tasks of the area. Chronologically, these approaches can roughly be divided into three categories; early pioneers, learning based methods that use hand crafted features, and deep learning based methods.

Earlier methods, considered as the pioneers of the edge detection research, mainly focused on exploiting the variations in intensity and color throughout an image to detect edges. Marr [14] introduced his famous edge detection theory, based on the zero-crossings of the Laplacian of a two dimensional Gaussian function, computed on the color channels of an image. Marr's Theory was followed by the Sobel operator or the Sobel filter [8], which computes the

gradient map of an image and thresholds it to obtain edges. A development on the Sobel operator was introduced by the Canny edge detector [3], which uses Gaussian blurring as a preprocessing step and uses upper and lower thresholds to obtain edges. The idea of using Gaussian blurring became very popular in later approaches as well, as it equips the systems with robustness to noise. These methods were successful pioneers of the area overall, however their performances are not enough to compete with today's approaches.

After the appearance of the feature engineering concept, people started using manually created features for the edge detection problem, based on low level cues such as texture, intensity, color and gradient. [4] showed that it is possible to employ learning on such features and classify pixels as whether they are edge or not. [9] formulated the edge detection problem in terms of statistical inference, and proposed one of the first data driven approaches of the area. Other kinds of features have been introduced such as Pb [15], which was later developed to gPb [1]; combining brightness, color and texture information in different ways. Sketch tokens [11] represented the transition from hand crafted features that uses low level cues, to features that uses mid level cues of an image. Structured forests [5] became the state-of-the-art until the rise of deep learning, by exploiting color and gradient information coming from local image patches. All above approaches were successful in terms of developing onto each other's results, starting from the pioneer methods, however they still can't cope with deep learning architectures, due to the limitations of hand crafted features in representing high level semantic information.

The rise of CNNs and deep learning have lead to many different approaches to edge detection problem. One of the first approaches that included the use of CNNs in an edge detection task was [6], which combines CNNs with nearest neighbor search. DenseNet [7] proposed extracting feature vectors from each pixel and using an SVM classifier to classify them. HED [21] was introduced as an efficient and accurate edge detector, by using the famous VGG16 [18] network as its baseline to perform image to image predictions. Some little improvements were made to HED by using relaxed deep supervision [13]. Lately, [12] proposed combining all the convolutional layer outputs in an efficient way to obtain predictions, so that they obtain a richer set of features, instead of just using the last convolutional layer.

Deep learning based methods easily became the state-of-the-art in the edge detection problem. Our method based on the U-Net architecture is promising because it resembles the latest state-of-the-art methods in different ways, besides the usage of CNNs. First, the U-Net architecture is able to perform image to image training and predictions, just like HED [21]. Moreover, the U-Net partially adopts the idea of using all convolutional layer outputs presented in [12], by concatenating the features coming from the contracting path while performing upsampling in the expansive path. The details of our architecture are presented in more detail in the following sections.

3. Methodology

In this section, we demonstrate the architecture that we use in detail and investigate why it is suitable for our task, as well as the training procedure.

Image recognition tasks can be divided into two main categories. The first category includes the tasks that aim to associate each input image with a class label, such as digit classification or object classification. The second category consists of tasks that aim to associate each pixel of an image with a class label. such as object recognition and edge detection, where each pixel of the image is labeled as edge or not edge. Deep learning, especially after the rise of CNNs, have successfully conquered such tasks, by using various suitable architectures based on the specific problem. As edge detection falls into the second category, the U-Net architecture [17] provides an opportunity with its end to end segmentation capabilities, in other words, pixel-wise classification by design.

3.1. U-Net Architecture

The U-Net architecture is specifically targeting the task of end to end image segmentation on microscopic biomedical images. The main objective is to separate out the pixels of an image that correspond to biologic cells, from the background pixels. For that purpose, the network is specifically tuned to be able to distinguish overlapping or touching cells from each other. The U-Net architecture became the state-of-the-art of its research area, winning the EM segmentation challenge at ISBI 2012 by a large margin [17]. Our idea is that the U-Net architecture can be adopted for the generic edge detection task, since edge detection is just another form of end to end image segmentation.

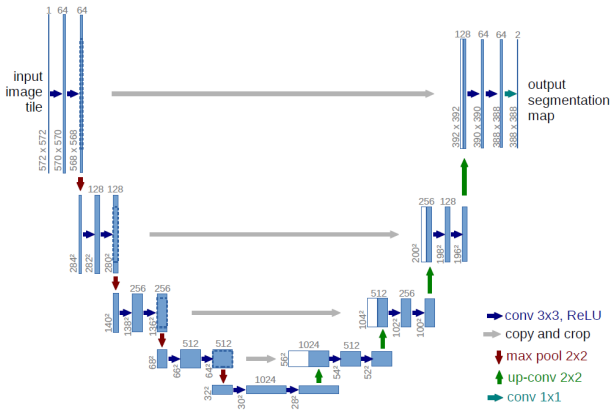

Figure 1. U-Net Architecture

The network architecture is illustrated in Figure 1. It

consists of two parts, the "contracting path" (left portion) and an "expansive path" (right portion) [17]. The contracting path follows the architecture of a generic CNN. It consists of three sets of two convolution operations, each followed by ReLU activation functions and a 2x2 max pooling operation for downsampling. The contracting path can be seen as an encoder that captures the contextual information. After downsampling the image four times while increasing feature channels in each pooling operation, the expansive path starts to upsample the image back to its original size. It consists of three sets of two "up-convolution" operations, followed by ReLU activation functions and an upsampling opereation. The upsampling operation halves the number of features, and concatenates them with the other half coming from the corresponding part of the contracting path. The expansive path can be seen as a decoder that enables precise localization using upsampling. As a result, U-Net can be seen as a convolutional neural network that does not have fully connected layers [10].

3.2. Training

We train our model on the BSDS500 dataset and benchmark [1]. The dataset consists of 200 train, 100 validation and 200 test images. Dimensions of the input images are either (321x481x3) or (481x321x3). Provided ground truths have the same dimensions, respective of the size of the input image. We use these images after applying some pre-processing operations; such as cropping, transposing, majority vote for ground truths and data augmentation, explained in more detail in later sections of the paper.

Data manipulations, training and prediction tasks were conducted on a Python environment. We use Pytorch library and GPU for the training part. Python implementation using Pytorch was inspired from Usuyama's GitHub repository [20] which was designed for creating segmentation masks for synthetic images using the U-Net architecture.

In training phase, instead of using a single loss metric, we use a combination of Binary Cross-Entropy Loss and Dice Loss [19]. Edge detection is a task where positive instances (edge pixels) are less frequent compared to the negative instances. As a result of this, using only the traditional Cross-Entropy Loss might be problematic. Thus, we incorporate the Dice Loss to our loss function as well. Using the combination of different loss metrics is a common practice in Image Processing area. [16] Lastly, since the output of the U-Net architecture are logit values we use a Sigmoid transformation before calculating the losses at each step. The loss metric formula is given as:

$$Loss = 0.5 * \text{mean}(\text{CE}(p, \hat{p})) + 0.5 * \text{DL}(p, \hat{p})$$

where:

$$\text{CE}(p, \hat{p}) = -\left(p \log(\hat{p}) + (1-p) \log(1-\hat{p})\right)$$

$$\text{DL}(p, \hat{p}) = 1 - \frac{2 \sum p_{h,w} \hat{p}_{h,w}}{\sum p_{h,w} + \sum \hat{p}_{h,w}}$$

Note that, in this representation, CE returns a matrix output where DL returns a scalar. While calculating the final loss, we use the mean of the loss matrix which was returned by CE and the output of DL.

We train our U-Net model using 40 epochs. At each epoch, the model was trained using 1600 training images and validated using 800 validation images. At the end of each epoch, we calculate the loss and save the model if the loss metric was lower than the previous

In order to find the best parameters for our model setup, we use an extensive grid search experiment using different parameters. The parameters we use in the grid search, along with the optimal parameters that give the smalles validation loss are below:

Parameter	**Candidates**	**Winner**
optimizer	['adam', 'rmsprop', 'sgd']	'adam'
learning rate	[1e-3, 1e-4, 1e-5]	1e-4
lr decrease steps	[10, 20, 40]	10
lr decrease rate	[0.5, 0.1]	0.1
batch size	[1, 4, 8]	4

Table 1. Grid Search Results

Here, 'optimizer' is the optimization algorithm that is used in weight and learning rate update, 'learning rate' stands for the amount that the weights are updated after each epoch, 'lr decrease steps' stands for the number of epochs we need to wait before updating the learning rate, 'lr decrease rate' represents the update rate of the learning rate, and 'batch size' stands for the number of images used in training at each iteration.

After carefully tuning the parameters, the best validation loss was found as 0.416876. Training and validation losses vs epoch curve for our best model is below:

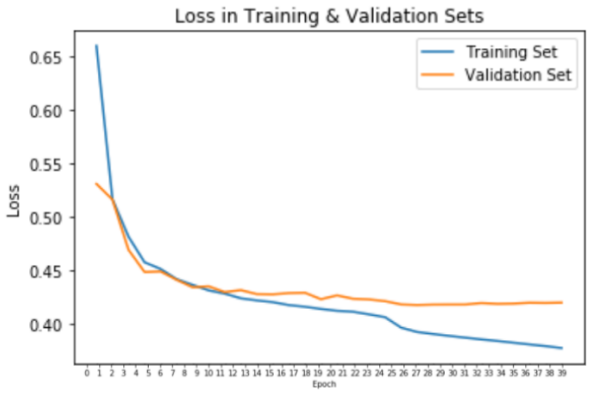

Figure 2. Loss vs Epoch

It can be seen that after approximately 20 epochs, the validation loss stops decreasing and training loss continues to decrease.

4. Experiments

In this section, we discuss the conducted experiments in order to refine the results of our model, as well as its performance on one of the edge detection benchmarks.

4.1. BSDS500 Dataset

We train and evaluate our model on Berkeley Segmentation Dataset and Benchmark (BSDS500) [1] which contains 200 training, 100 validation and 200 testing images. Each image is labeled by multiple different ground truth annotators. The BSDS500 dataset, on which most of the previous works have been evaluated, has been one of the most important benchmarks on the edge detection task. We evaluate our model on the test set using three metrics provided with the benchmark, fixed contour threshold (ODS), per image best threshold (OIS) and average precision (AP).

4.2. Data Pre-processing

The BSDS500 [1] dataset contains colored images which come from different contextual backgrounds. The images are not specific to a certain subject or area. As a result of this fact, we did not apply blurring or denoising methods to our images. Based on our experiments with these pre-processing methods, the outcomes of such methods and their contribution to our edge-detection task are heavily instance dependent.

The original BSDS500 [1] data has multiple ground truths for a single image. These ground truths come from the hand-drawings of different annotators. The number of annotators per image is not fixed, it may vary from 4 to 9 for each image. Hence, there is a need to come up with an algorithm that decides on a single ground truth for each image. A common practice to make this decision is to use a majority vote based approach. Since the pixel values for a ground truth can either be 0 or 1, the actual ground truth value of a pixel can be decided using a majority vote. For example, if 3 out of 5 annotators labeled a certain pixel as an edge pixel, then that pixel is considered to be an edge pixel on the final ground truth image, based on the majority vote. Ties count as edge as well, since the number of edge pixels are significantly lower than non-edge pixels. For our model, we use a slightly modified version of the majority vote decision rule by adding an additional vote to the edge votes in advance. For example, if 2 out of 5 annotators labeled a certain pixel as an edge pixel, then that pixel is considered to be an edge pixel on the final ground truth image, based on the modified majority vote. Using the modified majority vote decision rule, we managed to get more continuous and stable edges compared to the original majority vote rule. The differences can be seen in Figure 3.

Figure 3. Ground Truths

The dimensions of the images in the dataset are either (481x321x3) or (321x481x3). In order to get consistency among the dataset, we transpose the images whose dimensions are (481x321x3). As a result, all the input images share the same dimensions (321x481x3). The same idea applies to the ground truth images as well. Ground truths that have dimensions (481x321) were transposed to (321x481).

After reaching consistency among image dimensions in the dataset, we also crop images and ground truths by 1 pixel from each dimension, making the input image size of our architecture (320x480). This cropping operation ensures that all the feature maps inside the architecture have even dimensions, so that pooling and upsampling operations can be made consistently.

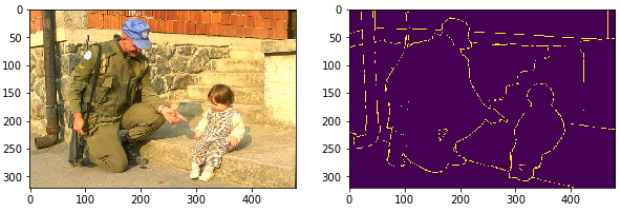

Figure 4. Image and Ground Truth Examples

As the final part of image pre-processing we populate our images using rotations with different angles. For each image in our training and validation sets we create 7 more images using 45° rotations. Using this approach we increase the number of our training instances from 200 to 1600. Image augmentation provided more images to the learning phase of the project and helped the model to be more effective in dealing with rotated images.

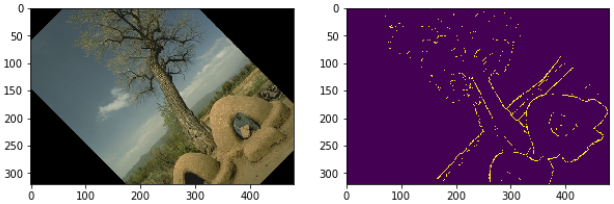

Figure 5. Image and Ground Truth Examples with Rotation

4.3. Data Post-processing

Although the ground truths consist only of 0-1 pixels, due to the nature of the U-net architecture, the prediction output consists of logits, whose values are in between $(-\infty, \infty)$. As a result, in order to create edge predictions, we apply the Sigmoid function to the output of the model and obtain predicted edge probabilities for the pixels.

$$p(y) = \frac{1}{1 + e^{-y}}$$

The benchmark provided with the BSDS500 dataset applies different thresholds to edge probabilities to obtain precision-recall curves.

4.4. Results

After carefully tuning the parameters and finding the best parameters that yielded the lowest validation loss, we tested the model on the 200 test images of the dataset. We evaluate our model using the benchmark provided with the BSDS500 dataset [1]. We obtain the fixed contour threshold (ODS) F-score as 0.703, per image best threshold (OIS) F-score as 0.708 and mean average precision (mAP) as 0.584. The table that presents the results of various methods, taken from [21], is below.

Model	ODS	OIS	mAP
Canny [3]	0.600	0.640	0.580
BEL [4]	0.660	-	-
U-Net [17] (ours)	0.703	0.708	0.584
gPb [1]	0.726	0.757	0.696
Sketch Tokens [11]	0.727	0.746	0.780
SE [5]	0.746	0.767	0.803
N4-Fields [6]	0.753	0.769	0.784
DeepEdge [2]	0.753	0.772	0.807
HED [21]	0.782	0.804	0.833

Table 2. Metrics Comparison

The precision-recall curves for various methods, taken from [21], and for our architecture are below.

Although being better than traditional approaches, compared to other deep learning based state-of-the-art methods, our U-Net architecture lags a little bit behind, in terms of performance metrics. However, we still believe that the performance of the U-Net architecture is open to improvements, and the scores can be increased with more experiments towards optimization. Inspecting the precision-recall curves, we observe that the recall of the U-Net architecture drops down faster compared to other architectures. This presents an opportunity to improve the architecture, by tuning the network in a way to prevent false negatives more aggressively.

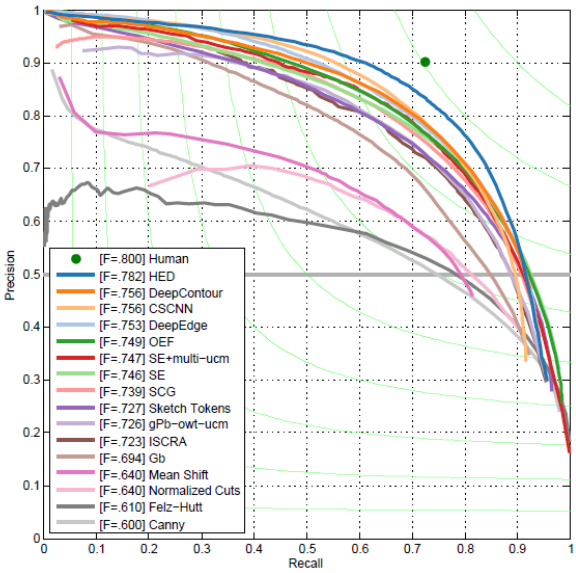

Figure 6. Precision-Recall Curves for Various Methods

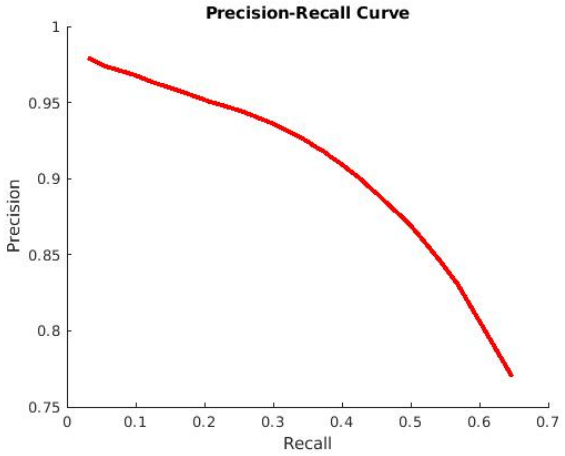

Figure 7. Precision-Recall Curve for U-Net Architecture

Besides the performance metrics, we visually inspect the outputs of our model to understand its capabilities in terms of capturing contextual information. The resulting output edge masks are promising.

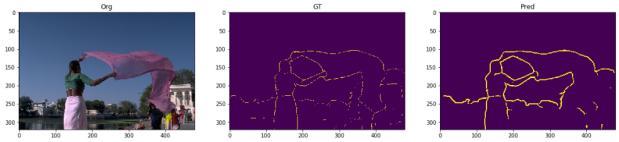

Figure 8. U-Net Prediction Example: Woman

The U-Net architecture is really successful at predicting the edges that occur along sharp color change lines, which shows that the network successfully incorporates zero-crossings to its perception.

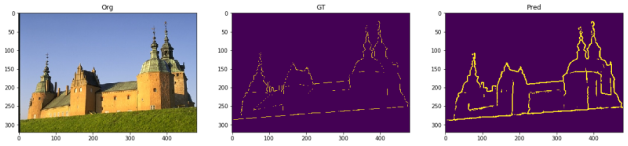

Figure 9. U-Net Prediction Example: Castle

We also compare the performance of the U-Net architecture with other edge detection methods such as Canny [3], Sobel [8] and HED [21]. In general, Canny [3] and Sobel [8] edge detection methods created detailed and unnecessary edges, while HED an U-Net are more successful in terms of capturing contextual information, being learning based methods. Output edge examples using different methods are listed below (Figures 8 to 15):

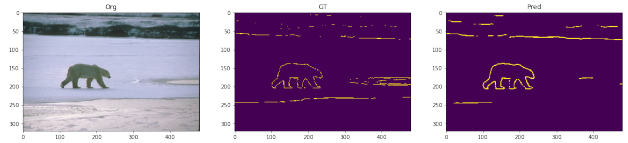

Figure 10. U-Net Prediction Example 1

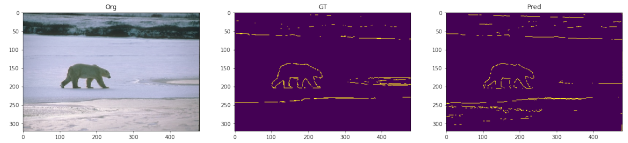

Figure 11. Canny Prediction Example 1

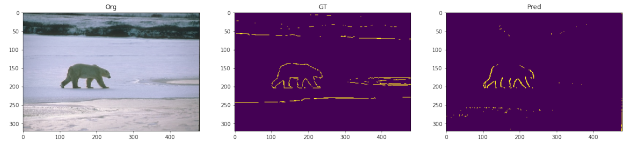

Figure 12. Sobel Prediction Example 1

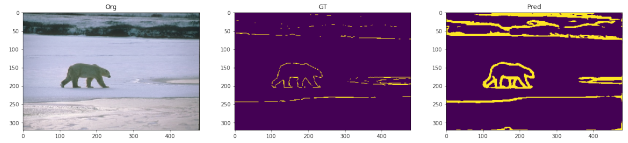

Figure 13. HED Prediction Example 1

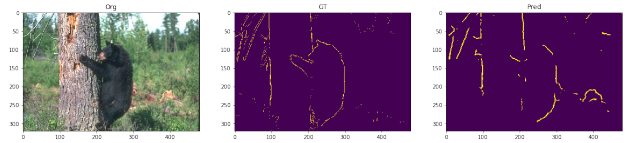

Figure 14. U-Net Prediction Example 2

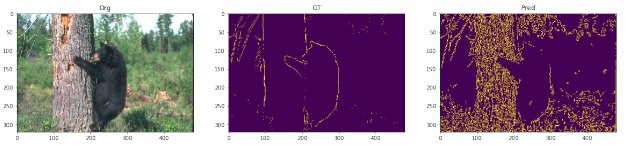

Figure 15. Canny Prediction Example 2

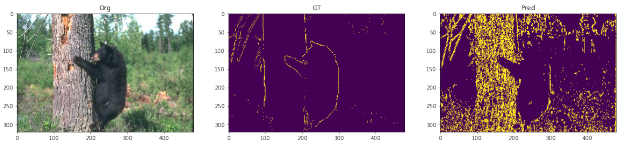

Figure 16. Sobel Prediction Example 2

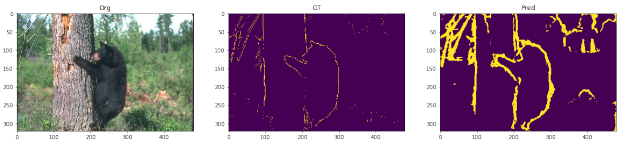

Figure 17. HED Prediction Example 2

In edge detection tasks, the appearance and usability of the final mask is as important as the performance metrics. By looking at the examples, it can be observed that U-Net performs really well when the image does not have blurry parts and the colors around the edges are noticeable. However, U-Net may miss the edges when the image or some parts of the image is blurry. In these types of images, HED is more successful in detecting the edges. This might be because HED generates more thick edges, which creates an advantage in predicting the hard cases where the edges are blurry. In order to thin those edges, many edge detection approaches including HED incorporate non-maximum suppression (NMS) based edge thinning operations in their models. Since U-Net already gives thinner edges compared to HED, it might not require the NMS algorithm, which might save computation time.

5. Conclusion

In this project, we addressed one of the most fundamental tasks in computer vision with a novel approach. We used the U-Net architecture on the edge detection task, on the BSDS500 dataset. Although we didn't come close to state-of-the-art performances, we obtained promising results. The fact that the U-Net architecture is specifically targeting microscopic cell images might have created a few drawbacks for the edge detection task. Yet, the architecture gives promising results which can be improved by tuning the network towards the edge detection task. Edge detection is a class-imbalanced problem and it is not trivial to detect edges with high precision. There exist a lot of important factors that are very impactful on the results. We believe that, in order to achieve good results in edge de-

tection, one must take into account ground truth selection, data pre-processing and hyper-parameter tuning very carefully. We believe that our architecture is still open to improvements and might perform better with more optimization among any of the above issues.

References

[1] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011.

[2] G. Bertasius, J. Shi, and L. Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection, 2014.

[3] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[4] P. Dollar, Zhuowen Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1964–1971, 2006.

[5] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *CoRR*, abs/1406.5549, 2014.

[6] Y. Ganin and V. S. Lempitsky. N^4-fields: Neural network nearest neighbor fields for image transforms. *CoRR*, abs/1406.6558, 2014.

[7] F. N. Iandola, M. W. Moskewicz, S. Karayev, R. B. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *CoRR*, abs/1404.1869, 2014.

[8] N. Kanopoulos, N. Vasanthavada, and R. L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[9] S. Konishi, A. L. Yuille, J. M. Coughlan, and S. C. Zhu. Statistical edge detection: Learning and evaluating edge cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):57–74, 2003.

[10] H. Lamba. Understanding semantic segmentation with unet, Feb 2019.

[11] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3158–3165, 2013.

[12] Y. Liu, M.-M. Cheng, X. Hu, K. Wang, and X. Bai. Richer convolutional features for edge detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[13] Y. Liu and M. S. Lew. Learning relaxed deep supervision for better edge detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 231–240, 2016.

[14] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980.

[15] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549, 2004.

[16] L. Nieradzik. Losses for image segmentation, Sep 2018.

[17] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.

[20] Usuyama. github.com/usuyama/pytorch-unet, Jun 2019.

[21] S. Xie and Z. Tu. Holistically-nested edge detection. *CoRR*, abs/1504.06375, 2015.