Semantic Segmentation using U-Net and Region Adjacency Graphs

Christopher Kolios
CP8307 - Introduction to Computer Vision
Ryerson University
Toronto, Canada
ckolios@ryerson.ca
500743717

Yiannis Varnava
CP8307 - Introduction to Computer Vision
Ryerson University
Toronto, Canada
yvarnava@ryerson.ca
500708182

***Abstract*—Recent approaches to segmentation tasks have been based on deep learning techniques, leaving the question: "Is there still a need for traditional segmentation models?". In this paper, we compare a U-Net with several traditional semantic segmentation methods such as thresholding, Chan-Vese, and Region Adjacency Graphs (RAGs) on "A Large Scale Fish Dataset". The models were evaluated by comparing their intersection-over-union (IoU) values and Dice scores. The U-Net achieved a mean-IoU of 0.89 and Dice score of 0.94 while the RAG method achieved a mean-IoU of 0.72 and Dice score of 0.81, indicating that the U-Net is the superior model. However, some limitations of deep learning-based models such as a dependence on a large amount of data and compute resources for training and inference may influence the model choice for certain applications. The source code for this experiment is available at https://github.com/yvarnava/CP8307-Semantic-Segmentation-Project.**

***Index Terms*—deep learning, segmentation, u-net, region adjacency graph**

I. Introduction

Semantic segmentation is a critical research area of computer vision that focuses on sectioning an image into different classes, pixel by pixel [1]. A class can represent a type of object such as a person, car, or a street sign. Detecting different classes in images or videos is very desirable in applications such as medical imaging, self-driving cars, augmented reality, among many others. A semantic segmentation model will output a segmentation map, where each pixel is assigned to a class. Recent advances in segmentation models have been based on deep learning approaches such as the U-Net [2], Fully Convolutional Network (FCN) [3], and family of Region-based Convolutional Neural Networks (RCNN) [4]. Prior to the rise in popularity of using deep learning for image segmentation, traditional methods such as thresholding, Chan-Vese segmentation [5], and region adjacency graphs [6] were often used.

A. *Convolutional Neural Networks*

Many deep learning-based segmentation models are based on convolutional neural networks (CNNs) [2] [3] [4] [7] because these types of networks are able process image data more efficiently than traditional multilayer perceptrons [8]. The reason for this is because a two-dimensional image is fed into the network as opposed to a one-dimensional vector, allowing the network to localize features. The CNN performs a defined number of spatial convolutions the filters and an image to extract relevant information. There are three types of layers in CNNs; convolutional, pooling, and fully connected layers [1]. Fig. 1 illustrates the basic structure of a CNN.

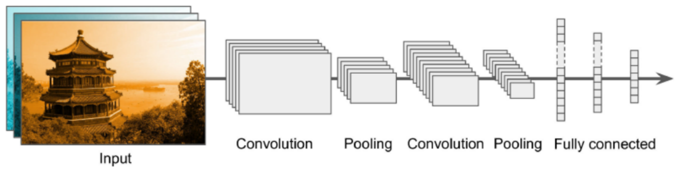

Fig. 1. Basic CNN structure [8].

The convolutional layer is responsible for performing spatial convolutions between the input and each filter. This produces a feature map for each filter. In practice, the hyperparameters for defining a convolutional layer such as Conv2D in TensorFlow include the number of filters, the filter size, the stride, and the padding. Backpropagation is used to train the update the weights of the network, which are the filter values. Depending on the type of padding used, i.e., 'same' (equal padding around all edges), or 'valid' (no padding), the size of the feature maps may change. A nonlinear activation such as ReLU can be applied after each convolutional layer to allow the network to learn nonlinear decision boundaries.

The pooling layer applies a statistical function such as max or average to windows of the feature maps to reduce the image dimension and allow the network to detect complex features [8]. In TensorFlow, the MaxPool2D layer can be used to implement this operation. The hyperparameters include the pool size (kernel size), stride, and padding.

The last layer of a CNN depends on the application. For classification problems, a fully connected layer which is a one-dimensional vector is used to output a probability score for each class. For segmentation problems, the final layer of a CNN is a 1×1 convolution, which can be used to determine the number of output channels. For instance, a $1 \times 1 \times 6$ convolutional layer at the end of a CNN architecture will output an image with 6 channels. This topic will be explained further with respect to the U-Net in Section I-B.

B. U-Net

Ronneberger et al. proposed the U-Net for biomedical image segmentation in 2015 [2]. This paper has over 34000 citations and serves as a point of reference for many segmentation algorithms [9] [10] [11]. A visual representation of the network is shown in Fig. 2, where the left half of the network represents the encoder, the bottom layer represents the bottleneck, and the right half represents the decoder. This section will outline basic architectural details of the U-Net. Specific implementation details will be outlined in Section II-A.

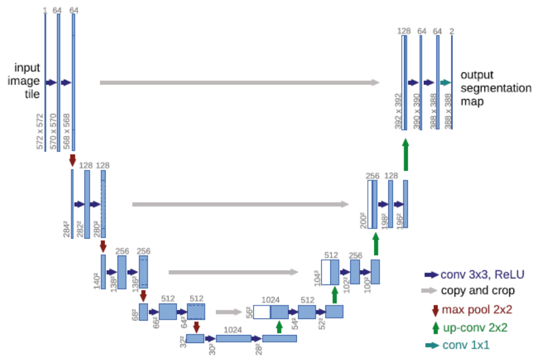

Fig. 2. U-Net architecture [2].

The encoder and decoder both have four levels. Each level of the encoder consists two consecutive convolution-ReLU pairs and a max pooling layer. At each level, the number of filters double. The bottleneck consists of two convolutional layers and a ReLU activation, with double the number of filters from the previous stage. This allows the network to learn the most important features of the input image, since the image is the most compressed at this stage (it has been downsampled by a factor of two, four times). Each level of the decoder consists of upsampling and upconvolution, which decrease the number of feature maps by a factor of two, followed by concatenation with the output of the corresponding level from the encoder side, and another two convolution-ReLU pairs.

The softmax activation defined in (1) is used to associate each pixel in the output with the class that has the highest probability. This is used because the softmax activation function ensures that the output values (probabilities) are between 0 and 1, and sum to 1. Each variable in (1) is described as follows: σ represents the sigmoid function, x_i represents the i^{th} training example contained in input x, K represents the number of classes, and j represents a loop variable used iterate through the classes.

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{1}$$

C. Thresholding

It is important to note that there exists a subsection of the semantic segmentation problem known as binary semantic segmentation. They are similar, but with binary semantic segmentation there are only two possible group classifications: 0 or 1. One of the most simple approaches to binary segmentation is thresholding, in which the key idea is that some threshold is picked, and pixels above that threshold are classified as foreground (1), and the rest as background (0). This threshold can be set manually or automatically. Thresholding is a core concept in computer vision.

D. Chan-Vese

Chan-Vese segmentation is a segmentation algorithm that was first described in 1999 by Tony Chan and Luminita Vese [5]. In Chan-Vese segmentation, the image is first divided into level sets (in some pattern, with a popular method being a checkerboard pattern). Then, with each iteration, an energy minimization problem along the curves in the image and the level sets is formulated, such that the energy is described by intensity differences and curve parameters, which are dependent on shape. By minimizing the energy along contours, the algorithm is able to select significant borders which are not wholly dependent on the image gradient, and instead partly on the shape and length of the contours in the image. With each further iteration, the level sets are further optimized to key regions which are segmentable [12].

E. Region Adjacency Graphs

Region Adjacency Graphs (RAGs) are simple graphs that images can be formulated in terms of, where the regions in the image correspond to the nodes, and the regions that are adjacent share an edge. Each node is an image section, and each edge corresponds to a shared border between two regions. The edges can be weighted in a variety of different ways, but one useful way to define edges is to weigh them according to the difference of mean colour between the two regions [6]. In the context of semantic segmentation, as there exist different regions of the image, they can all be assigned to some class. Additionally, RAGs have a useful property in which regions with similar colours can be easily merged. By setting a threshold, then merging nodes with edge weights less than that threshold (i.e. with a small mean colour difference), these regions can be combined efficiently. This merging procedure is relatively efficient, only needing to iterate over all edges in a given RAG. While an RAG approach is great for segmentation purposes, it offers nothing in terms of classification by itself.

F. Dataset

"A Large Scale Fish Dataset" was used to train the U-Net and validate all segmentation models implemented for this experiment [13]. The dataset consists of 9000 images of 9 classes of fish with their corresponding binary segmentation masks. The scikit-learn library in Python was used to generate an 80-20 training/testing split to partition the dataset. That is, 7200 samples were used for training, and 1800 samples were used for validating the model. The dataset was randomly shuffled prior to being partitioned in order to prevent class-bias, such as training the model on certain classes of fish and testing it on other classes. In this situation, the model would learn how to segment only certain types of fish and fail to segment other types. The dataset included data augmentation, which is a method often used to generate more samples from existing samples by rotating and flipping images [1]. Data augmentation is often used in segmentation problems to improve model performance [2].

G. Evaluation Metrics

The evaluation metrics used for benchmarking segmentation models are the Intersection-over-Union (IoU) and the Dice Score [1].

The IoU, also known as the Jaccard Index, is an evaluation metric used for segmentation models [1]. (2) displays the formula for the IoU, where A represents the predicted mask, B represents the true mask. It takes the ratio of the intersection between the predicted mask and the true mask and divides it by the union of the two masks. Intuitively, it is desired to maximize the intersection and minimize the union. The IoU is typically used to provide a worst-case score.

$$\text{IoU} = J(A,B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid} \tag{2}$$

Another metric often used to evaluate the performance of segmentation models is the Dice score, or the Dice coefficient [1]. (3) displays the formula for the Dice score, where A and B represent the same variables as the IoU. The dice score is typically used to provide an average score, and is often represented by true-positive (TP), false-positive (FP) and false-negative (FN) values.

$$\text{Dice} = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid} = \frac{2TP}{2TP + FP + FN} \tag{3}$$

II. Methods

A. U-Net Implementation

A TensorFlow implementation of the U-Net architecture was used for fish segmentation. Specifically, the Functional API from TensorFlow was used. TensorFlow documentation on image segmentation and functions from the Advanced Computer Vision with TensorFlow course from Coursera were also utilized in this U-Net implementation [14] [15]. The network was trained using the limited, free GPU resources from Google Colaboratory. For this exact reason, it was difficult to make many modifications to the network to fine-tune it to this problem, as continuous use of the platform would result in a lock-out period from the service for an undetermined amount of time.

During the preprocessing stage, the dataset was read in using the TensorFlow API [16]. Both fish images and ground truth masks were read in as JPEG images and resized to 128×128 as the U-Net expects. It was important to consider that the fish images required three channels, as they were colour images, and the ground truth masks only required one channel as they are binary masks. After the images were read in, they were normalized such that the pixel intensity values were between zero and one. Next, the scikit-learn library was used to split the dataset into training and testing set. The training set is composed of 80% of samples while the testing set is composed of the remaining 20% of examples. A specific seed was used to produce repeatable dataset splits if this function was called multiple times. This helped ensure consistency in the testing set used in comparisons of other models.

Our implementation of the U-Net for segmenting fish is as follows. Reference code can be found at the GitHub link in the abstract of this paper. The diagram of the implemented U-Net can be found in Fig. 6 within Appendix VII-A.

First, a function for defining the convolutional unit is defined. Since this component is often used in the network, it was decided to parameterize it such that the number of filters and kernel size can be adjusted if needed. The convolutional unit uses the "he_normal" kernel initializer from TensorFlow to initialize the weights for two consecutive convolutional layers, followed by a ReLU activation layer. The kernel size was fixed to 3x3 with a stride of 2, and "same" padding was used to retain the image size after convolution.

Each level of the encoder portion of the network is defined by an encoder unit, which contains a convolutional unit and a 2×2 max pooling layer. Four consecutive encoder units are used to create the encoder of the network. The difference between each encoder unit is the filter size. As previously mentioned, the filter size doubles at each level of the encoder; i.e., $64, 128, 256, 512$. The bottleneck layer consists of a single convolutional unit with 1024 filters. The input of the bottleneck is the output of the fourth encoder unit. Likewise, the output of the bottleneck feeds into the decoder.

Now that the input image has been encoded (downsampled), it passes through a decoder network to extract features from the image. In other words, each decoder stage will perform an upsampling operation. Each decoder unit is up-convoluted or "deconvoluted" by a 3×3 kernel of stride 2. The decoder units utilize skip connections, which have been proven to increase the performance of the network avoiding the vanishing gradient problem [17]. The skip connections consist of the pooling output from the encoder stage that were saved before. A concatenation of the up-convoluted result with the pooled result are then passed through a convolution unit. Unlike the encoder, the number of filters in the decoder are halved at each level, starting from 512.

The last convolutional layer is a $1 \times 1 \times 2$ convolution which shapes the dimension of the output image to have two classes (background and fish). Finally, the output is fed into a softmax activation function to assign each pixel to a class based on their probability values.

The following parameters were used to train the U-Net:

- GPU: NVIDIA Tesla K80 12GB
- Optimizer: Adam
- Learning Rate: 0.001
- Loss: Sparse Categorical Cross Entropy
- Number of Epochs: 10
- Batch Size: 64
- Buffer Size: 1000

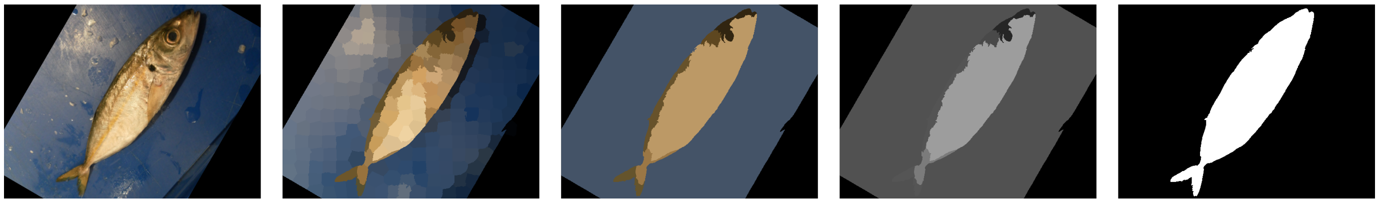

Fig. 3. RAG Segmentation Procedure on a Horse Mackerel. Images show (from left to right) the original image, the image separated into superpixels by colour, the image after RAG merging, the grayscale image, and the final binary segmentation after corner background removal.

The Adam (Adaptive Moment Estimation) optimizer allows for adaptive learning rates when minimizing the sparse categorical cross entropy loss function. This loss function was chosen as the original U-Net paper used a cross entropy loss as well. Due to computation restrictions with Google Colaboratory, the model was only trained for 10 epochs, which took about 30-45 minutes depending on the GPU resources available. After training, the model is saved as a .h5 file with the current date and time appended in the name.

B. Thresholding Implementation

Code for all of the classical segmentation algorithms was programmed in Python 3.8 in Spyder [18], using the matplotlib [19], scikit-image [20], and numpy [21] Python libraries. Reference code can be found at the GitHub link in the abstract of this paper.

Image thresholding was implemented from first principles, as it is a relatively simple algorithm and a custom approach was desired. Both a manual threshold and an automatic threshold approach were coded and tried. Ultimately automatic thresholding was selected as it could be applied to each image independent of some user-specified threshold. The automatic thresholding was also implemented from first principles, from the Wikipedia page on Image Thresholding [22] [23]. The general approach takes an initial threshold, divides the image into two parts, thresholds based on the initial threshold, calculates the mean of the foreground and background, and generates a new threshold by averaging the means. This procedure continues while the difference between the current threshold and the previous threshold is above some user-specified value. This value was experimentally determined, and is visible in the code and GitHub.

C. Chan-Vese Implementation

The Chan-Vese segmentation algorithm was implemented using the Python setup as described in II-B. The scikit-image python library's built-in segmentation.chan_vese function [20] was utilized to implement the Chan-Vese segmentation, on the grayscale version of each image. The parameters can be found in the corresponding code and GitHub. These were manually tuned after some experimentation. After the Chan-Vese segmentation was generated, the image was eroded then the area of the non-fish/background was expanded using scikit-image.morphology's erosion and area_opening methods. This was done to eliminate some of the checkerboard patterns that persisted in a few image cases, to increase the accuracy of the segmentation.

D. RAG Implementation

The RAG implementation was done using the Python setup as described in II-B. The primary package that was used was scikit-image's RAG methods, including graph.rag_mean_color and graph.cut_threshold. The first step of the RAG process was to merge regions that are similar in colour into large superpixels (done using the built-in algorithm in scikit-image). Then, an RAG was constructed, with edges representing the difference in mean colour between the superpixels (the superpixels being the nodes themselves). Then, regions that are similar in mean colour were then combined, by checking the edges and modifying the RAG. In terms of the input parameters to the RAG method, they were determined through experimentation, and are visible in the associated code and GitHub. As RAG thresholding is not in itself a classification algorithm, a custom approach was used to separate and classify the background and the fish, so as to convert the problem to a binary semantic segmentation problem.

It should be noted that due to the rotation done as part of the data augmentation step in the Kaggle database, there would often be corners of the image that were rightfully black (i.e. not part of the image). First, the image was converted to grayscale. This was done for ease of implementation for the next step. Next, using loops each edge of the image was examined by sweeping along each corner (top left to top right, top left to bottom left, for all possible starting corners and directions), and the index of the first non-zero pixel was determined. Then, using a bucket tool (skimage's flood_fill [20]), the regions of the pixels with the value at each starting index were painted to be 0's (to correspond to the background). The reason why this was able to be done is because in a large majority of images the fish was not touching any of the real image corners, so any region encountered would likely be part of the background (and could therefore be safely assigned to 0). All remaining pixels after the omni-directional border sweep and paint were set to 1, to correspond to the fish. This assumption (of a fish not being the first pixel from any of the corners) allowed for the removal of the boundary regions which would otherwise be much more difficult to separate. Fig. 3 shows an example of the process from start to finish.

TABLE I
IOU VALUES FOR SAMPLE IMAGES.

	U-Net	RAG	Chan-Vese	Threshold
Image 1	0.80	0.64	0.53	0.21
Image 2	0.92	0.84	0.23	0.49
Image 3	0.92	0.90	0.35	0.14
Image 4	0.94	0.87	0.26	0.32

TABLE II
DICE SCORES FOR SAMPLE IMAGES.

	U-Net	RAG	Chan-Vese	Threshold
Image 1	0.89	0.78	0.69	0.34
Image 2	0.96	0.91	0.42	0.66
Image 3	0.96	0.94	0.51	0.25
Image 4	0.97	0.93	0.41	0.49

E. Result Evaluation

The test set consisted of 1800 image-mask pairs, representing 20% of the total database. The U-Net was compared to the RAG, Chan-Vese, and thresholding segmentation methods on an initial set of 4 sample images. These 4 images were chosen to show a range of perceived segmentation difficulties. Results were generated in terms of IoU and Dice score. The classical methods that performed well were then tested on the entire test set, and compared with the U-Net.

III. RESULTS

The qualitative IoU and Dice score values for four sample images are presented in Table I and Table II, respectively. The four sample images are displayed in Fig. 4. For the sample images, the U-Net performs consistently well, the RAG segmentation performs well, and Chan-Vese and Thresholding segmentation perform poorly. All segmentation algorithms erroneously classified the top of the table in Image 1 as part of the fish. The U-Net and RAG got a good mask for Images 2, 3, and 4. Neither Chan-Vese nor thresholding generated a visually-good result for any of the sample images. The IoU and Dice scores quantifying these performances are presented as Table 1. They confirm the visual analysis, in that U-Net is better than RAG, which is significantly better than both Chan-Vese and Thresholding, which both perform poorly.

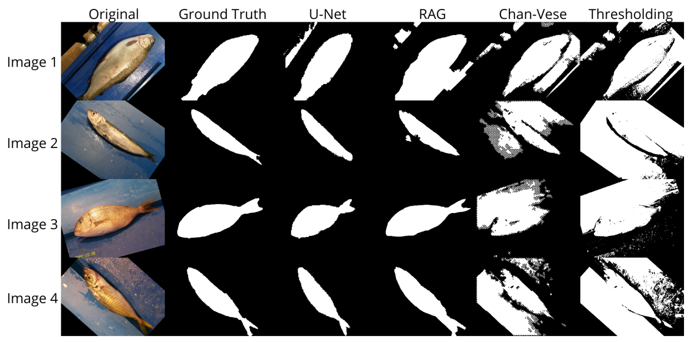

Fig. 4. Experimental results for sample images (visual).

TABLE III
QUANTITATIVE RESULTS FOR TEST SET.

	U-Net	RAG	Chan-Vese	Threshold
IoU	0.89	0.72	N.C.	N.C.
Dice	0.94	0.81	N.C.	N.C.

The results on the full test set for U-Net and RAG are presented as Table III. The U-Net had a mean IoU of 0.89, and a mean Dice score of 0.94. The RAG approach had a mean IoU of 0.72, and a mean Dice score of 0.81. While both algorithms performed well, the U-Net is evidently superior. Results for Chan-Vese and Thresholding were not calculated for the entire test set, as they were deemed to be not promising in preliminary analysis. For the RAG approach, the medians were 0.78 and 0.88 for IoU and Dice score, respectively. The IoU and Dice score result histogram distributions for the RAG implementation are presented as Fig. 5.

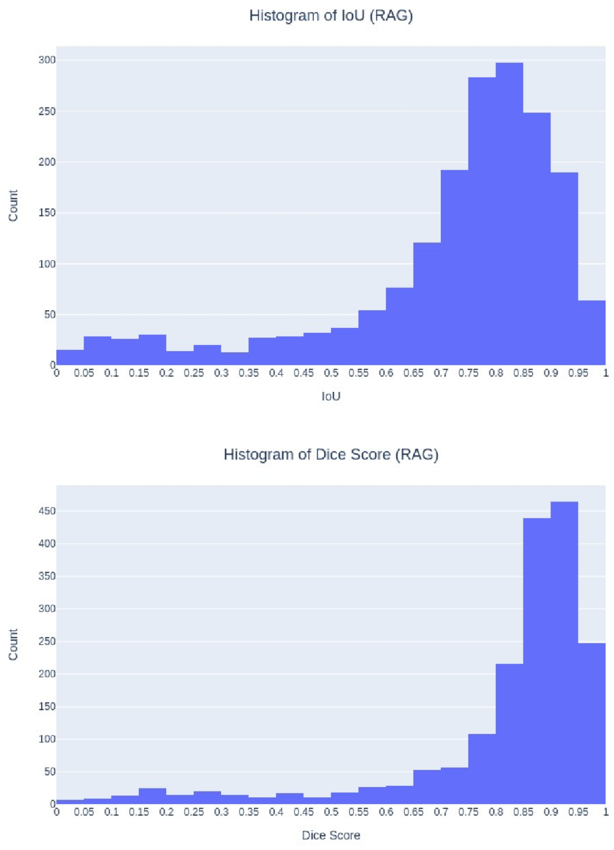

Fig. 5. RAG result histograms.

IV. DISCUSSION

Fig. 4 displays the experimental results for four sample images. It is important to realize that the images were resized to 128×128 for the U-Net segmentation masks, which is why they are a different shape than the traditional methods which are 590×445. The IoU and Dice score calculations for the U-Net were done with respect to the resized ground truth masks.

It can be observed in Fig. 4 that the worst performing sample (Image 1) for all results is the example with an anomaly in the background. That is, there is a table edge in the background that is not present in other samples. This affected the U-Net's ability to segment the fish since other samples do not have this property. Also, the model is not trained to differentiate the table from the fish itself. For the traditional methods, the intensity value of the table is closer to that of the fish than the blue background, tricking the algorithms into believing that it is also part of the fish.

Another aspect of the nature of the dataset that can alter segmentation results are the shadows around the edge of the fish. The lighting was not universal, causing an ambiguity in the outline of the fish. Even to the human eye, it is difficult to distinguish the fish from the table in the shadowed regions. Specifically, the fish tails were a difficult part for all algorithms to segment with precision due to the sharp edges. The average IoU and Dice scores were affected by this because even the ground truth masks were not perfect. A higher quality dataset would resolve these issues.

In terms of the different classical approaches, there was a wide gap between RAG segmentation and the other two approaches. Thresholding was not capable of properly segmenting the fish, as it was too simple of a method. Chan-Vese seemed to introduce artifacts to the image mask, and could not clearly resolve the fish from its background. The RAG method got a good differentiation between background and fish, likely owing to the colour differences between the fish and the background. If the fish was blue, for example, the RAG would have much more difficulty. Additionally, it should be noted that there is a small peak on the left side of the RAG results histogram, as seen in Fig. 5. This explains part of why the median IoU/Dice is much higher than the mean for the RAG approach. The reason why the RAG segmentation approach works so poorly in a small subset of examples is likely due to the boundary sweeping approach for semantic segmentation. The underlying assumption of the fish never being the very corner pixel was flawed, and there were a few cases where the whole fish would be set to background due to this, even though the actual segmentation procedure properly delineated it. However, in a vast majority of cases, the assumption was both valid and useful.

A. Future Work

There are a few aspects of the U-Net implementation that could be improved upon given more time and computational resources such as adding dropout or batch normalization layers to accelerate training and reduce overfitting [24] [25]. Further, an expanded training set containing higher quality examples as well as more training samples would improve the model performance. Finally, utilizing TensorBoard while training the network on the IoU or Dice score losses as opposed to calculating these metrics after the model has been trained could potentially provide more insights about the learning process and lead to further optimizations of the network.

For the classical segmentation methods, there are also a few aspects that could be improved upon in future studies. First, more segmentation methods than just thresholding, Chan-Vese, and RAGs could be implemented. There is a wealth of classical segmentation methods available in the literature, and there is undoubtedly one that is well suited to the fish dataset. Secondly, the RAG algorithm could be improved so that it is more robust to image instances where the fish intersects with the corner of the image. While it generally performed well, by the skew of the results histogram it is clear that there is some room for improvement. Finally, a simply machine learning algorithm (such as Finite Linear Discriminate Analysis, or a Support Vector Machine) could be employed to tune the parameters that were used in the RAG and other classical methods to fit the dataset well. A training-testing split could be utilized, to ensure that overfitting is not taking place, though our problem was framed in terms of fish semantic segmentation.

V. Conclusion

In conclusion, the U-Net offered a better performance than all of the classical methods, with a mean IoU and Dice score of 0.89 and 0.94 respectively, in comparison to the RAG segmentation approach, which had a mean IoU and Dice score of 0.72 and 0.81. The thresholding and Chan-Vese segmentation approaches did not have good results, so the only successful classical approach was the RAG segmentation. The only significant disadvantages of the U-Net approach are the requirement of a large dataset (which, even with data augmentation, may not be available), and the compute time. Ultimately, for semantic segmentation problems where a large amount of data is available, and there is adequate processing power, the U-Net vastly outperforms classical methods.

VI. Author Contributions

CK worked on the implementation, research, and analysis of the classical segmentation techniques. YV worked on the implementation, research, and analysis of the U-Net technique. Both CK and YV worked on the in-class presentation, as well as the research for, writing, and editing of this paper.

References

[1] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, ser. Lecture Notes in Computer Science, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," 2015, pp. 3431–3440. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014, pp. 580–587. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html

[5] T. Chan and L. Vese, "An Active Contour Model without Edges," in *Scale-Space Theories in Computer Vision*, M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 141–151.

[6] A. Tremeau and P. Colantoni, "Regions adjacency graph applied to color image segmentation," *IEEE Transactions on Image Processing*, vol. 9, no. 4, pp. 735–744, Apr. 2000, conference Name: IEEE Transactions on Image Processing.

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, Apr. 2018, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[8] G. Aurélien, *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*. OReilly, 2019.

[9] Z. Zhang, Q. Liu, and Y. Wang, "Road Extraction by Deep Residual U-Net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, May 2018, conference Name: IEEE Geoscience and Remote Sensing Letters.

[10] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "UNet++: A Nested U-Net Architecture for Medical Image Segmentation," *arXiv:1807.10165 [cs, eess, stat]*, Jul. 2018, arXiv: 1807.10165. [Online]. Available: http://arxiv.org/abs/1807.10165

[11] M. Z. Alom, C. Yakopcic, M. Hasan, T. M. Taha, and V. K. Asari, "Recurrent residual U-Net for medical image segmentation," *Journal of Medical Imaging*, vol. 6, no. 1, p. 014006, Mar. 2019, publisher: SPIE. [Online]. Available: https://www.spiedigitallibrary.org/journals/journal-of-medical-imaging/volume-6/issue-1/014006/Recurrent-residual-U-Net-for-medical-image-segmentation/10.1117/1.JMI.6.1.014006.full

[12] P. Getreuer, "Chan-Vese Segmentation," *Image Processing On Line*, vol. 2, pp. 214–224, 2012, https://doi.org/10.5201/ipol.2012.g-cv.

[13] O. Ulucan, D. Karakaya, and M. Turkan, "A Large-Scale Dataset for Fish Segmentation and Classification," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 2020, pp. 1–5.

[14] "Image segmentation | TensorFlow Core." [Online]. Available: https://www.tensorflow.org/tutorials/images/segmentation

[15] "Advanced Computer Vision with TensorFlow." [Online]. Available: https://www.coursera.org/learn/advanced-computer-vision-with-tensorflow

[16] "Image segmentation U-Net." [Online]. Available: https://kaggle.com/dikshabhati2002/image-segmentation-u-net

[17] Y. Zhao, J. Chen, Z. Zhang, and R. Zhang, "BA-Net: Bridge Attention for Deep Convolutional Neural Networks," *arXiv:2112.04150 [cs]*, Dec. 2021, arXiv: 2112.04150. [Online]. Available: http://arxiv.org/abs/2112.04150

[18] "Anaconda software distribution," 2020. [Online]. Available: https://docs.anaconda.com/

[19] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[20] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: https://doi.org/10.7717/peerj.453

[21] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[22] Thresholding (image processing), "Thresholding (image processing) — Wikipedia, the free encyclopedia," 2021, [Online; accessed 16-December-2021]. [Online]. Available: https://en.wikipedia.org/wiki/Thresholding_(image_processing)

[23] S. E. Umbaugh, "Digital image processing and analysis : human and computer vision applications with cviptools / scott e umbaugh." Boca Raton, FL, 2011.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[25] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Mar. 2015, arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167

VII. Appendix

A. U-Net Model Summary

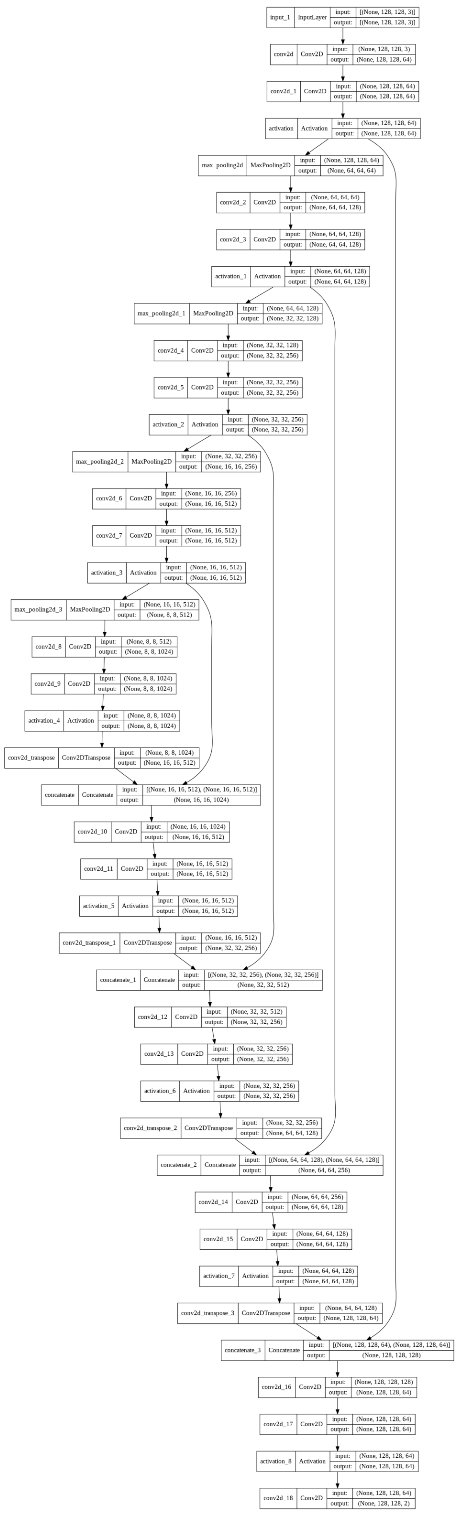
input_1 InputLayer input: [(None, 128, 128, 3)] output: [(None, 128, 128, 3)]
conv2d Conv2D input: (None, 128, 128, 3) output: (None, 128, 128, 64)
conv2d_1 Conv2D input: (None, 128, 128, 64) output: (None, 128, 128, 64)
activation Activation input: (None, 128, 128, 64) output: (None, 128, 128, 64)
max_pooling2d MaxPooling2D input: (None, 128, 128, 64) output: (None, 64, 64, 64)
conv2d_2 Conv2D input: (None, 64, 64, 64) output: (None, 64, 64, 128)
conv2d_3 Conv2D input: (None, 64, 64, 128) output: (None, 64, 64, 128)
activation_1 Activation input: (None, 64, 64, 128) output: (None, 64, 64, 128)
max_pooling2d_1 MaxPooling2D input: (None, 64, 64, 128) output: (None, 32, 32, 128)
conv2d_4 Conv2D input: (None, 32, 32, 128) output: (None, 32, 32, 256)
conv2d_5 Conv2D input: (None, 32, 32, 256) output: (None, 32, 32, 256)
activation_2 Activation input: (None, 32, 32, 256) output: (None, 32, 32, 256)
max_pooling2d_2 MaxPooling2D input: (None, 32, 32, 256) output: (None, 16, 16, 256)
conv2d_6 Conv2D input: (None, 16, 16, 256) output: (None, 16, 16, 512)
conv2d_7 Conv2D input: (None, 16, 16, 512) output: (None, 16, 16, 512)
activation_3 Activation input: (None, 16, 16, 512) output: (None, 16, 16, 512)
max_pooling2d_3 MaxPooling2D input: (None, 16, 16, 512) output: (None, 8, 8, 512)
conv2d_8 Conv2D input: (None, 8, 8, 512) output: (None, 8, 8, 1024)
conv2d_9 Conv2D input: (None, 8, 8, 1024) output: (None, 8, 8, 1024)
activation_4 Activation input: (None, 8, 8, 1024) output: (None, 8, 8, 1024)
conv2d_transpose Conv2DTranspose input: (None, 8, 8, 1024) output: (None, 16, 16, 512)
concatenate Concatenate input: [(None, 16, 16, 512), (None, 16, 16, 512)] output: (None, 16, 16, 1024)
conv2d_10 Conv2D input: (None, 16, 16, 1024) output: (None, 16, 16, 512)
conv2d_11 Conv2D input: (None, 16, 16, 512) output: (None, 16, 16, 512)
activation_5 Activation input: (None, 16, 16, 512) output: (None, 16, 16, 512)
conv2d_transpose_1 Conv2DTranspose input: (None, 16, 16, 512) output: (None, 32, 32, 256)
concatenate_1 Concatenate input: [(None, 32, 32, 256), (None, 32, 32, 256)] output: (None, 32, 32, 512)
conv2d_12 Conv2D input: (None, 32, 32, 512) output: (None, 32, 32, 256)
conv2d_13 Conv2D input: (None, 32, 32, 256) output: (None, 32, 32, 256)
activation_6 Activation input: (None, 32, 32, 256) output: (None, 32, 32, 256)
conv2d_transpose_2 Conv2DTranspose input: (None, 32, 32, 256) output: (None, 64, 64, 128)
concatenate_2 Concatenate input: [(None, 64, 64, 128), (None, 64, 64, 128)] output: (None, 64, 64, 256)
conv2d_14 Conv2D input: (None, 64, 64, 256) output: (None, 64, 64, 128)
conv2d_15 Conv2D input: (None, 64, 64, 128) output: (None, 64, 64, 128)
activation_7 Activation input: (None, 64, 64, 128) output: (None, 64, 64, 128)
conv2d_transpose_3 Conv2DTranspose input: (None, 64, 64, 128) output: (None, 128, 128, 64)
concatenate_3 Concatenate input: [(None, 128, 128, 64), (None, 128, 128, 64)] output: (None, 128, 128, 128)
conv2d_16 Conv2D input: (None, 128, 128, 128) output: (None, 128, 128, 64)
conv2d_17 Conv2D input: (None, 128, 128, 64) output: (None, 128, 128, 64)
activation_8 Activation input: (None, 128, 128, 64) output: (None, 128, 128, 64)
conv2d_18 Conv2D input: (None, 128, 128, 64) output: (None, 128, 128, 2)

Fig. 6. U-Net model summary (TensorFlow).