Instrument Shaft Segmentation using Unsupervised Non-local Consensus Voting

Mitchell Phillips

Advisor: Agung Julius

Intuitive Surgical Advisors: Geoff Richmond, Nick Bernstein

Rensselaer Polytechnic Institute

Professional Project - ECSE 6970, Fall 2017

Abstract

Instrument shaft segmentation for surgical robotic images was performed using the methods described in Pathak et al.'s 2017 Conference on Computer Vision and Pattern Recognition (CVPR) paper, "Learning Features by Watching Objects Move" [1]. In particular, pseudo ground truth data for instrument segmentation was generated using the unsupervised motion-based segmentation algorithm presented in section 5.1 of Pathak et al. [1]. Convolutional neural networks (CNNs) trained from pseudo ground truth data have been shown to outperform the leading unsupervised methods for object detection. It is desired that the obtained pseudo ground truth data for instrument segmentation will be used to train Fully Convolutional Networks (FCNs) to perform semantic segmentation.

The unsupervised motion-based segmentation algorithm is largely inspired by Faktor and Irani's 2014 British Machine Vision Conference (BMVC) paper, "Video Segmentation by Non-Local Consensus Voting" [2]. Pathak et al.'s algorithm, denoted as uNLC, differs from Faktor and Irani's algorithm, denoted as NLC, as uNLC substitutes a trained edge detector for an unsupervised superpixel generator.

Using the methods described in Pathak et al. [1] and Faktor and Irani [2], pseudo ground truth data for instrument shaft segmentations was obtained and evaluated.

Contents

1 Introduction **3**

2 Related Works **4**

3 Algorithm Description **5**

3.1 uNLC Overview 5

3.2 Region Extraction and Description 6

3.3 Saliency Mapping 6

3.4 Consensus Voting and Propagation 8

4 Results **8**

4.1 Data Collection 8

4.2 uNLC Results on Robotic Surgical Images 8

4.2.1 Positive Results 8

4.2.2 Negative Results 10

5 Future Work **12**

5.1 Fully Convolutional Networks 12

5.2 Additional Observations and uNLC Modifications 12

6 Conclusion **13**

A Installation and Tutorial **15**

1 Introduction

Minimally invasive robotic surgery enables surgeons to perform difficult procedures with little to no harm on patients. Since the early 2000's, robotic systems, such as the *da Vinci Surgical System*, have provided patients with lower risk to complications, minimal scaring, and shorter hospital stays for a variety of different surgeries. These complex systems have had a tremendous impact on operating room (OR) procedures and are made up of a variety of components which enable them to accomplish a wide array of tasks. The *daVinci* in particular is made up of a patient side cart, a vision cart, and a surgeon console. The patient side cart is equipped with a set of precision tools (forceps, needle drivers, retractors, etc.) which are controlled through master-slave manipulation at the surgeon console. In addition to the precision tools, a 3D endoscope streams HD video to both the vision side cart and the surgeon console to provide patient anatomy information to both the surgeon and additional personal in the OR [3].

While the master-slave control enables the surgeon to navigate restricted areas of the human anatomy safely, a certain amount of physical interaction with the tissue is lost. Thus, surgeons must rely on additional information, such as video, to perceive the environment. One method to overcome these difficulties from this disconnect is through vision-based and mark-less surgical tool detection. These approaches depend only on the video information that is transmitted to the surgeon and do not require any additional hardware modifications, thus making them a cost effective and viable solution [4].

An element of vision-based surgical tool detection is instrument shaft segmentation. This process involves partitioning a given image into multiple segments to indicate whether or not a pixel is part of an instrument shaft (Figure 1). Instrument shaft segmentation provides a stepping stone for tool identification and tracking which in turn will improve the accuracy of tool position estimations, close control loops with image feedback, aid UI and accompanying safety features, as well as provide additional off-screen tool tracking information.

Instrument segmentation is an incredibly challenging problem due to the issues of various lighting conditions, shadows, textures from both the tools and the tissue, metallic properties of the tools, the variability in appearance of different tools, and any blood which may obscure the view. These visual challenges are stacked on top of the fact that the tissue in the image is moving due to biological processes, and that both the camera and instruments are moving and manipulating the tissue.

The proposed solution for instrument shaft segmentation applies the methods discussed in [1]. Specifically, Pathak et al.'s uNLC algorithm was applied to robotic surgical images to generate pseudo ground truth data. It is desired that from the obtained results, an FCN may be trained to perform semantic segmentation on instrument shafts.

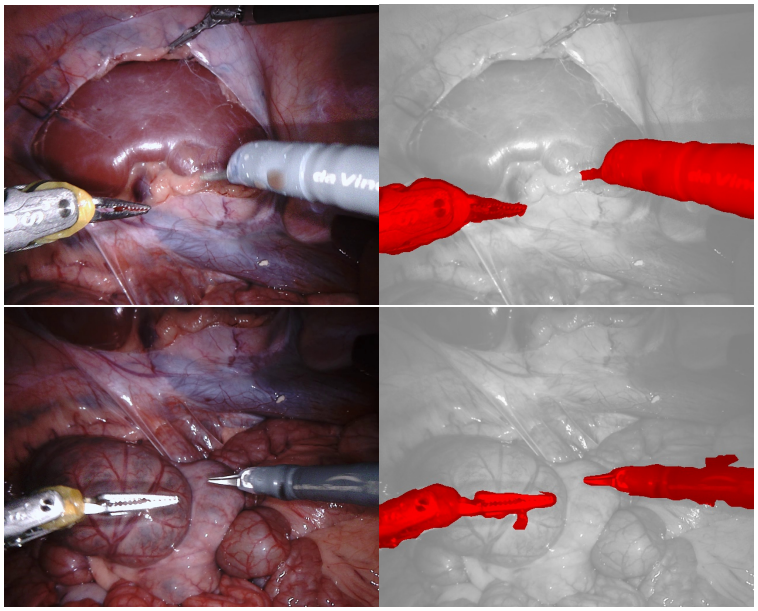

Figure 1: *Example frames and respected segmentations demonstrating the visual information returned by the endoscope and illustrating binary shaft segmentation.*

2 Related Works

An assortment of techniques and methods of robotic tool segmentation and detection have been studied throughout the years. Such methods have included Randomized Trees [5] and Support Vector Machines (SVMs) [6]. However, more recent techniques of tool segmentation utilize CNNs (or ConvNets) [7] and FCNs [8]. In particular, the FCN-8 and ResNet-101 model architectures have been shown to successfully perform both binary [9],[10], and multi-class segmentation [11]. However, in order for these methods to work well, large amounts of annotated data is required. While there are several instrument segmentation datasets available [12] there is nothing to the scale of ImageNet [13] or CoCo [14]. Due to the scarce amounts of training data, current methods turn to transfer learning to perform classification and segmentation. However, results are not always guaranteed through this approach. Thus, there is a current need of generating labeled data which can be used to effectively train ConvNets for instrument segmentation.

Pathak et al. proposed a solution that uses unsupervised motion-based segmentation on videos to obtain pseudo ground truth images, and demonstrated the ability to train a ConvNet to perform object detection from a single frame. While the performance of object detection using this method ranks below supervised learning methods, it established the ability that ConvNets can be utilized without significant amounts of previously annotated data. Furthermore, it may be possible to iteratively train a ConvNet to improve the result by combining refined outputs from the ConvNet with the unsupervised segmentation results [1].

3 Algorithm Description

3.1 uNLC Overview

The publicly available uNLC algorithm [1] was applied to robotic surgery images to accomplish instrument shaft segmentation. The use of uNLC stems from its ability to perform motion segmentation on videos that feature both highly non-rigid objects and complex backgrounds [2]. This proves particularly useful for surgical images as the background consists of moving tissue and human anatomy, as well as the fact that the endoscope may undergo intricate transformations as a surgeon adjusts its position. Such a feat is accomplished by iteratively performing consensus voting over a saliency map. The uNLC algorithm is summarized in Figure 2 and can be described as the following:

Given an image sequence, each frame is broken down into set of regions. These regions are then described by a specific set of measurements, and a nearest neighbor graph is constructed over the regions using these measurements. In addition, an initial saliency map is computed. This saliency map is based on the optical flow and the estimated dominant motion of a given frame. Depending on the dominant motion across all the frames, either a motion-based or visual-based saliency map is created. The saliency map is averaged across all regions in the sequence. Then, a consensus voting process is initiated, and propagates the saliency map across all the frames in the sequence using the region descriptions and a nearest neighbor search. Once the propagation is complete, the consensus voting results are cast to either foreground or background regions and completes the segmentation process [1, 2].

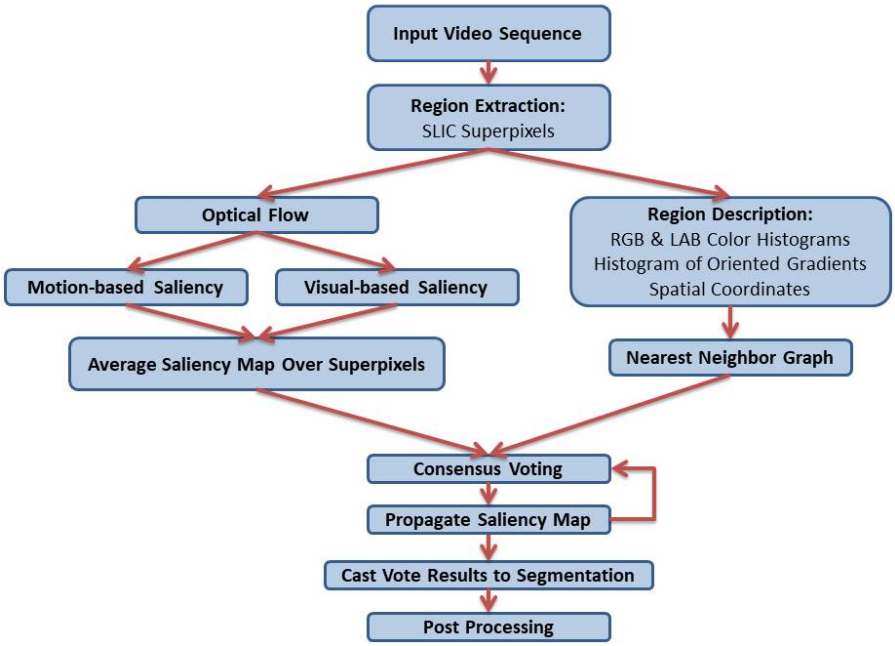

Figure 2: *uNLC Algorithm*

3.2 Region Extraction and Description

The region extraction process is a superpixel calculation. Superpixels are used to describe image regions and reduce the complexity of the image [15]. This stage in the algorithm is where uNLC and NLC differ from one another. Where NLC adopts a trained edge detector, involving random decision trees and a watershed transform [16], uNLC instead performs the region extraction process by simple linear iterative clustering (SLIC) [15]. SLIC is a K-means clustering approach to generating superpixels and has been shown to track image boundaries. Thus, SLIC is a viable replacement for the edge detection method in NLC. An example of SLIC superpixel segmentation is shown in Figure 3. The superpixel region descriptions from SLIC are used to determine the initial saliency maps.

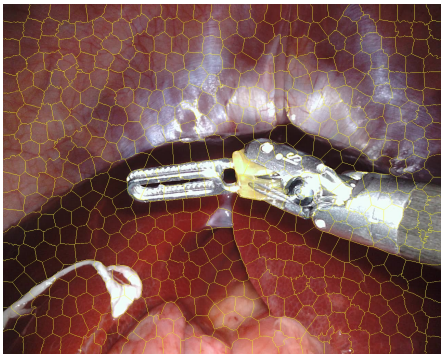

Figure 3: *SLIC Region Extraction*

Each extracted region is then described through a variety of measurements including a concatenation of the RGB and LAB color histograms, a histogram of oriented gradients (HOG) [17], and the normalized spatial coordinates of the regions. Each of the 6 channels of the color histograms are divided into 20 bins and the HOG is computed over a 15×15 patch about the center of each superpixel, with a cell size of 9×9 pixels per cell, 3×3 cells per block, and 6 orientation bins [2].

3.3 Saliency Mapping

Optical flow is used to determine if motion based or visual based saliency is to be applied. In general, optical flow provides a measure of how two images change from one other and the motion that may be occurring. Both uNLC and NLC utilize Ce Liu's Optical Flow algorithm [18]. The controlling parameters for optical flow are the regularization weight, the down-sample ratio, the width of the coarsest level, the number of outer and inner fixed point iterations, and the number of successive over-relaxation (SOR) iterations. The complete uNLC pipeline down-samples frames to a size of 100×100, and the optical flow is measured using a spectral radius of 3. This low resolution representation is used throughout the duration of the initial motion saliency calculation before up-sampling the results through bilinear interpolation [2]. Down-sampled examples of the optical flow estimation are presented in Figure 4.

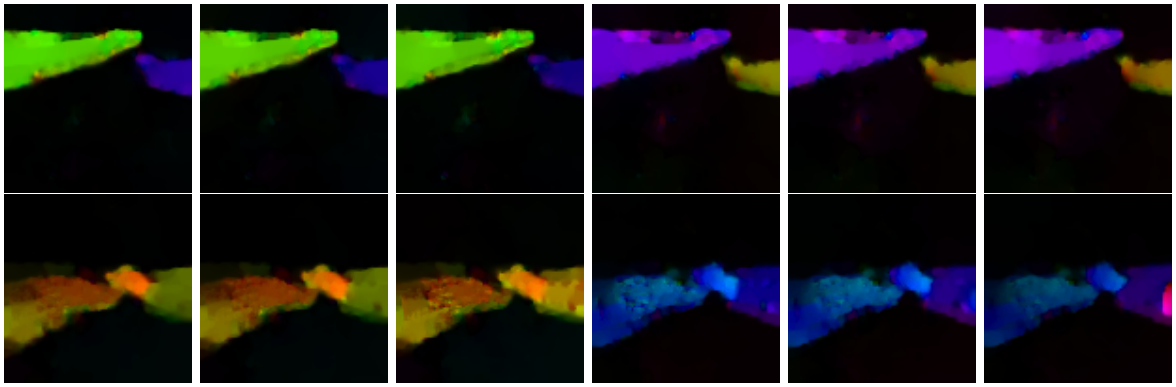

Figure 4: *Down-sampled Optical Flow with Spatial Radius of 3. Each row corresponds to the optical flow measurement of radius 3 about a frame.*

From the optical flow results, the dominant motion, denoted as either *static* or *translational*, of a frame can be determined. If the optical flow median magnitude is below a set threshold, chosen to be 1, the frame is labeled to be *static*. To check if a frame is *translational*, a histogram of optical flow orientations, weighted by the optical flow magnitudes, is created. If the bin with the most counts has a weight larger than a predefined threshold, chosen to be 0.75, the frame is labeled to be *translational*. A 5×5 region of flow vectors about each pixel is then used to calculate the deviation from the estimated dominant motion. These deviations are computed in one of two ways depending on the dominant motion. For *static* frames, the measure is a deviation of flow magnitudes from zero. For *translational* frames, the measure is a deviation of flow orientations from the dominant direction. The deviations are used as the saliency scores if there is dominant motion [2].

The interpretation for the deviations in *static* frames stems from the fact that the median optical flow magnitude is less than 1. Thus, it is assumed that the frame is not moving. Regions which deviate from zero are regions which are moving, and consequently are assigned larger saliency scores to indicate the region as an estimated foreground object. Effectively, if instruments are moving in a frame, then the motion is captured and is appropriately labeled as foreground.

As for *translational* frames, the optical flow direction that is most apparent provides the intuition that either the camera or the majority of pixels in the frame are moving in the same relative direction. Regions or pixels that deviate from this dominant direction are assigned larger scores for the saliency map to indicate that these regions are foreground objects. This enables the uNLC algorithm to segment instruments when the endoscope performs complex camera motions and transformations.

However, it is possible for frames to not meet either criteria for dominant motion. This can occur from non-rigid backgrounds, such as moving tissue of the human anatomy. In situations where this happens, the visual-based saliency is calculated using manifold ranking [19].

Once the initial saliency scores are computed, frames are up-sampled to their original size through bilinear interpolation. The saliency scores are then averaged across superpixel regions and normalized across the entire video sequence [1, 2].

3.4 Consensus Voting and Propagation

The final step in the uNLC algorithm is to broadcast the saliency map across a nearest neighbor graph. This graph is constructed using a kd-tree and searches for the 4 nearest neighbors of every superpixel (using the created region descriptor) over a spectral radius of 15 frames, including the current frame. Thus, the 4 nearest neighbors of every region are found in the 15 previous frames, the current frame, and the following 15 frames. This results in 124 nearest neighbors for every region in each frame, described by the color histograms, HOG, and spatial coordinates. A Gaussian weighting between neighbors and a normalization is then performed on the kd-tree results to obtain the final nearest neighbor graph.

Once the graph is constructed, the initial saliency map is cast to the graph and an iterative voting procedure is conducted. Each iteration consists of updating every region by a weighted average of that region's 124 nearest neighbors. After a certain number of iterations is complete, the votes are projected to either foreground or background, where further post-processing may occur. This marks the completion of the uNLC algorithm [1, 2].

4 Results

4.1 Data Collection

All surgical images and videos used were obtained from Intuitive Surgical. Five video clips with dimensions 1024(height) $\times$ 1280(width), with a resolution of 96 dpi, and a frame rate of either 30 or 60 frames/second with instrument(s) present were provided. Two pairs of videos were noted to be stereo correspondents. Thus, a total of three unique video sequences were evaluated. The video clips varied in length and the actions performed by the instrument(s) ranged from little or no movement to significant tool motion and human anatomy manipulation. In some video sequences, the camera position was adjusted by an operator. Furthermore, the environments for each of the video sequences are different and feature various procedures. No stereoscopic parameters were used.

4.2 uNLC Results on Robotic Surgical Images

Videos were first converted to image sequences by sampling each video by their respected frame rate and then sending the image sequences through the uNLC pipeline. Computing resources used for the evaluation of uNLC on surgical robotic images required that the video clips were fairly short and frames were reduced in size to avoid memory errors. A complete tutorial and information regarding the accompanying software packages to this experiment can be found in Appendix A. A subset of the results is discussed below.

4.2.1 Positive Results

Figure 5 features a subset of images from video_161006_101402_1.avi where two instruments are manipulating tissue. The segmentation results of Figure 5 are shown in Figure 6. The images correspond to frames 72 to 103 of the source video, with a frame gap of approximately 5 frames. It is noted that the frames were resized to be 600×480. For the final implementation of uNLC, the frames are resized by a factor of 2, and the optimal frame gap is still being investigated. The obtained results are considered good due to the fact that the segmentations (red blobs) correspond to the instruments exhibiting motion. Furthermore, the manipulated tissue has not been segmented.

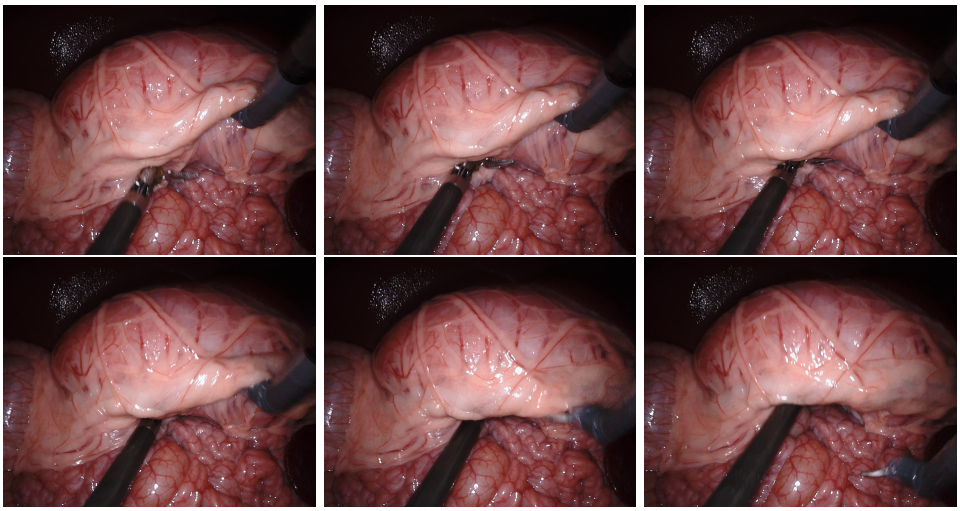

Figure 5: *Video Sequence from video_161006_101402_1.avi. Frames 72 to 103.*

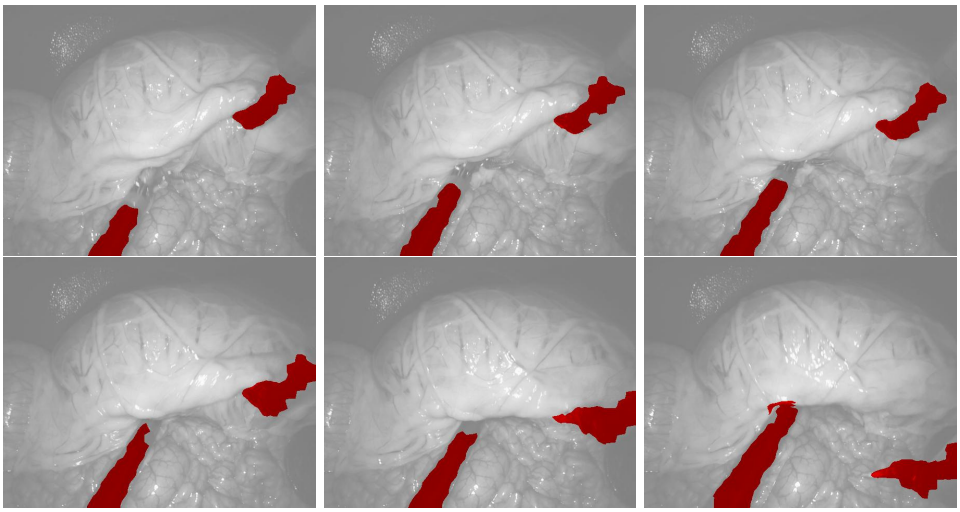

Figure 6: *Segmented Results. Video Sequence from video_161006_101402_1.avi. Frames 72 to 103*

uNLC has the ability to segment tools when there is complex camera motion, as indicated in Figures 7 and 8. In the sequence, the camera is shifted to a new perspective while the instruments are still within the image frame. While the tools may not be moving themselves, the instrument(s) are still labeled appropriately and are segmented correctly through uNLC's motion based saliency and optical flow estimate for dominate camera motion.

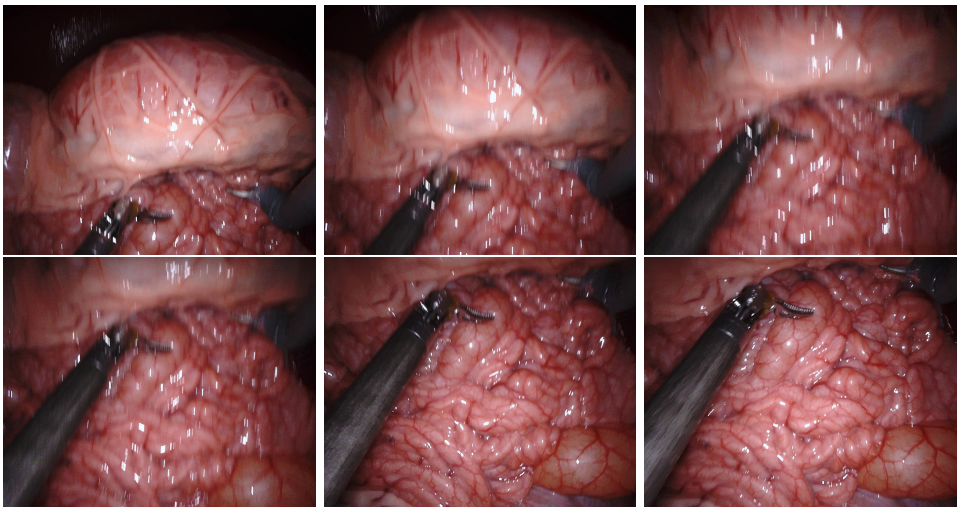

Figure 7: *Video Sequence from video_161006_101402_1.avi. Frames 147 to 174.*

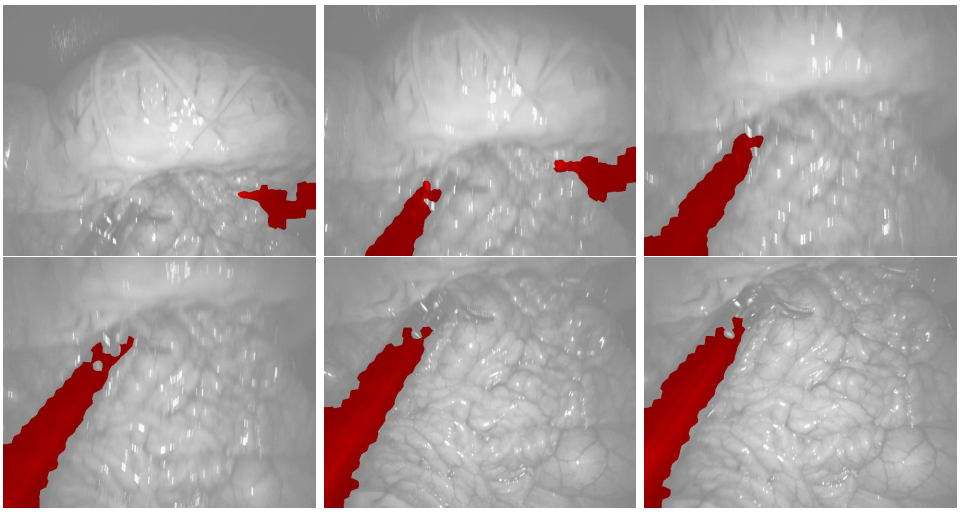

Figure 8: *Segmented Results. Video Sequence from video_161006_101402_1.avi. Frames 147 to 174.*

4.2.2 Negative Results

Not all obtained results from uNLC are positive and segmentations often fail. For the case of robotic surgical images, failed segmentations include not marking pixel locations that belong to moving tools, marking tissue that may be moving, or marking elements that exhibit no motion at all. Examples of failed segmentations are shown in Figure 9.

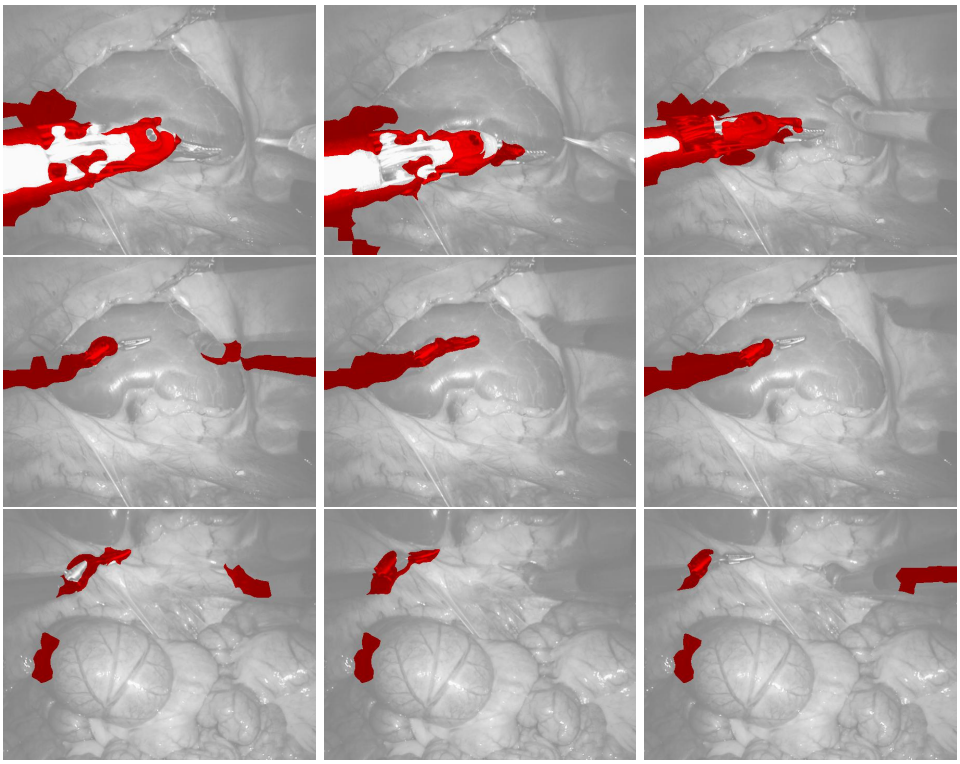

Figure 9: *Selected frames of Poor Segmentation Results*

The segmentation results can be considered poor as not all the moving instruments have been segmented, and the segmentations that have been obtained are incredibly noisy and do not adhere well to object boundaries. In addition, it has been observed that uNLC often times does not label multiple moving tools correctly. For example, if there are two or three instruments present, and at least two are moving, only one of the tools will be marked as foreground pixels. These observations agree with Pathak at el., where it was stated that uNLC often fails when the assumption of a single moving object is not met [1].

The greatest contributer to incorrect segmentations is believed to be caused by the user specified frame gap parameter. While it would be ideal to use all possible frames belonging to a sequence, current computational resources limit the ability to do this. The developed pipeline accounts for this limitation and splits videos into smaller, more manageable sequences. However, incorrect segmentations can still occur and may be caused by previous steps in the algorithm. Recall that the regions are first described by superpixels formed through SLIC. As the superpixels dictate the regions that are to be marked as either foreground or background, if the boundaries which they form are not accurate, the resulting segmentation at the end of the algorithm will be incorrect as well. Furthermore, the region descriptors used are prone to error due to instrument textures, lighting, and reflectance properties.

5 Future Work

5.1 Fully Convolutional Networks

Despite the fact that negative results may be obtained, it has been shown that ConvNets can still learn from noisy image segmentations [1]. The next step is to apply the results from uNLC as the training data for Fully Convolutional Networks (FCNs) [8]. While Convolutional Neural Networks (CNNs) have been shown to perform well for classification and recognition (AlexNet, VGG, ResNet), FCN's are valuable tools for performing segmentation.

CNNs are traditionally composed of convolution, pooling, and activation layers, in addition to fully connected layers to map the results to a probability distribution. An FCN however, is made up entirely of nonlinear filters from the combination and variation of convolution, pooling, and activation layers. Only using these nonlinear filters allows the output to correspond to the spatial dimensions of the original image. Segmentation results are then obtained by up-sampling the nonlinear filter outputs by deconvolution and activation functions. Furthermore, existing CNNs for classification can be cast to FCNs for segmentation. This is done by taking an existing architecture of a CNN, changing the fully connected layers to convolution layers, and then substituting the final classification layer with a convolution of size 1×1 with an output dimension corresponding to the number of classes. Thus, what was once considered a probability distribution over various classes for the entire image is now cast to each pixel, creating a class prediction for every pixel. This output is then up-sampled to provide the final segmentation prediction. Further modifications to the network, such as the inclusion of skip layers, are performed to then improve and refine the results of the FCN [8]. In practice, a pre-trained FCN may need to be adapted and fine-tuned on the obtained images.

FCNs have been successful in performing instrument shaft segmentation and several methods and architectures have been proposed. In [9], a pre-trained FCN-8 model architecture on the PASCAL dataset was fine-tuned on surgical instruments to obtain segmentations. Likewise, in [11], instrument segmentation was performed using the ResNet-101 architecture and dilated convolutions. It is desired to implement such ideas on the obtained pseudo ground truth images from uNLC. Possible future experiments may include the use of a pre-trained ResNet model on the CoCo dataset, investigating the effects of having pseudo ground truth segmentations where only one of multiple instruments is indicated, and exploring the use of a mixture of pseudo ground truth and professionally annotated images.

5.2 Additional Observations and uNLC Modifications

The Kernel Temporal Segmentation (KTS) algorithm [20] was used as a pre-processing technique for dividing up video clips before sending sequences through the uNLC pipeline in Pathak et al.'s implementation [1]. Long videos featuring different scenes can be separated out using KTS. It was decided not to implement this algorithm as the majority of video sequences being evaluated featured similar content and long clips were divided up manually. If a large number of robotic surgical images were to be considered at once, containing a variety of surgical environments with different tools, it may be beneficial to include KTS to divide up video sequences into appropriate categories.

Furthermore, while Pathak et al. did not report any runtime metrics for uNLC, Faktor and Irani achieved an average runtime of 12 secs per frame for NLC [2]. Unfortunately, the implementation of uNLC for this procedure did not achieve such results. Run times of approximately 30 secs per frame were experienced. It is believed that the bottle neck in the algorithm is the nearest neighbor search due to the obtained run times

at this step. This may be caused by the fact that the search occurs over high dimensional data from the region descriptors.

While it has been shown that ConvNets can achieve acceptable performance for noisy data [1], it has yet to be proven specifically for robotic surgery images. Thus, it may be beneficial to explore changes for uNLC to improve the results or decrease the computation time. Potential changes include investigating alternatives for the nearest neighbor search, the substitution of Ce Liu's Optical Flow calculation for a more recent, state-of-the-art algorithm (such as application of MR-Flow [21] or FlowNet2[22]), or using other descriptors that are more aligned/specific to surgical instruments. Moreover, the restriction of unsupervised methods does not need to be considered, and any techniques involving trained data and supervised learning can be evaluated.

6 Conclusion

Pseudo ground truth data for instrument segmentation was generated using the unsupervised motion-based segmentation algorithm presented by Pathak et al. in the paper *Learning Features by Watching Objects Move* [1]. Results for segmentation varied, however it is believed that the obtained pseudo ground truth may be beneficial for training FCN's. Several improvements for the algorithm have been proposed, and recommendations to build off the work have been presented. The next step for this procedure is to investigate the effects of using pseudo ground truth data to train an FCN to perform instrument shaft segmentation.

References

[1] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan, "Learning features by watching objects move," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] A. Faktor and M. Irani, "Video segmentation by non-local consensus voting," in *BMVC*, 2014.

[3] Intuitive Surgical Inc., "The davinci surgical system." https://www.intuitivesurgical.com/products/davinci_surgical_system/.

[4] D. Bouget, M. Allan, D. Stoyanov, and P. Jannin, "Vision-based and marker-less surgical tool detection and tracking: a review of the literature," *Medical Image Analysis*, vol. 35, no. Supplement C, pp. 633 – 654, 2017.

[5] A. Reiter, P. K. Allen, and T. Zhao, *Feature Classification for Tracking Articulated Surgical Tools*, pp. 592–600. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[6] D. Bouget, R. Benenson, M. Omran, L. Riffaud, B. Schiele, and P. Jannin, "Detecting surgical tools by modelling local appearance and global shape," vol. 34, pp. 1–1, 12 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[9] L. C. García-Peraza-Herrera, W. Li, C. Gruijthuijsen, A. Devreker, G. Attilakos, J. Deprest, E. Vander Poorten, D. Stoyanov, T. Vercauteren, and S. Ourselin, *Real-Time Segmentation of Non-rigid Surgical Tools Based on Deep Learning and Tracking*, pp. 84–95. Cham: Springer International Publishing, 2017.

[10] L. C. García-Peraza-Herrera, W. Li, L. Fidon, C. Gruijthuijsen, A. Devreker, G. Attilakos, J. Deprest, E. B. V. Poorten, D. Stoyanov, T. Vercauteren, and S. Ourselin, "Toolnet: Holistically-nested real-time segmentation of robotic surgical tools," *CoRR*, vol. abs/1706.08126, 2017.

[11] D. Pakhomov, V. Premachandran, M. Allan, M. Azizian, and N. Navab, "Deep residual learning for instrument segmentation in robotic surgery," *CoRR*, vol. abs/1703.08580, 2017.

[12] "EndoVision Instrument Dataset." https://endovissub2017-roboticinstrumentsegmentation.grand-challenge.org/data/.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[14] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[15] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 2274–2282, Nov 2012.

[16] P. Dollar and L. Zitnick, "Structured forests for fast edge detection," International Conference on Computer Vision, December 2013.

[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005.

[18] C. Liu, *Beyond Pixels: Exploring New Representations and Applications for Motion Analysis*. PhD thesis, Massachusetts Institute of Technology, 5 2009.

[19] C. Yang, L. Zhang, R. X. Lu, Huchuan, and M.-H. Yang, "Saliency detection via graph-based manifold ranking," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 3166–3173, IEEE, 2013.

[20] D. Potapov, M. Douze, Z. Harchaoui, and C. Schmid, *Category-Specific Video Summarization*, pp. 540–555. Cham: Springer International Publishing, 2014.

[21] J. Wulff, L. Sevilla-Lara, and M. J. Black, "Optical flow in mostly rigid scenes," *CoRR*, vol. abs/1705.01352, 2017.

[22] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *CoRR*, vol. abs/1612.01925, 2016.

A Installation and Tutorial

Refer to the following GitHub link for complete installation instructions and software packages. Instructions listed here may change overtime. GitHub: https://github.com/phillm4/ISS_uNLC.
Using the methods described by Pathak et al. and Faktor & Irani, pseudo ground truth data for instrument shaft segmentations was obtained and evaluated. The installation process and instructions on using the software is described below.

Disclaimer

The majority of the software used for this project is from Pathak's videoseg library. While a handful of modifications and changes were introduced, the software is ultimately under the ownership of Pathak et al, the authors of "Learning Features by Watching Objects Move".

```
@inproceedings{pathakCVPR17learning,
    Author = {Pathak, Deepak and Girshick, Ross and Doll\'{a}r,
              Piotr and Darrell, Trevor and Hariharan, Bharath},
    Title = {Learning Features by Watching Objects Move},
    Booktitle = {Computer Vision and Pattern Recognition ({CVPR})},
    Year = {2017}
}
```

Description of Contents

File / Folder	Description
additional_tools	Contains extra scripts that may be beneficial
lib	Location where all external code will be stored
modified_scripts	Contains several scripts that will be placed into external code libraries
_init_nlc_path.py	Script which initializes library paths for iss_main.py
iss_main.py	Program to initiate uNLC algorithm
test_batch.zip	Zipped folder containing test images for the program demo

It is intend in the future to remove the bulk of the dependencies on scikit-image and PIL, and instead use OpenCV exclusively. This process has already been initiated.

Required Libraries and Additional Dependencies

In order to run the python scripts for uNLC, several additional libraries are required. One of the required libraries is pyflow. Pyflow is a wrapper around Ce Liu's C++ implementation of Coarse2Fine Optical Flow, and utilizes the python package Cython. Cython consists of C-extensions for Python. When attempting to build the pyflow library on a windows machine, "error: Unable to find vcvarsall.ba" was encountered. This appears to be a common error due to Visual Studio. As a result, all work was completed on an Ubuntu 16.04 LTS system. Additional libraries include the following:

```
OpenCV3 - cv2
Cython
numpy
Python Imaging Library - PIL
scipy
scikit-image - skimage
```

The majority of the libraries can be installed via pip or conda with the exception of OpenCV. While the uNLC algorithm does work without the FFmpeg dependencies, it is required if video files are to be read. If it is not desired to work with videos, the pip or conda installation of OpenCV should work.

Installation Instructions

The installation instructions mimic those of Pathak's videoseg, however the installations of Dense CRF, Kernel Temporal Segmentation, and DeepMatching are all neglected. These packages need to be installed if it is desired to run Pathak's `full_pipe.py` script. In addition, several modifications are required in order for functions to work properly. A handful of scripts have been included as an attempt to mitigate these issues.

Notation

Throughout the installation procedure, the bulk of the user path will be omitted. For example, `/home/user/path/$` will be referred to as `$` , and commands like, `$ cd /home/user/path/ISS_uNLC/lib/` will be indicated by `$ cd ISS_uNLC/lib/` .

1. Download and install this repository.

```
$ cd [Path where this will be installed]
$ git clone https://github.com/phillm4/ISS_uNLC.git
```

2. Download and install NLC. As this will be used as a python library, a `__init__.py` file needs to be included.

```
$ cd ISS_uNLC/lib/
$ git clone https://github.com/pathak22/videoseg.git
$ cp __init__.py videoseg/
```

3. Download and install pyflow. Note that the installation path has changed. Furthermore, it is required to build pyflow as it is a C++ wrapper.

```
$ cd ISS_uNLC/lib/videoseg/lib/
$ git clone https://github.com/pathak22/pyflow.git
$ cd pyflow/
$ python setup.py build_ext -i
```

4. Download and install Visual Saliency. Similar to NLC, a `__init__.py` file needs to be included. Note the installation path.

```
$ cd ISS_uNLC/lib/videoseg/lib/
$ git clone https://github.com/ruanxiang/mr_saliency.git
$ cp __init__.py mr_saliency/
```

After installing the above dependencies, several modifications are required for everything to work properly.

5. Fix MR.py issues. `mr_saliency/MR.py` handles the visual saliency calculation in uNLC. However, the script is not compatible with Python 3 nor the current version of scikit-image. In order to fix these issues, `<>` needs to be changed to `!=`. Python 3 no longer supports `<>` as a comparison operator. The second change is to remove the importing of 'lena' from the skimage.data module. lena has been removed from scikit-image due to copyright issues. A simple fix is to import a different image from the skimage.data module and name that as lena. To make this process simpler, a modified `MR.py` script is included in this repository and can be placed in the `mr_saliency` library. This procedure is shown below.

```
$ cd ISS_uNLC/
$ cp modified_scripts/MR_mod.py lib/videoseg/lib/mr_saliency/
```

6. Add nlc_mod.py. The last step in the installation process is to move the included `nlc_mod.py` script into *videoseg/lib*. `nlc_mod.py` is a modified version of `videoseq/src/NLC.py` which allows for the tuning of the pyflow parameters. It is intended to update `nlc_mod.py` in the future as to remove this step in the installation process and to remove additional dependencies.

```
$ cd ISS_uNLC/
$ cp modified_scripts/nlc_mod.py lib/videoseg/src/
```

uNLC should be ready to use.

Usage Instructions

All operations are handled by `iss_main.py`. This is the main program to run the uNLC algorithm and is a wrapper around Pathak's nlc algorithm. It is inspired by the NLC library's `run_full.py` and `nlc.py`. The function can be executed from the command line and accepts a variety of arguments. In general, `iss_main.py` requires an input corresponding to the path of either an image directory, a directory containing subfolders with images, or a video. Additional arguments include the ability to adjust the output directory and a frame gap. Use `-h` to view the arguments.

```
$ cd ISS_uNLC/
$ python iss_main.py -h

usage: iss_main.py [-h] [-indir INDIR] [-outdir OUTDIR] [-batch BATCH]
                   [-vid VID] [-fgap FGAP]

Wrapper for Pathak's Non-Local Consensus Voting algorithm for Instrument Shaft
Segmentation.

optional arguments:
  -h, --help      show this help message and exit
  -indir INDIR    Path to the input directory that contains the images to
                  test.
  -outdir OUTDIR  Path to the output directory to save results. If no
                  directory is specified, 'results' directory will be created
                  at the same level as the input directory.
  -batch BATCH    Batch input. Path to a directory that contains sub-folders
                  containing images to test.
  -vid VID        Path to the video file to be converted. Include the video
                  itself. If inputting a video, the video will be split into
                  subfolders to conserve memory when running uNLC.
  -fgap FGAP      Frame gap between images in a sequence. Default 0.
```

Note that there are three different methods for inputing data, and two additional arguments which correspond to an output directory and a frame gap. The frame gap is important when specifying the frames that uNLC is to compute across in a given image sequence. If a frame gap too small is chosen, memory errors may occur and the computation may take a long time (up to an hour). When using uNLC, it is important to be aware of the length of image sequences and the frame gap as to prevent memory issues. If a frame gap too large is chosen, the obtained results will be useless. Furthermore, uNLC works best on image sequences where there is significant motion by the foreground objects. If the objects are barely moving, or move relatively slow, it may be beneficial to increase the frame gap. The output directory is where the segmented images will be saved. If this is not specified, a results directory will be created at the same level of any given input. Two examples are shown below.

Example (1.)

Perform segmentation using the '-batch' input. In the context of `iss_main.py`, this option is to be selected if there is a folder that contains several subfolders, each of which that contains images. This is illustrated as the following.

```
- batch_folder/
    -- image_directory_00/
          --- img_00_00.jpg
          --- img_00_01.jpg
          --- ...
    -- image_directory_01/
          --- img_01_00.jpg
```

```
            --- img_01_01.jpg
            --- ...
        -- ...
```

In order to then perform segmentation on this batch, the commands and potential output are shown below. For this example, the included *test_ batch/* will be used as the input batch folder, the output directory will be generated automatically, and a frame gap of 3 frames will be used. This process will take several minutes to complete.

```
$ cd ISS_uNLC/
$ unzip test_batch
$ rm test_batch.zip
$ python iss_main.py -batch test_batch -fgap 3

Batch:  0
Input Directory:  /home/.../ISS_uNLC/test_batch/00
Output Directory:  /home/.../ISS_uNLC/results
Loading images: [ ##.# %]
Total Sequence Shape:  (##, ###, ###, #)
Memory Usage for Sequence: ##.## MB.

*****Performing uNLC*****

Superpixel computation: [ ##.# %]
Superpixel computation finished: #.## s

Descriptor computation: [ ##.# %]
Descriptor computation finished: #.## s

NearestNeighbor computation: [ ##.# %]
NearestNeighbor computation finished: ##.## s

Constructing pyramid...done!
Pyramid level 4
Pyramid level 3
Pyramid level 2
Pyramid level 1
Pyramid level 0
Constructing pyramid...done!
Pyramid level 4
Pyramid level 3
Pyramid level 2
Pyramid level 1
Pyramid level 0
Constructing pyramid...done!

Motion Saliency computation finished: #.## s
Consensus voting finished: #.## s

*****uNLC complete. Save results.*****

Removing low energy blobs finished: #.## s

Batch:  1
Input Directory:  /home/.../vidpath/src_images/00
Output Directory:  /home/.../vidpath/results
```

```
Loading images: [ ##.# %]
Total Sequence Shape:  (##, ###, ###, #)
Memory Usage for Sequence: ##.## MB.
...
```

Once the process is complete, the segmentation results will be located in the created *results/* folder.

Example (2.)

Segment a **short** video that is located at *home/.../vidpath/video.avi* (not included in this repository). No output directory will be specified and a frame gap of 3 will be used. Segmenting directly from a video is not recommended at this time and it is important that the video is short.

```
$ cd ISS_uNLC/
$ python iss_main.py -vid home/.../vidpath/video.avi -fgap 3

Video to Image Conversion: [ ##.# %]

Batch:  0
Input Directory:  /home/.../vidpath/src_images/00
Output Directory:  /home/.../vidpath/results
...
```

The output should mimic that of Example 1.
Once the process is complete (it may take several minutes depending on the length of the image sequence), two new folders *src_images/* and *results/* should have been created where the video is located. One contains the frames from the video (*src_images/*), the other contains the segmentation results (*results/*). The size of the batches created by the video to image conversion can be edited and is denoted images_per_batch.

Tuning uNLC

If it is desired to adjust any of the parameters for the uNLC algorithm besides the frame gap, these can be modified within `iss_main.py` under the iss_uNLC() function.

```
## # Parameters for uNLC and pyflow.
# memory_limit - Memory limit of image sequence for numpy array.
# resize_fraction -  Fraction that image will be resized by.
# max_superpixels - Total number of superpixel regions.
# vote_iterations - Number of times to perform consensus voting.
# segmentation_energy_threshold - Threshold for finding foreground objects.
# relative_energy - Remove objects where:
#   (total energy <= relative_energy * foreground_pixel_size)

memory_limit = 100
resize_fraction = 0.5
max_superpixels = 1000
vote_iterations = 100
segmentation_energy_threshold = 0.3
relative_energy = segmentation_energy_threshold - 0.1
clearVoteBlobs = False
clearFinalBlobs = True
pyflow_parameters = dict(
    alpha = 0.012,
    ratio = 0.75,
    minWidth = 20,
```

```
nOuterFPIterations = 7,
nInnerFPIterations = 1,
nSORIterations = 30)
```

Next Steps

This concludes the installation and use instructions for uNLC. Several changes to the algorithm are planned for the future. It is intended to use the generated results to train a Fully Convolutional Network to perform instrument shaft segmentation. Additional scripts including a pytorch example and a slic superpixel demo can be found in the *additional_tools* directory.