Naval Research Laboratory

Washington, DC 20375-5320

NRL/7340/MR--2021/1

MIST: An Interactive Machine Learning Application for Video Segmentation

Laura K. Smith

Chris J. Michael

Center for Geospatial Sciences
Oceanography Division

February 25, 2021

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
25-02-2021	NRL Memorandum Report	

4. TITLE AND SUBTITLE

MIST: An Interactive Machine Learning Application for Video Segmentation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Laura K. Smith and Chris J. Michael

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
6B66

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/7340/MR--2021/1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
One Liberty Center
875 N. Randolph Street, Suite 1425
Arlington, VA 22203-1995

10. SPONSOR / MONITOR'S ACRONYM(S)

ONR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This document presents the background, implementation details, and preliminary results for the U.S. Naval Research Laboratory's proprietary Motion Imagery Segmentation Tool (MIST), an application built for Matlab. This report is selfcontained in explaining the necessary concepts within the fields of statistics, machine learning, and computer vision in order to understand the inner workings of the MIST application and the preliminary results.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Chris J. Michael
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	SAR	30	**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U			(228) 688-4955

This page intentionally left blank.

CONTENTS

1. INTRODUCTION 1

2. BACKGROUND 1
   2.1 Machine Learning 1
   2.2 Imagery 4
   2.3 Image Segmentation 5

3. IMPLEMENTATION 5

4. DATA SET 7

5. RESULTS 8

6. MIST USER'S GUIDE 11

7. CONCLUSION 24

REFERENCES 24

FIGURES

1 Annotation with respect to Labeling Technique 8

2 Classification Accuracy with respect to Annotated Images 9

3 An Example of Predicted Annotation 10

TABLES

1 Image Filtering Functions 5

2 Feature Space Variables 6

3 Breakdown of Pixel Count 8

This page intentionally left blank.

MIST: AN INTERACTIVE MACHINE LEARNING APPLICATION FOR VIDEO SEGMENTATION

1. INTRODUCTION

Many branches of research rely on video capture as a measurement tool. For instance, video analysis of water surfaces may be used to map the underlying bathymetry [2]. In many cases, the region of interest must be extracted from the video in order to make measurements more accurate. Although modern-day machine learning applications could potentially do this, they are overwhelmingly geared towards supervised approaches which require very large curated training sets. Though these techniques may yield high-accuracy results, replicating these results on locally created data can be frustrating for several reasons. First, camera calibration and regional context may cause a local video capture to be an outlier against a curated general-purpose training set. This can severely degrade the accuracy of the classifier. Second, conventional approaches are typically *batch learning* approaches, meaning that they do not support fine-tuned adjustments or corrections to incorrect classifications. Finally, many conventional applications are provided solely as source code or application-program interfaces and do not provide intuitive graphical user interfaces to analyze and tune the machine learning model.

It may be more favorable for a user to build a special-purpose training set focused on the particular class and video of interest. The *Motion Imagery Segmentation Tool* (MIST) is a semi-automatic annotation tool that performs this task. It provides a pixel-perfect extraction of a single class with no prior training data. MIST has several forms of output including a full-resolution video with all unnecessary pixels masked and a set of B&W images denoting the pixels of interest for every frame in the video. The application provides a set of productivity tools for manual annotation along with an online human-in-the-loop machine learning methodology that allows for quick refinement via user annotation. In this document, we will describe the background necessary to understand the state-of-the-art techniques leveraged by the application we present, the underlying machine learning implementation, and a user's guide outlining a sound procedure for creating high-quality extractions.

2. BACKGROUND

2.1 Machine Learning

2.1.1 Basic Concepts

Broadly, machine learning is simply the estimation of a mathematical function that produces the desired output in order to complete some task. There are two general categories of machine learning: those that require samples of the desired output function, called *supervised learning*, and those that do not, called *unsupervised learning*. A *feature* is an independent variable that describes some characteristic of the incoming data. The desired output is generally a feature referred to as a *class*. Every feature is mapped to an axis of the mathematical function. A *feature vector* represents some point on the function. Known feature vectors used to model the estimation of the mathematical function are referred to as *training data*. The mathematical model that performs the function estimation is typically referred to as a *classifier*.

Manuscript approved February 22, 2021.

2.1.2 Interactive Machine Learning

The vast majority of machine learning research falls into the category of *automatic machine learning*. Automatic machine learning tends to focus on the accuracy of the implementation independent of training time, computer performance, and usability. On the other hand, *interactive machine learning* implementations are evaluated with criteria that become important for problems where human intervention is necessary [3].

Applications of interactive machine learning usually entail an *online machine learning* implementation, which describes a model that improves over time via incremental discovery of training data. Interactive machine learning implementations focus on computational performance, cognitive load, and incremental training, as opposed to automatic machine learning implementations where these metrics are ignored. Because interactive machine learning applications present guesses of the task at hand to a user for confirmation or correction, *active learning* is usually implemented to maximize the rate of convergence under some cognitive cost incurred by the user. Active learning is a method of training by presenting deliberately chosen examples for labeling. Typically, instances are chosen based on their statistical significance in order to reduce incorrect guesses by the implementation [4]. There are many ways to calculate statistical significance of the problem space, such as using data clustering or filtering.

A term coined from recommender systems research, the *cold start* problem is a data modeling problem where the task of a new user on a potentially new set of data is not initially known [5]. The cold-start problem causes very long convergence times in many supervised learning classifiers, since these implementations rely on large amounts of training data. There are many ways to mitigate the cold-start problem including active learning and data filtering. Though cold-start learning is less common in commercial machine learning applications, it is very useful in situations where there is no labeled dataset, data context shifts (known as *concept drift*), or the user's task or interests may change over time (known as *concept evolution*). Akin to the cold-start problem is *one-shot learning*, where a machine learning implementation is evaluated on its performance under a very small selection of training data [6].

The human-computer interface, most typically a graphical or textual driven interface on a personal computer, is a very important part of an interactive machine learning implementation. These interfaces may be built to minimize the cognitive load of the user while constraining the problem space for more effective classifier c onvergence [7].

2.1.3 Supervised Learning Implementations

For the reader to better understand the methodology of our experimentation, we will now provide high-level descriptions of the underlying supervised learning algorithms used in MIST.

k Nearest Neighbors is an *instance-based* supervised learning algorithm. Instance-based learners are those that keep explicit training data feature vectors in their memory where they are searched and operated upon to perform a classification. k Nearest Neighbors uses a distance metric, such as Euclidean distance, in order to evaluate the input to find the k closest feature vectors to that input within its training instances. Once found, some mathematical operation is performed across those feature vectors in order to generate the output classification. k Nearest Neighbors is a good algorithm for delivering quick results. It can draw accurate predictions from sparse data, minimizing the time cost of classification. It generally boasts a very small parameter space, so it is easy to understand and thus may yield a more trustworthy implementation.

The most important parameter is typically k, the number of neighbors, and this is often manually determined through trial and error. Adjusting k can fit the model more tightly or loosely to the data, allowing the user to avoid overfitting [8]. With a manageable amount of data (typically less than one million instances) and a lower feature space (typically around a dozen or so), most implementations are able to train and classify very quickly. However, the search algorithm used to find the nearest neighbors grows logarithmically with respect to the number of feature points. More detrimental to performance lies in the fact that higher dimensionality causes the search algorithm to become much less efficient, which is known as the 'curse of dimensionality'. With a higher amount of training instances and features, k Nearest Neighbors algorithms can very quickly suffer crippling speed deterioration. Therefore, this algorithm is best used on relatively small or constrained problems.

Neural networks are biologically inspired machine learning implementations that have the potential to solve much more complex problems. Unlike instance-based learners, which draw conclusions by comparing feature vectors, neural networks attempt to find a formula for correctly predicting the class of a feature vector without directly using trained instances. Neural networks begin with an untrained network of neuron models and then iteratively make adjustments to this network through a complex technique called backpropagation. Through backpropagation, neural networks tweak each neuron's parameters to correct an incorrect prediction, allowing the neural network to learn from its mistakes. As a result, neural networks tend to asymptotically approach a maximum accuracy when training, so training must be terminated once a point of diminishing returns is reached. Although neural networks can take much longer to train than instance-based classifiers, they make predictions quickly due to the lack of need to search labeled instances. Due to their speed and accuracy, supervised neural networks have dominated machine learning literature in the past 10 years [9]. However, the vast parameterization of neural networks makes for low potential for understandability and trust.

One of the more elementary neural networks is the multilayer perceptron. It is made up of several layers of neuron models that perform simple multiplicative calculations. Each perceptron has a set of weights and a bias, and these are the parameters that get adjusted during backpropagation. The first layer of the network uses each magnitude of the feature vector as inputs, and then each layer after uses the outputs of the previous layer as input [10]. Since each neuron can only execute simple functions, this structure allows them to work together to create a function that solves complex classification problems. More details of multilayer perceptrons is discussed by Hagan et al. and Bishop.

Despite their capabilities, multilayer perceptrons lack the technological advancements of more complex neural networks. Due to high processing availability on modern-day computing hardware, it is possible to design neural networks with a relatively high number of neurons and layers. These implementations are commonly referred to as *deep networks*. A popular deep network for image-related problems is the convolutional neural network (CNN) [12, 13]. Unlike multilayer perceptrons, which train on one feature vector at a time, CNN's train on matrices of neighboring feature vectors. These matrices, called feature maps, preserve the spatial layout of the input instances. This allows convolutional neural networks to incorporate complex spatial relationships into the model used for classification [14]. With multiple convolutions, these networks can find broad image elements through examination of localized elements. They can also recognize image elements when they shift, so they interpret images with spatial invariance. Given enough training data, convolutional neural networks will typically outperform k nearest neighbors and multi-layer perceptrons for more complex classification problems. However, deep networks typically require even larger amounts of time to train since the complexity of backpropagation increases with respect to the number of neurons and layers in the network.

A challenge for those designing deep networks is deciding how many layers to use. If a network has too few layers for a given problem, then adding more will improve its performance. When designing a network, one option is to use an excessive number of layers, since in theory, a network with too many layers should still be able to solve the given problem. However, evidence shows that using an excessive number of layers begins to degrade accuracy [13]. *Residual learning* attempts to fix this accuracy degradation problem, allowing deeper networks to be used. Residual learning improves the accuracy of excessively deep neural networks by making "shortcut connections" in the network. These are branches in the network structure that allow data to bypass layers. Unnecessary layers are effectively ignored, making the network less deep. As a result, residual networks are easier to optimize. This structure is a good fit for problems with unknown complexity.

DeepLab v3+ is a network model designed specifically for semantic segmentation, a process which we will cover in more detail in the next section. This model uses ResNet-18, an 18-layer residual neural network, as the base. According to Napoletano et al., this layer depth is a "good tradeoff between computational time and performance". DeepLab v3+ applies special pooling operations to the ResNet architecture. Consult Chen et al. for more details.

2.2 Imagery

2.2.1 Optical Flow

Optical flow, also known as image velocity, describes the velocity of points in a video [17]. For any location within the frame of a video, optical flow algorithms estimate the velocity of the object at the corresponding time and location. MIST associates each pixel in an image with a 2D vector describing optical flow. Intuitively, the optical flow characteristics of certain features, like water, will be unique from other features, like trees.

The Horn and Schunk method and the Lucas-Kanade Derivative of Gaussian method are two commonly used algorithms for calculating optical flow. Both methods require time-dependent, grayscale images and derive predictions from derivatives of pixel intensity [17]. This is further discussed in [18].

2.2.2 Spatial Filtering

Image filtering alters an image to achieve a desired effect, such as sharpening. Spatial filters do this by changing each pixel intensity value based on the values of nearby pixels. These filters examine each pixel and apply a function based on a set spatial relationship to pixels in the neighborhood. For example, Matlab's `stdfilt()` function creates an image where each pixel is the standard deviation of pixel intensity values in a 3x3 neighborhood [1]. This returns an output image where higher magnitudes represent more color variation in the corresponding input image. Other filters, some described in Table 1, also provide different representations of an image.

2.2.3 Texture Analysis

Texture analysis attempts to quantify texture and distinguish different textures from one another. This is done by observing patterns in pixel intensity values. A gray level co-occurrence matrix, for example, returns statistics on the likelihood of two pixel intensity values to be horizontally adjacent [19]. Texture analysis can be used to separate regions with different textures, known as texture segmentation. This is useful when color or brightness does not clearly separate regions.

Table 1—Image Filtering Functions

Basic Image Filtering in the Spatial Domain	
`stdfilt`	Local standard deviation of image
`entropyfilt`	Local entropy of grayscale image
`fibermetric`	Enhance elongated or tubular structures in image
Edge-Preserving Filtering	
`imguidedfilter`	Guided filtering of images

2.3 Image Segmentation

Image segmentation refers to the process of delineating a specific object or region of interest from a digital image [20]. Implementations of image segmentation typically produce a contour or a mask that shows a simplified representation of the interesting part of the image. Research in the field of image segmentation is broad and deep, with most applications falling within the fields of medical imagery [21], autonomous vehicles [22], and geographic region digitization [23].

Though there are numerous approaches to image segmentation, many approaches are targeted towards certain classes by using a model for delineation. Such implementations typically allow for fine-tuned adjustment of the model for a local dataset. A popular technique for such adjustment, where a user may color small subsets of pixels that represent positive and negative samples of the class, is referred to as *scribbling* [24]. Scribbling is a low-labor refinement technique for generating labeled class examples, which may allow for the model to quickly converge to the desired accuracy. However, the diversity of the labeled examples are only as good as the user's guess, and thus scribbling is prone to low-quality labels and longer convergence times.

The act of assigning every pixel in an image to a particular class is referred to as *semantic segmentation*. Semantic segmentation is a popular subfield of of image segmentation that tries to solve a more generalized problem of understanding an entire scene rather than just one object. Automatic approaches to semantic segmentation are numerous, and many of the most recent contributions use supervised machine learning to generate highly accurate results [25]. These implementations require pre-labeled sets of images, called *training sets* with known classes accurately marked for each pixel. Training sets are typically created by hand over long periods of time and require hundreds or even thousands of hours to create and curate [26]. Training datasets tend to be built upon market-driven classes such as those necessary for autonomous vehicles, and thus may not readily be used for extraction of a more nuanced class that a specialized scientist may be interested in.

3. IMPLEMENTATION

The Motion Imagery Segmentation Tool (MIST) aids in video processing by masking the pixels showing the desired class in each frame. A user of MIST is able to generate a full mask of a video using a minimal set of manual annotations. The tool has a graphical user interface (GUI) for annotation and ease of use.

MIST uses a combination of scribbling and pixel-perfect annotation to generate an adequate amount of training data for accurate deep neural-network classifiers specifically tuned towards a single video or a

group of contextually similar videos. Scribbles are used to train a shallow neural network, which predicts the labels for the full image. The user is then able to correct the full image labels through a simple click-and-drag contour interface. This quickly creates pixel-perfect annotations. Many pixel-perfect labeled frames may then be used to train a deep network to predict the region masks for the entire video. This is all done through a GUI, so the user does not have to concern themselves with machine learning parameterization, databases, or code.

Table 2—Feature Space Variables

	Classifier	
Variables	patternnet	DeepLab v3+
red	x	x
green	x	x
blue	x	x
x position	x	
y position	x	
HS flow	x	x
LKDoG flow	x	x
stdfilt	x	x
entropyfilt	x	x
fibermetric	x	x
imguidedfilter	x	x

The feature space used for each classifier is shown in Table 2. The RGB color channels of the image create the red, green, and blue feature values, and the matrix index of each pixel are its X and Y position. The feature space also contains optical flow values, which are calculated using both the Horn and Schunk (HS) algorithm and the Lucas-Kanade Derivative of Gaussian (LKDoG) algorithm. Because performance can differ significantly between different optical flow methods [17], these two algorithms each give a unique perspective on the images. For each algorithm, the pixel velocity magnitude is used as a feature, creating "HS flow" and "LKDoG flow". The rest of the feature space variables are filtered images, and the filters are described in Table 1. Further documentation of the filters is provided by [1].

The shallow network is a 2 layer, feedforward multilayer perceptron created using Matlab's `patternnet()` function. This classifier makes fast predictions and trains quickly on small data sets. The training pool is kept small by using scribble annotations. With these measures, patternnet can draw accurate conclusions about region boundaries faster than a human. Even though the classifier's predictions may need touching up by hand, this method gets the bulk of the work done without tedium.

The deep network used to create the final products is DeepLab v3+. It does not include x and y position in its feature space since explicit spatial data is redundant for this type of classifier. As a residual network, it is well-suited for problems of ranging complexity, which is ideal since the complexity of the user's data is unknown. Matlab also has a built in function to create this model.

The frames used to train DeepLab v3+ are chosen by an active learning implementation. This is done under the assumption that frames that are visually different will be good candidates for scribbling. Each image is reduced to a multidimensional point by calculating the mean of each feature for the pixels in that

frame. The images are then clustered into N groups, and the image closest to the centroid of each group is retrieved for the user to annotate. This creates a set of N training images.

A similar active learning implementation is also used to direct scribbling. At any point during scribbling, the user can choose to "show uncertainty". This displays an overlay image with low uncertainty pixels washed out while high uncertainty pixels are shown in full color. A box appears around the region with the highest uncertainty, showing live statistics during annotation of the pixels inside. The tool guides scribbling by suggesting that the user color 90% of the pixels inside the box. We chose to employ the uncertainty model presented in [7] due to its accuracy for k Nearest Neighbor classification using color imagery. The formula is shown in Eq. (1), where $U_a(\vec{p}_j)$ represents the uncertainty of feature vector $\vec{p}_j$, $C_a(\vec{p}_k)$ represents the class of near neighbor $\vec{p}_k$, $\Delta(\vec{p}_j, \vec{p}_k)$ represents the Euclidean distance between $\vec{p}_j$ and $\vec{p}_k$, and n represents the number of features.

$$U_a(\vec{p}_j) = 1 - \left| \frac{\sum_{k=1}^{K} \frac{2C_a(\vec{p}_k)-1}{(1+\Delta(\vec{p}_j,\vec{p}_k))^n}}{K} \right| \tag{1}$$

4. DATA SET

The flood dataset is a hand-shot video of a stream that formed after heavy rain. The water is shallow and, in some areas, clear to the bottom. The video was taken with a mounted camera. About 20 seconds into the video, the camera pans further up the stream. The video resolution is 1920x1080. 200 frames were sampled at intervals of 1/6 seconds. The video was acquired by scientists at the U.S. Naval Research Laboratory who are conducting research in bathymetric estimation from video sources.

5. RESULTS

The flood data set was processed using MIST to create a mask of the pixels belonging to water. Figure 1 shows the amount of manual pixel annotations required for 3 different labeling techniques in order to obtain a classification accuracy of at least 99.4%. The following techniques were explored:

- A: Annotate the entire video (all 200 frames) by hand
- B: Annotate 3 full frames by hand and use them to train DeepLab v3+
- C: Annotate 3 frames using the scribble-then-predict method and use them to train DeepLab v3+

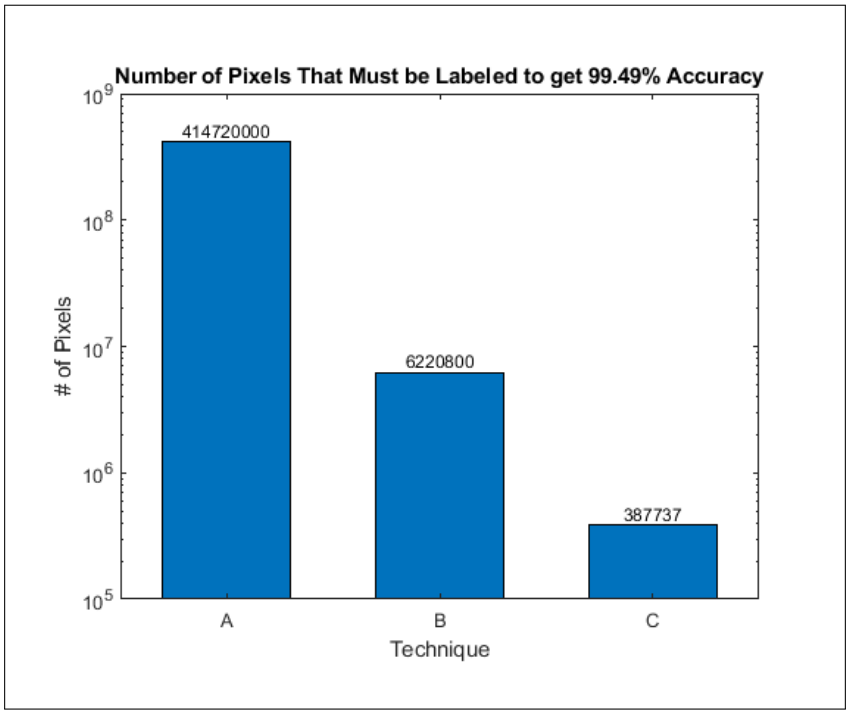


Fig. 1—Number of pixels that must be labeled to get 99.49% accuracy

For methods A & B, "# of pixels" was calculated by multiplying the video resolution by the number of frames to be annotated. For method C, "# of pixels" was calculated by summing the number of pixels that were scribbled and the number of pixels that were touched up (see Table 3).

Table 3—Breakdown of Pixel Count

Frame #	# Scribbled	# Touched Up	Total
48	108885	24052	132937
133	105619	16385	122004
178	115295	17501	132796
Total	329799	57938	387737

Technique C is employed by MIST, and requires the least amount of annotation. It requires the user to label 93.77% less pixels than technique B and 99.91% less than technique A.

99.49% accuracy was calculated using the following configuration:

Classifier:	DeepLab v3+
Training Frames:	48, 133, 178
Validation Frames:	5, 25, 45, 65, 85, 105, 125, 145, 165, 185

The classifier was trained on the fully annotated training frames, then used to predict the masks of the testing frames. The testing frames were also annotated by hand to compare to the predictions. The accuracy is the sum of all correctly labeled pixels divided by the number of pixels in the validation frames.

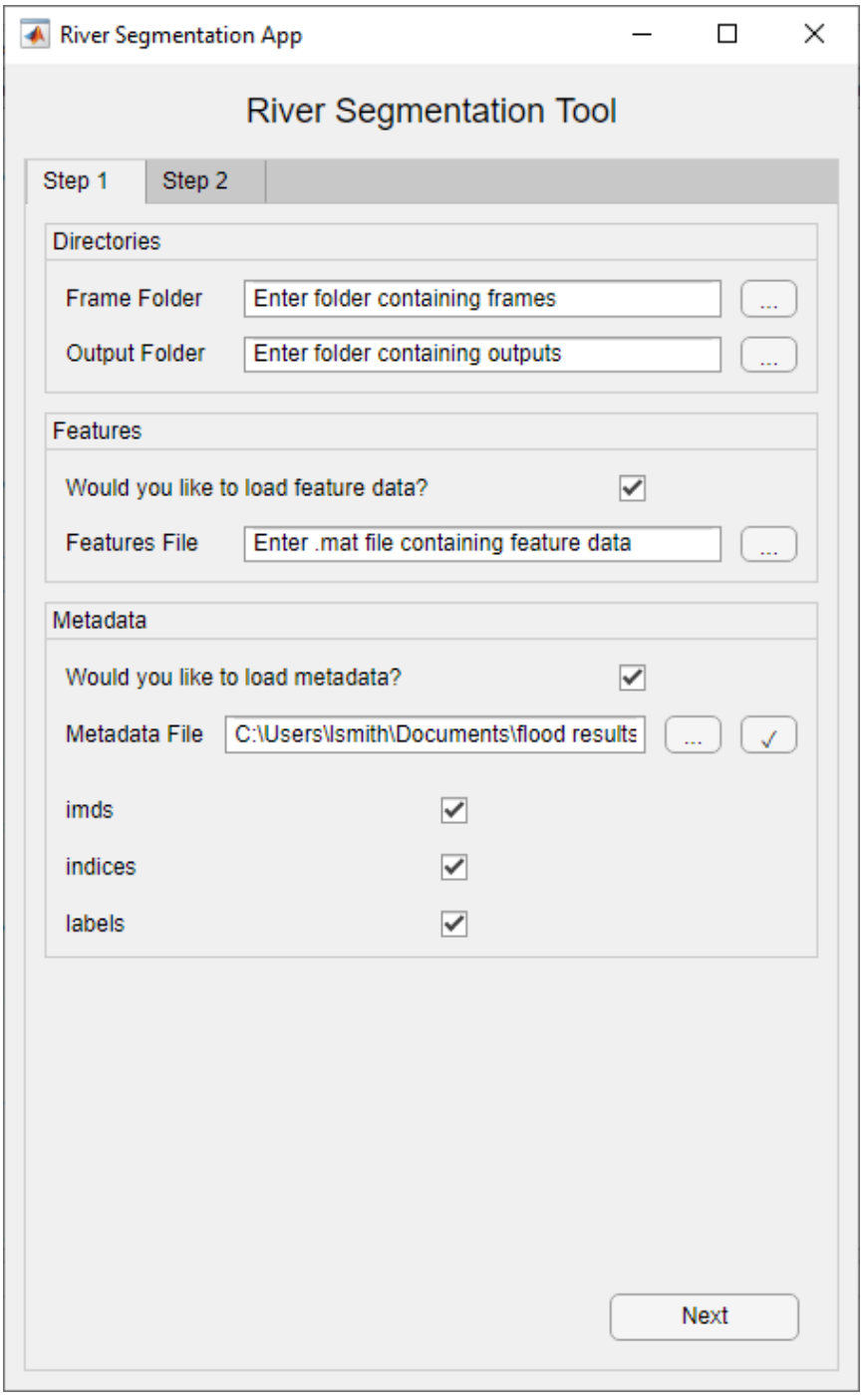
Fig. 2—Accuracy vs. # of training images

Accuracy was also tested for different numbers of training frames. Figure 2 shows accuracy with respect to the number of annotated images. It is important for users to understand how performance changes based on the amount of training data available.

Fig. 3—Frame 5 prediction (top image) and error (bottom image)

Misclassification is visualized in Fig. 3. The magenta overlay in the top image shows the pixels that were predicted to belong to water. The red overlay in the bottom image shows the pixels that were classified incorrectly. 99.56% of the pixels were classified correctly in this frame. The majority of misclassification occurs within a few pixels of the region boundary. This error may be negligible to the user.

6. MIST USER'S GUIDE



Step 1: Fill in the following information:

Directories Panel

- Frame Folder: Path to directory containing the frames
- Output Folder: Path to directory where outputs will be saved

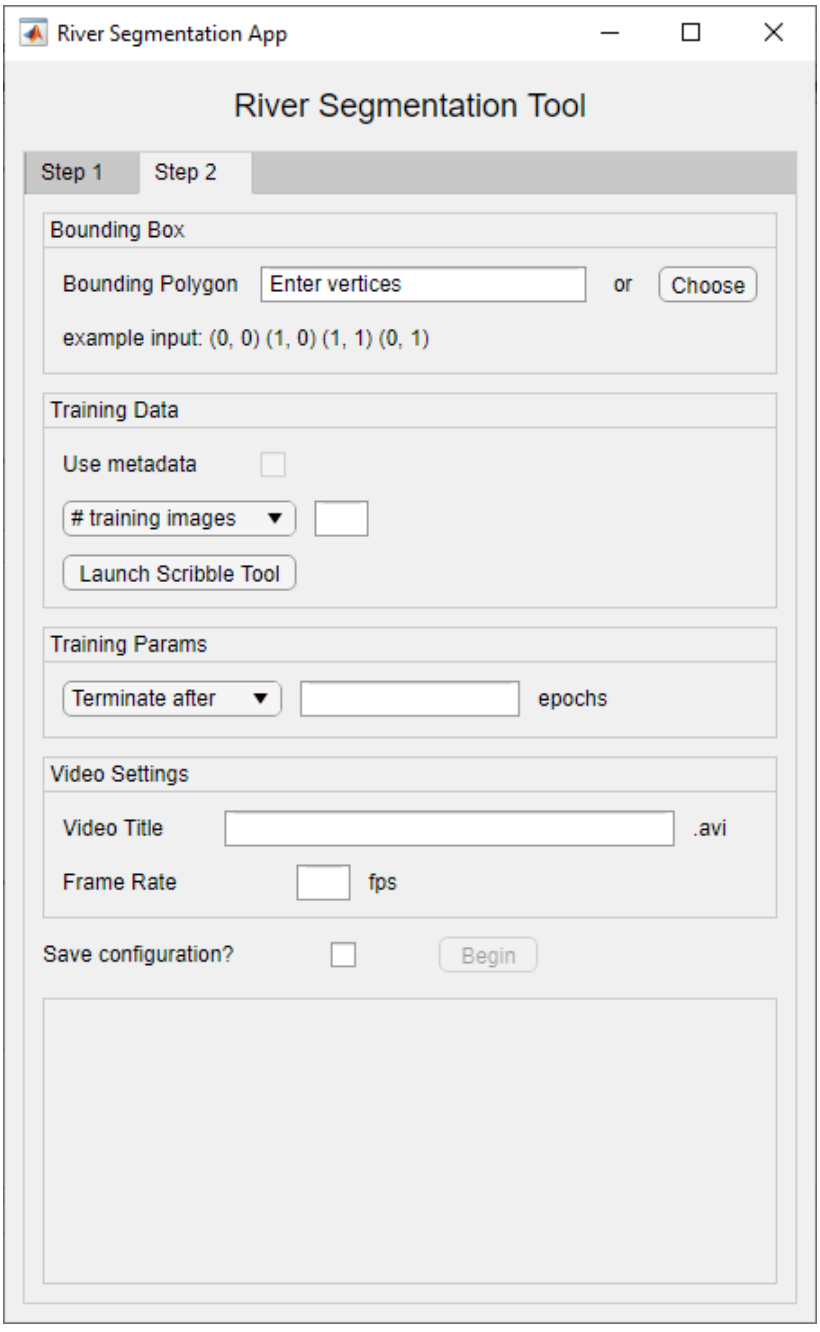
Features Panel

- Features File: (Optional) Path to the file containing optical flow and filter data

Metadata Panel

- Metadata File: (Optional) Path to the file containing metadata
- ✓ : Loads the metadata and displays the variables below
- Variable checkboxes: Choose whether to use or ignore each variable

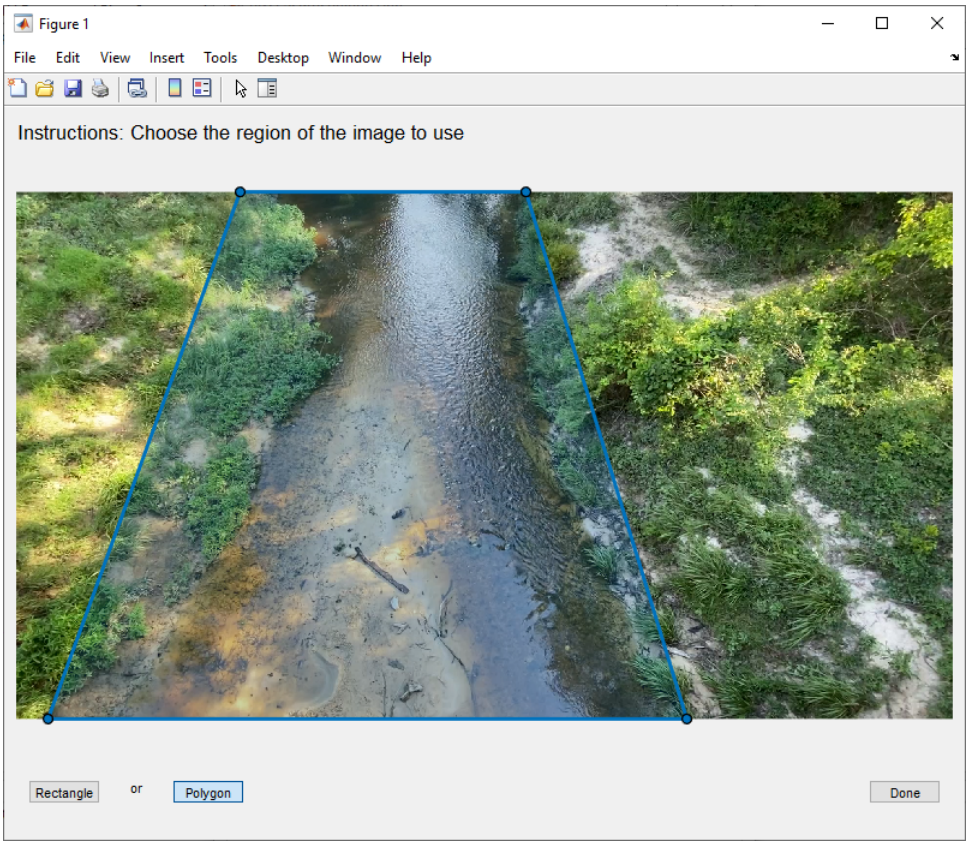
Once this information is filled out, click Next .



Step 2: Fill in the following information:

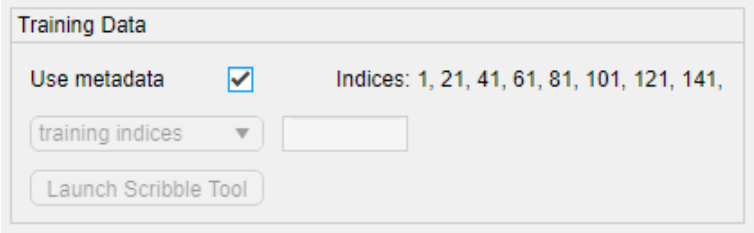
Bounding Box Panel

- Bounding Polygon: The vertices of the region of interest in the frame
- Choose : Opens a GUI (pictured below) to help with choosing vertices

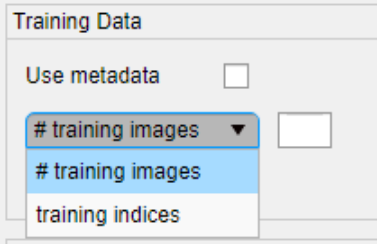


Training Data Panel

- Use Metadata: Option to use loaded-in annotations instead of re-making them

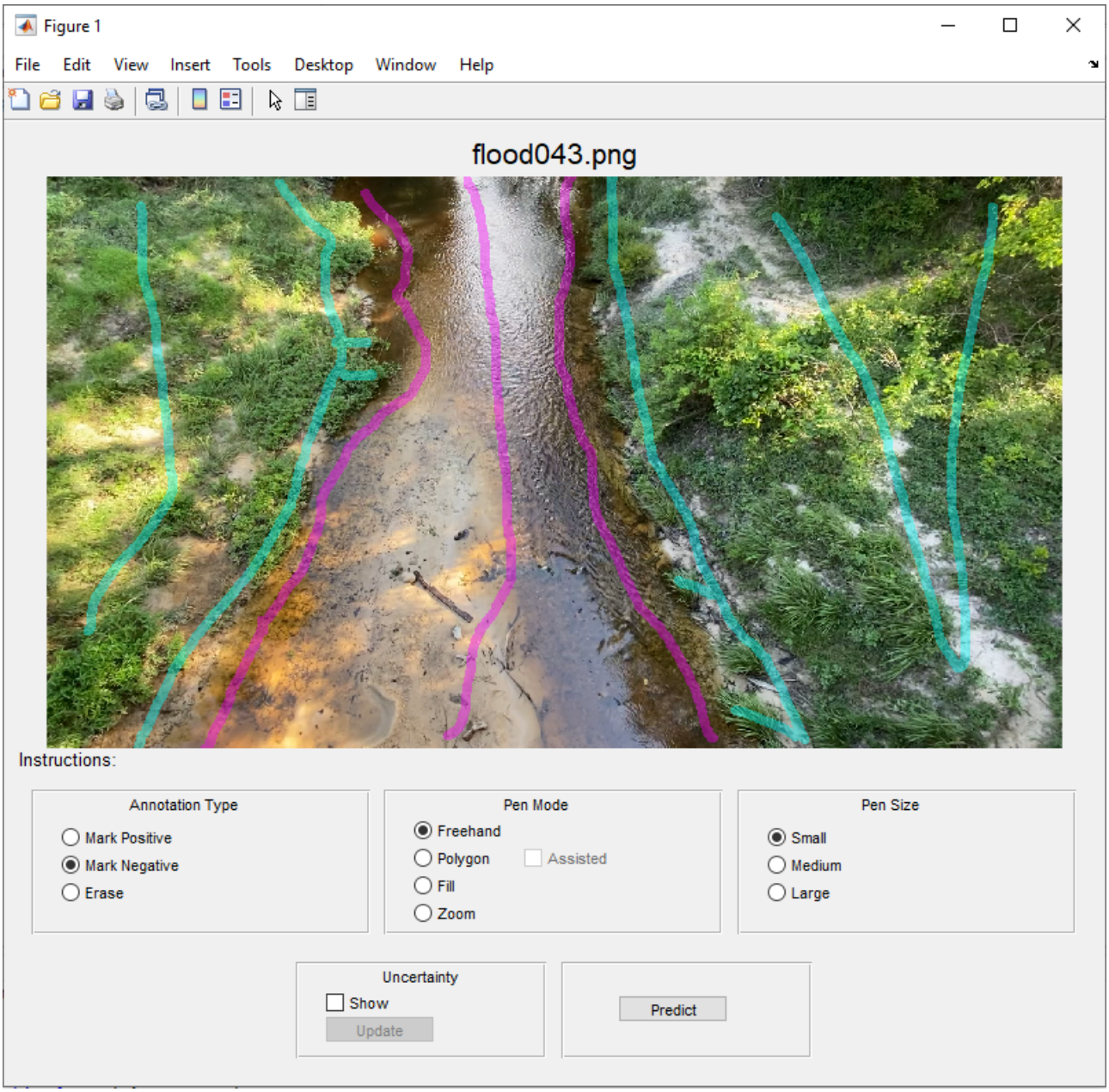


- # Training Images: The number of images to annotate and train on
- Training Indices: The exact indices of the images to use (overrides active learning)



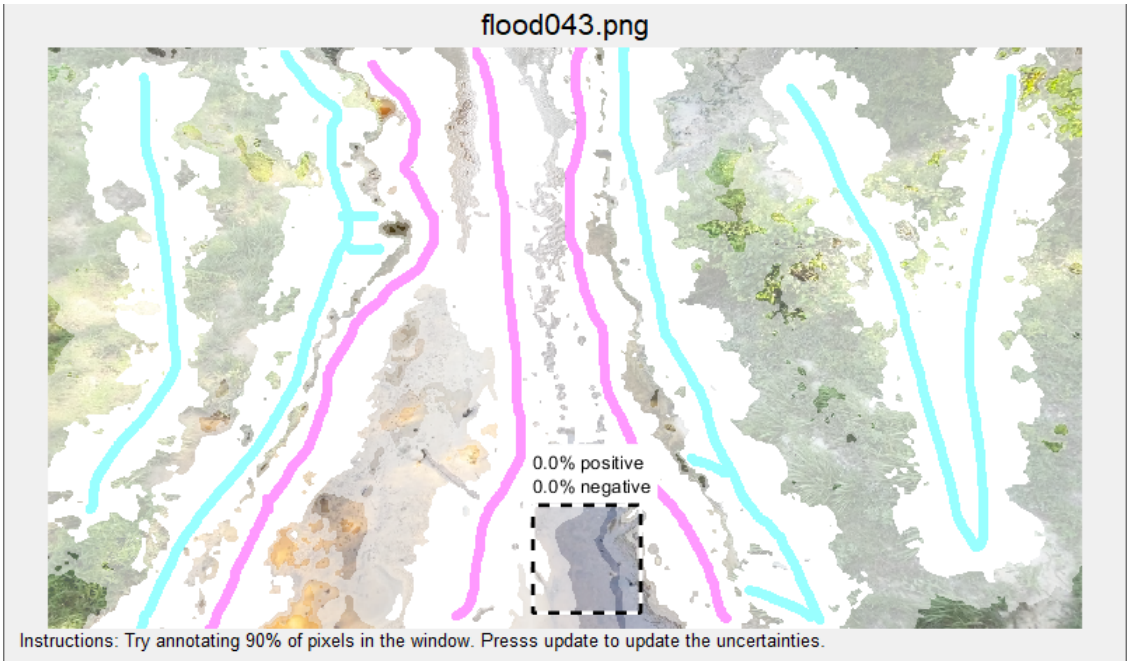
If not using metadata, click Launch Scribble Tool and perform the following steps.

Step 1: Scribbling

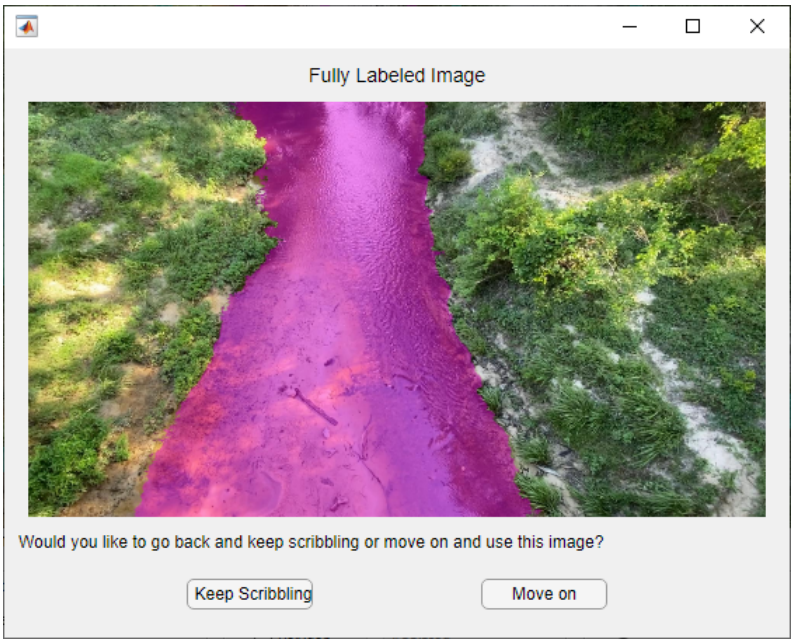


Color the positive examples of the desired class magenta and the negative examples cyan. For the best scribbles, use freehand mode to draw around the outside of each region. Then, color a small amount in the center of each region. Try to label an equal amount of area as magenta/ cyan near the region boundaries.

Choosing to show uncertainty will bring up the following view. Low uncertainty areas are colored white, while high uncertainty areas are in full color. A window of pixels with the highest uncertainty is outlined. The instructions read "try annotating 90% of pixels in the window. Press update to update uncertainties".

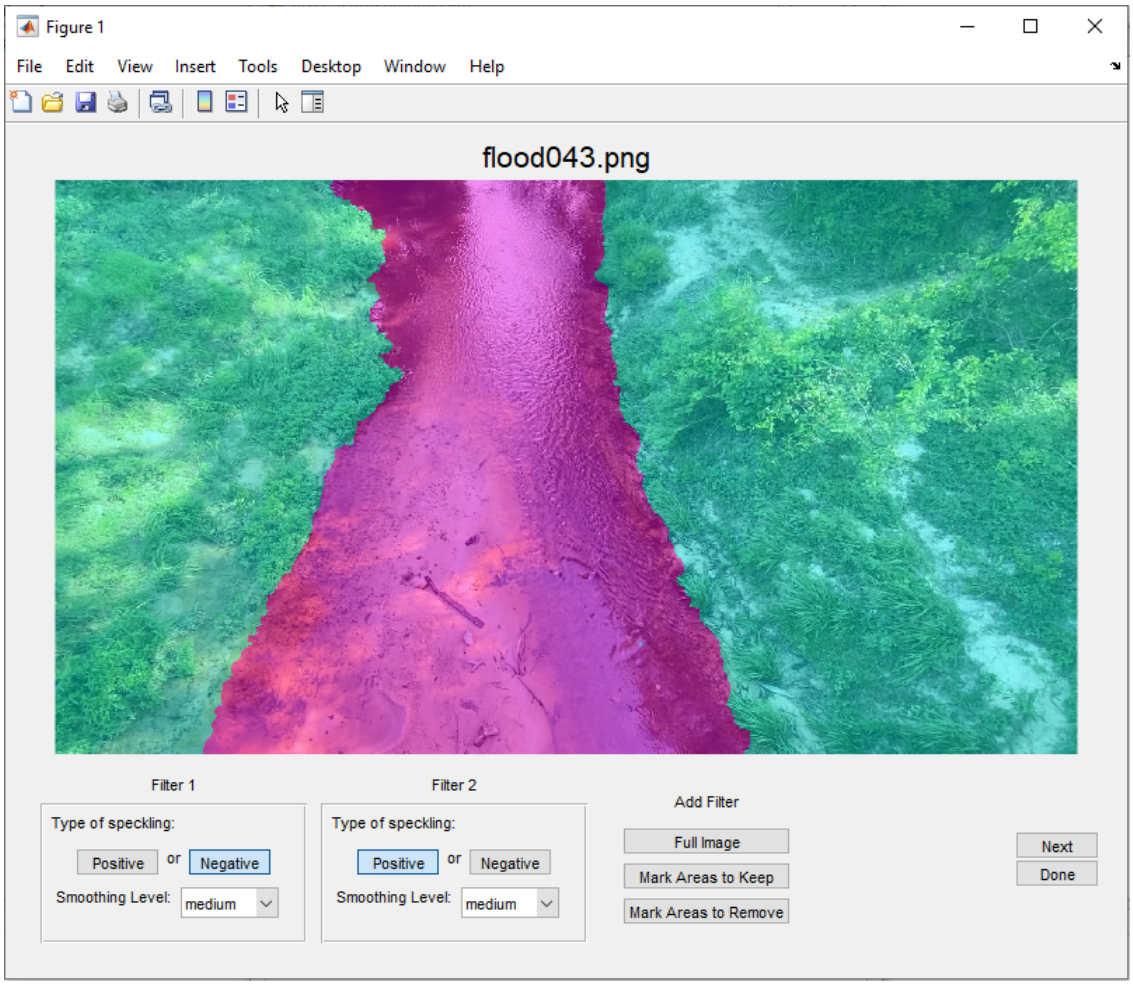


When satisfied with the scribbles, click Predict to fill in the rest of the image.



If the predictions are good enough, click move on . Else, click keep scribbling to go back.

Step 2: Smoothing

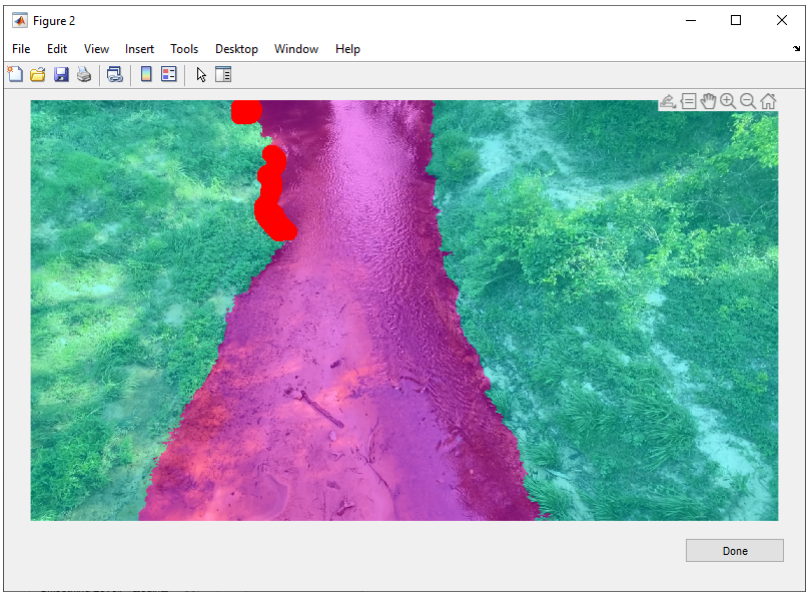


Once the user moves on, the smoothing tool will appear. The controls do the following:

Add Filter

- Full image: Perform smoothing on the full image
- Mark Areas to Keep: Perform smoothing on a specific region
- Mark Areas to Remove: Perform smoothing on the full image minus a specific region

The region to keep/ remove is chosen by coloring on the image.

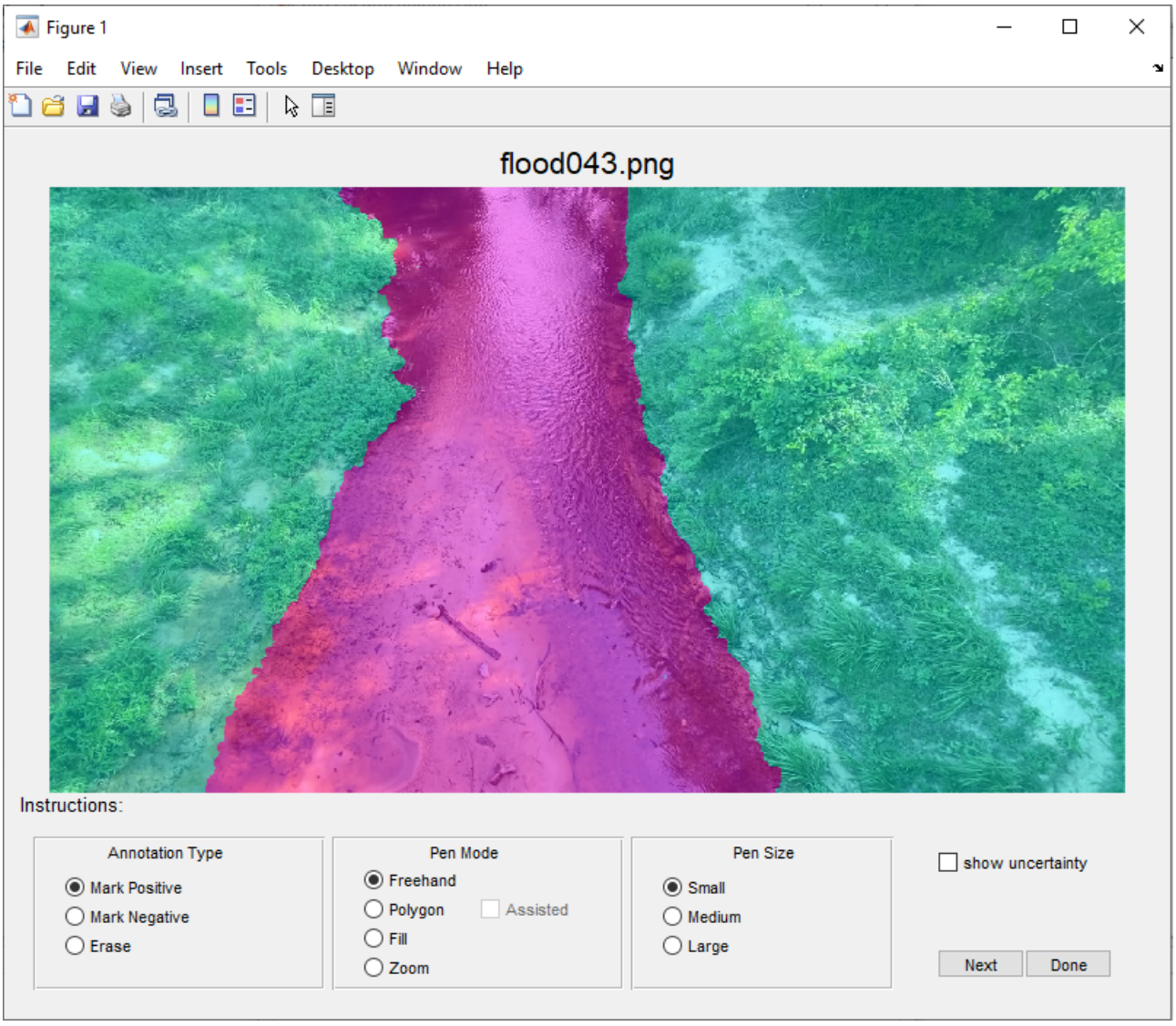


Type of Speckling

- Positive: Shrinks positive (magenta) speckles but enlarges negative (cyan) speckles
- Negative: Shrinks negative (cyan) speckles but enlarges positive (magenta) speckles

When done smoothing, click Next .

Step 3: Touching up

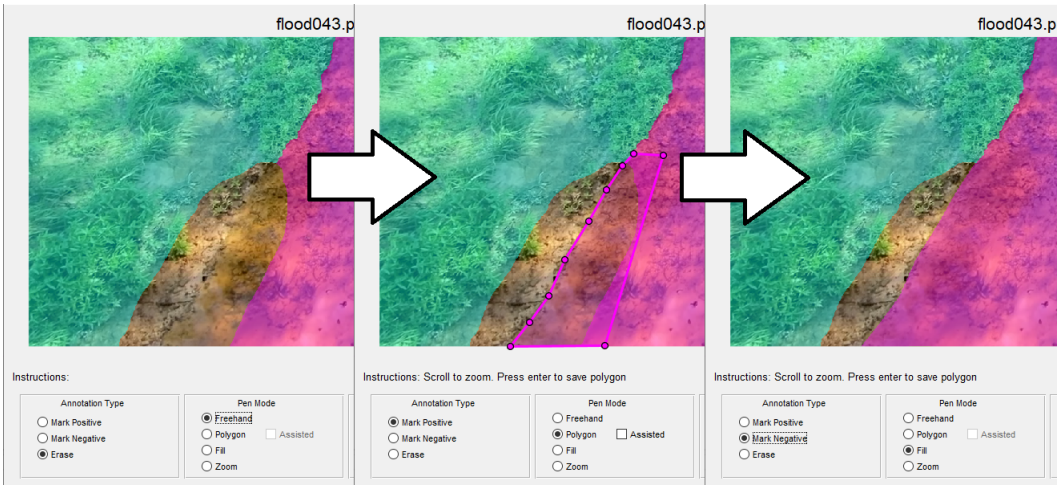


A GUI similar to the one from step 1 appears. It is used to adjust the predictions and make the labels accurate.

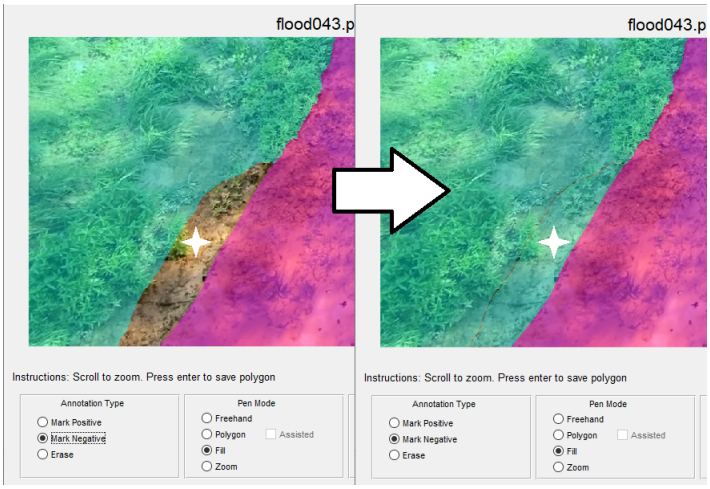
The pen modes do the following:

Freehand: Color on the image by dragging the mouse

Polygon: Select vertices of a polygon and drag to adjust. Press the enter key to commit.



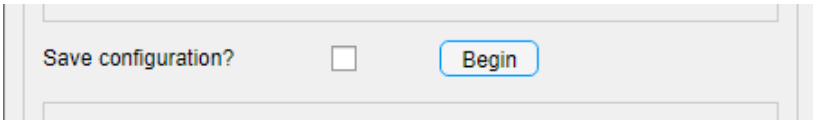
Fill: Select a region of the image and fill it with a color.



Zoom: Drawing is halted and zoom mode is enabled

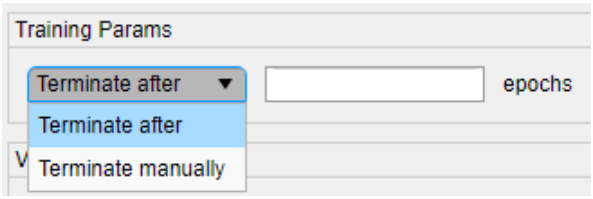
When finished touching up, repeat steps 1-3 for each image.

Begin is enabled after annotation is compete, or if “Use metadata” is selected.



Training Params Panel

- Terminate after: The number of epochs to train for.
- Terminate manually: Stop training after 50 epochs, or terminate sooner by clicking the stop icon on the training progress window.

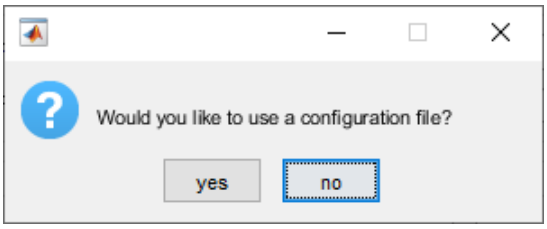


Video Settings Panel

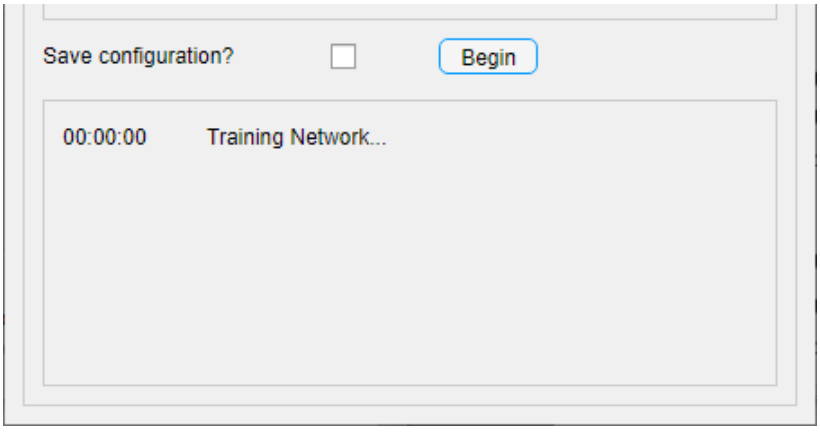
The tool creates a video from the predicted masks so that the user easily visualize them.

- Video Title: A name for the video product
- Frame Rate: The frame rate for the video product

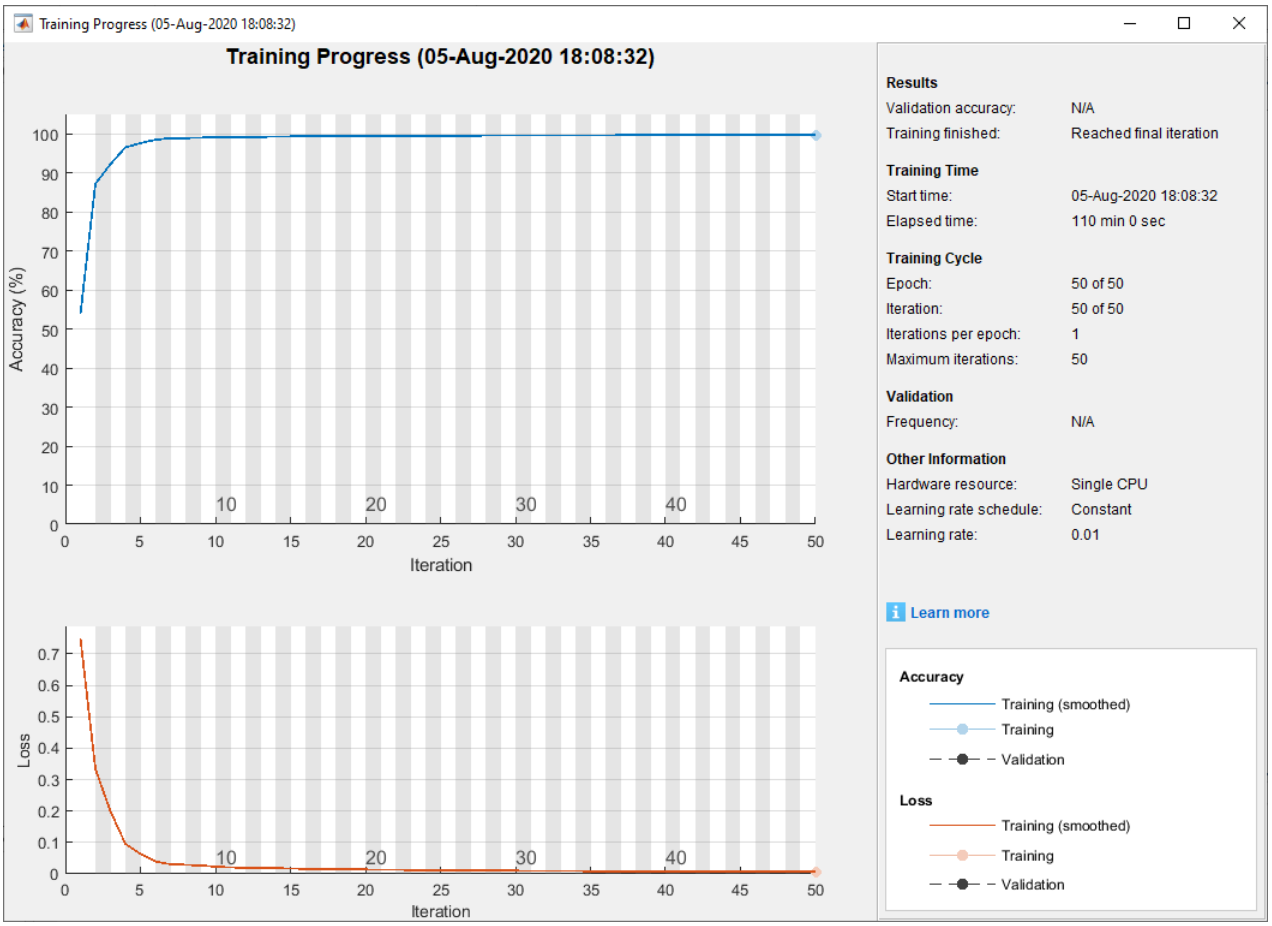
If "Save configuration?" is checked, then the next time upon opening the tool, the following dialogue box will appear. Clicking yes will pre-fill all of the textboxes and checkboxes.



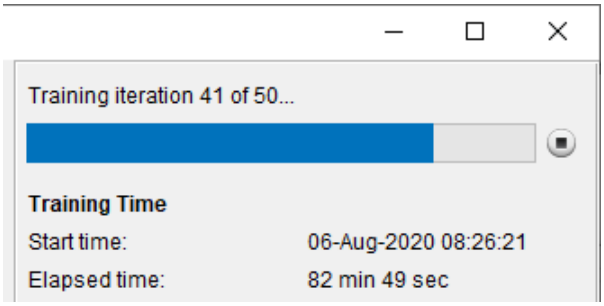
Once this information is filled in, click Begin . The bottom panel will display updates during the training/ predicting process.



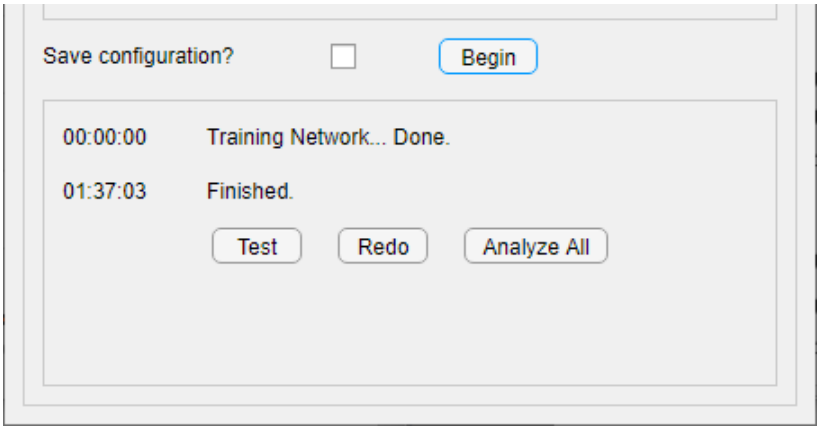
During training, a progress graph appears.



Training can be terminated early by pressing the stop button in the upper right-hand corner.



Once training is complete, the user has 3 options.

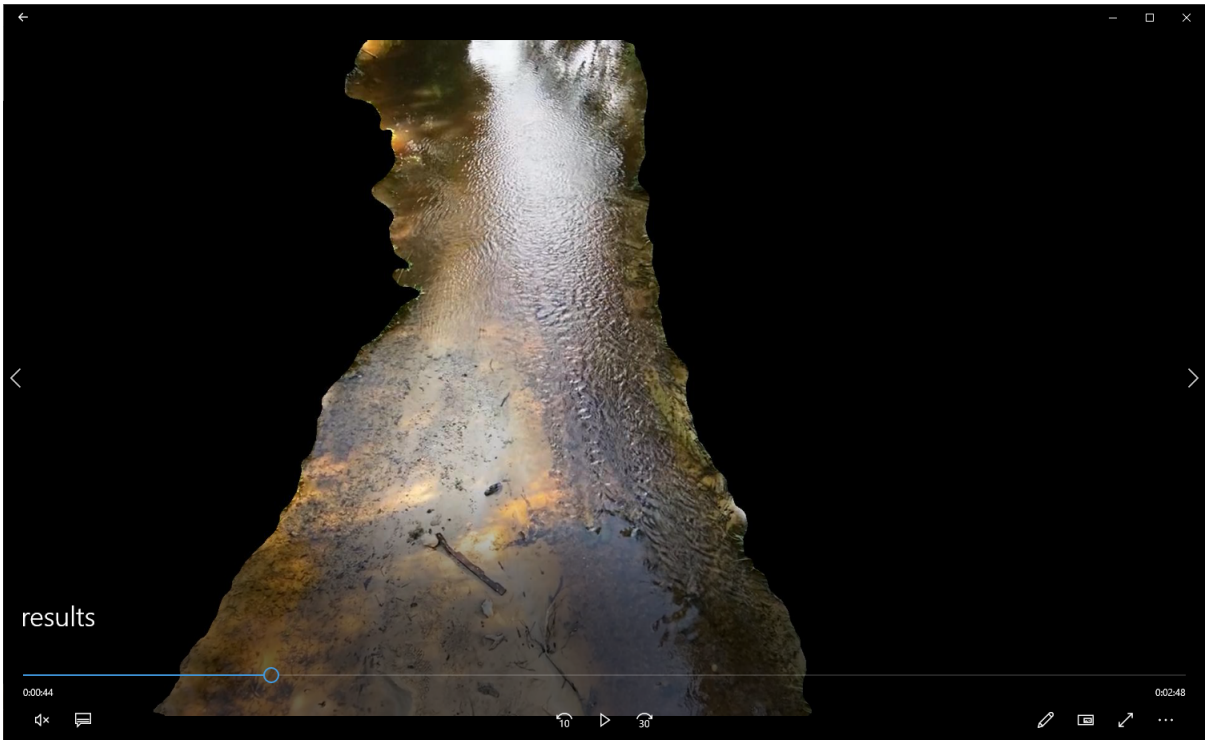


Test: The user can predict the masks for just a few images to test the network's accuracy.

Redo: The user can adjust any settings, and even re-make the annotations, then redo training.

Analyze All: The user can make the masks for the full video.

Choosing Analyze All is the final step for the tool. It will deposit the masks as .png files in a folder titled "results" within the outputs directory. It also creates a two videos so that the user easily visualize the result: one that masks out the non-class features and one that colors the class features magenta.



7. CONCLUSION

We have presented the the MIST application and provided a preliminary presentation of its performance. It was shown that MIST is able to adequately balance speed and accuracy by prioritizing minimal user input for annotation of training data through a scribble-then-predict system. Results show that the amount of manual pixel annotations required by the user is reduced by several orders of magnitude. Although speed is prioritized, the tool still requires fully annotated frames to ensure accuracy. The tool provides fast and accurate video pre-processing so that scientists may perform video analysis on more simplified imagery.

The authors would like to thank Joseph Calantoni, Blake Landry, and Carlo C. Zuniga Zamalloa for the project idea and the video sources.

REFERENCES

1. MATLAB, *version 9.8.0 (R2020a)* (The MathWorks Inc., Natick, Massachusetts, 2020).

2. J. A. Simeonov, K. T. Holland, and S. P. Anderson, "River Discharge and Bathymetry Estimation from Inversion of Surface Currents and Water Surface Elevation Observations," *Journal of Atmospheric and Oceanic Technology* **36**(1), 69–86 (2019), doi:10.1175/JTECH-D-18-0055.1.

3. J. A. Fails and D. R. Olsen Jr, "Interactive machine learning," Proceedings of the Proceedings of the 8th international conference on Intelligent user interfaces, 2003, pp. 39–45.

4. D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine Learning* **15**, 201–221 (1994), doi:10.1007/BF00993277.

5. B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Systems with Applications* **41**(4), 2065–2073 (2014).

6. L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence* **28**(4), 594–611 (2006).

7. C. J. Michael, S. M. Dennis, C. Maryan, S. Irving, and M. L. Palmsten, "A General Framework for Human-Machine Digitization of Geographic Regions from Remotely Sensed Imagery," Proceedings of the Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '19, New York, NY, USA (Association for Computing Machinery), 2019.

8. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, second ed. (Springer, New York, 2009), ISBN 978-0-387-84858-7.

9. J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks* **61**, 85–117 (2015), doi:10.1016/j.neunet.2014.09.003.

10. M. Hagan, H. Demuth, M. Beale, and O. De Jesús, *Neural Network Design*, second ed. (Martin Hagan, 2014), ISBN 978-0-9717321-1-7.

11. C. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics (Springer, New York, 2006), ISBN 978-0387-31073-2.

12. D. Lin, J. Dai, J. Jia, K. He, and J. Sun, "ScribbleSup: Scribble-Supervised Convolutional Networks for Semantic Segmentation," Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 3159–3167. doi:10.1109/CVPR.2016.344.

13. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

14. V. Dumoulin and F. Visin, "A Guide to Convolution Arithmetic for Deep Learning," *ArXiv e-prints* (2018). URL `arxiv.org/pdf/1603.07285.pdf`.

15. P. Napoletano, F. Piccoli, and R. Schettini, "Anomaly Detection in Nanofibrous Materials by CNN-Based Self-Similarity," *Sensors* **18**, 833–851 (2018), doi:10.3390/s18010209.

16. L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," Proceedings of the European Conference on Computer Vision (ECCV), 2018. doi:10.1007/978-3-030-01234-2_49.

17. J. Barron, D. Fleet, and S. Beauchemin, "Performance Of Optical Flow Techniques," *International Journal of Computer Vision* **12**, 43–77 (1994), doi:10.1007/BF01420984.

18. MATLAB, *opticalFlowHS* (The MathWorks Inc., Natick, Massachusetts, 2020). URL `www.mathworks.com/help/vision/ref/opticalflowhs.html`.

19. R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics* (6), 610–621 (1973).

20. R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Computer vision, graphics, and image processing* **29**(1), 100–132 (1985).

21. M. A. Balafar, A. R. Ramli, M. I. Saripan, and S. Mashohor, "Review of brain MRI image segmentation methods," *Artificial Intelligence Review* **33**(3), 261–274 (2010).

22. M. Liu, C. Wu, and Y. Zhang, "A review of traffic visual tracking technology," Proceedings of the 2008 International Conference on Audio, Language and Image Processing (IEEE), 2008, pp. 1016–1020.

23. M. D. Hossain and D. Chen, "Segmentation for Object-Based Image Analysis (OBIA): A review of algorithms and challenges from remote sensing perspective," *ISPRS Journal of Photogrammetry and Remote Sensing* **150**, 115–134 (2019).

24. G. Wang, W. Li, M. A. Zuluaga, R. Pratt, P. A. Patel, M. Aertsen, T. Doel, A. L. David, J. Deprest, S. Ourselin, et al., "Interactive medical image segmentation using deep learning with image-specific fine tuning," *IEEE transactions on medical imaging* **37**(7), 1562–1573 (2018).

25. A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A survey on deep learning techniques for image and video semantic segmentation," *Applied Soft Computing* **70**, 41–65 (2018).

26. M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3213–3223.