Video Segmentation

Ercan Alp Serteli

0067548

Table of Contents

Introduction

This report is prepared in the scope of COMP508 course project. Aim of the project is to design, develop and evaluate a segmentation algorithm for videos.

Image segmentation is one of the most fundamental in computer vision. Many vision problems can make use of, or even depend on purer images of objects isolated from their surroundings. Segmentation aims to solve that issue by partitioning an image into visually separated regions. Video segmentation is an application of image segmentation on sequences of images that are connected by time. While the output of video segmentation can be used for further analysis, it can also be used for entertainment and video production. There are many popular smart phone applications and webcam software that make use of real-time video segmentation to make the subjects look like they are in a different environment.

In the following sections, some background on the topic will be presented, methods used in the project and their implementation will be explained, the evaluation method and results will be shown and the report will be concluded.

Background

There are different kinds of video segmentation algorithms such as unsupervised, fully supervised, semi supervised or weakly supervised algorithms. I decided to aim for a semi supervised algorithm by taking user annotations at the first frame of the video for training, as done in this paper: [1]. There is also the difference between binary segmentation and multi-class segmentation. In binary segmentation, the images are segmented into background and foreground regions, whereas in multi-class segmentation the image may be segmented into many different regions. I chose to go with binary segmentation as it captures the essence of the problem without complicating parts like user annotation too much.

Image segmentation is a problem that researchers have been working on for a long time. Therefore, there are many different approaches to solve it, such as active contours, splitting and merging techniques, mean shift, normalized cuts, graph cuts, etc [2] . I have decided to use a generic classification method coupled with superpixels as a preprocessing method. This is because superpixels carry more information than pixels and they make the following processing faster as the number of superpixels is much less than the number of pixels [3]. The fact that they hold more information than pixels mean that they open the way for different algorithms to be easily used for segmentation. This information can include complex color or texture features, whereas a single pixel only holds a singular color information, making it difficult to use directly for segmentation. Computing superpixels has the disadvantage of adding the cost of computing them, but the SLIC superpixels technique [4] can construct them in a very efficient way, while conforming to the boundaries better than other techniques [5]. Performance is key in video segmentation, because a video is composed of numerous frames and each frame will need to be processed for segmentation, requiring much more processing time than the segmentation of a single image.

After the superpixel step, I decided to use Support Vector Machines (SVM) for classification. SVM is a very well-known algorithm used in many areas including computer vision, and is “currently considered to be the best off the-shelf learning algorithm” [6]. SVM is also known to learn well

with even a small number of training samples. This is important for the problem at hand because the training samples will depend on user annotation.

SLIC Superpixels

Simple Linear Iterative Clustering (SLIC) is a surprisingly simple algorithm. It performs a k-means based local clustering on pixels in the 5-D l,a,b,x,y space. The l,a,b features are the representation of the pixel in CIELAB color space and x,y features are its position in the image. Since the features are of different kinds, it does not make sense to use Euclidean distance in this space. Instead, SLIC uses its own distance measure D_s defined as such: [4]

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$
$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$
$$D_s = d_{lab} + \frac{m}{S} d_{xy} \ ,$$

Let N be the number of pixels in the image and K the number of superpixels, then S is defined as:

$$S = \sqrt{N/K}$$

The m parameter is introduced in order to control the compactness of superpixels. As m is increased, proximity becomes more important than color similarity and the superpixels get more compact. [4]

At the beginning, all superpixels are equally sized and their centers are S pixels apart. They are adjusted to a local lower gradient position to lower the probability of starting at an edge or on a noise pixel. Then the cluster memberships and cluster centers are iteratively updated until convergence. At the last step, connectivity of labels is enforced to get rid of stray labels. [4]

SVM Classification

SVM is a kernel machine that uses a subset of the training samples (called support vectors) to find a linear model that separates classes. For classification, the support vectors are the samples that are close to the class boundaries. [6] The method called kernel trick is used to substitute the dot product with a different function in order to extend the algorithm to different kinds of non-linear functions. The trick is to bypass intermediary computations to directly compute the required result. In this way, SVM can fit complex models easily. Examples of kernel functions that might be used are:

- Polynomial kernel
- Gaussian kernel
- Radial basis functions
- Sigmoid kernel

An advantage of kernel machines is that the problem has a global optimum, so there is no need for iterations or worrying about convergence. [6]

Methods

The methods designed for this project can be summarized as such:

- Training phase
  - Getting the first frame of the video
  - Oversegmenting it into superpixels using SLIC
  - Letting the user partially annotate the superpixels
  - Extracting features from the superpixels
  - Feeding the feature matrix into SVM to train it and adjust its hyper parameters
- Testing phase
  - Getting a frame from the video
  - Oversegmenting it into superpixels using SLIC
  - Predicting each superpixel as background or foreground using the SVM
  - Creating a mask with the predicted superpixels
  - Outputting the resulting image to a file and/or the screen

On top of these steps, some additional steps have been added for improvement:

- After the prediction step, information from a previous frame is used to adjust predictions by making use of the temporal inertia of regions in the image
- Again after the prediction step, some superpixels that have no neighbors of the same label are suppressed in an attempt to reduce noisiness
- Using direct real-time video stream as input instead of a video file
- Applying a new background to the output if wanted

Steps that need clarification are explained as follows:

User Annotation

The annotation phase is designed as such:

- The user is shown an image with the superpixel borders visible
- The user can use left and right mouse buttons to click or drag over superpixels to label them
- While the left and right buttons can label negative and positive, the middle mouse button is used to erase a label that was erroneously made
- After the user is done, they can press the return button on the keyboard to finish annotating

As long as the cursor touches any pixel of a superpixel, it will be considered as labeled. This makes it easy for the user to annotate border regions without getting too close to the border. While the user does not need to annotate the whole image, it is assumed that they will annotate at least an example

of each distinctly colored/textured region in the image. Figure 1 shows an example of the user annotation window.

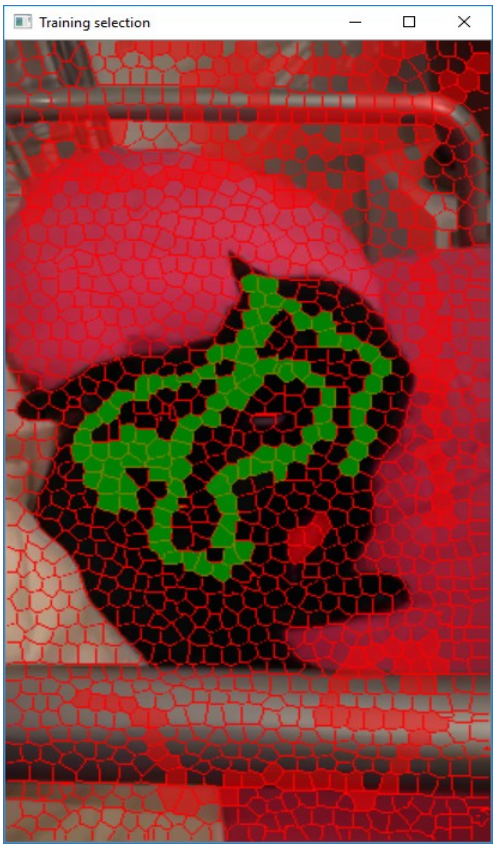

Figure 1: User annotation interface. Red superpixels are labeled as background and green superpixels are labeled as foreground by the user

Another approach to the annotation interface could be to let the user draw safe regions for background and foreground and use the generated masks to label superpixels. For example, the centroid of a superpixel could be checked in the mask to find its label. This might be a better approach in an end-user product, since it could also hide the underlying superpixels from the user. However it might result in less sensitive annotations, especially on the borders.

Feature Extraction

In order to apply classification on superpixels, features need to be extracted from them. Many possible feature combinations exist, but a good combination is including color and texture information. I decided to use RGB histogram and mean color to encode color information and LBP histogram to encode texture information. Other options exist such as using HSV or CIELAB color spaces instead of RGB, or using HOG or SIFT for texture/image features, but a decision had to be made. In the end, the biggest reason for these decisions is the ease of implementation since there was not enough time to try them all.

Mean color is simply the mean of the RGB values of all pixels in the superpixel. It is added as an aid to the RGB histogram because it is an intuitively meaningful feature.

RGB Histogram

The RGB (or BGR due to implementation) histogram is a three dimensional histogram that puts pixels into bins depending on their color values. The bin width can be adjusted for more or less granular information. However the number of features generated is the cube of bin width due to the 3D nature of the histogram. For example the number of features generated for 6 is 216 whereas for 16 it is 4096. High bin widths slow down the implementation too much, so 6 was chosen as a good balance of information and speed. The histogram is normalized after counting is done.

LBP Histogram

LBP (Local Binary Patterns) is a visual descriptor used to encode local texture information. In order to compute the LBP features, first a gray scale version of the image is created. LBP for a pixel is basically computed as an 8-bit value where each of the bits correspond to the comparison result between the pixel and one of its 8 neighbors. If neighbor is greater than or equal to the pixel, the bit is 1, otherwise 0. Figure 2 shows a representation of how it is computed. In this example, comparisons are done left to right, top to bottom.

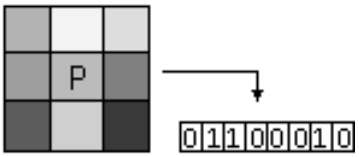

Figure 2: Computing an LBP Value

Normally the histogram is computed for an image by dividing the image into square cells and computing histograms for each cell. However, in this problem we need to compute an histogram for each superpixel, so no cells are needed. The histogram is simply computed using the LBP value of each pixel in a superpixel. The LBP histogram of a superpixel in the end shows a distribution of the kind of gradients that are prevalent in a superpixel, and that is presumably a useful texture information.

SLIC

SLIC needs an average superpixel size (or the number of superpixels, which is inversely proportional to that) and a compactness factor to work. I decided to manually choose these at depending on the video, since the size of the video or the closeness of objects matter when choosing the size. If a small size is given in a large resolution video, it may slow the implementation badly, but a large size may result in small objects not being oversegmented enough or well.

SVM

Before an SVM model is trained, some parameters need to be chosen. For the kernel type, I chose radial basis functions since they are known to be a good choice in many cases. A linear kernel (meaning no kernel) would not work well since we can be fairly sure that the superpixels in a real image will likely not be linearly separable. RBF is a good non-linear kernel in general use.

SVM also needs a penalty factor parameter to model non-separable cases. Instead of defining a value beforehand, I decided to use k-fold cross validation on the training set to determine the value automatically. Since this is only done at the training stage, it should not slow down the overall implementation.

Temporal Inertia

It is possible to assume that frames in a video follow one after the other in a logical order, time increasing linearly with each frame. This means that a region that was somewhat deep in a background region is likely to be background in the next frame, and vice versa. So, it can be said that the regions have an inertia to their label through time.

In order to make use of this assumption, the foreground and background masks of a previous frame can be morphologically eroded (white regions made smaller to avoid affecting border regions) and stored for the next frame's use. At the next frame, the computed superpixels can be checked to see if they lie at a location that was deep in a background or foreground region. This check can be simply and quickly done using the centroid position of the superpixel and getting the value that corresponds to that position in each of the masks. If a superpixel is found to be deep inside a foreground region, than its probability to be found as positive is increased, and vice versa. A demonstration of this can be seen in Figure 3.

The erosion is a relatively cheap filtering operation and its amount can be tuned to adjust the meaning of "being deep inside a region".

Increasing the probability can be done by getting "distance from discriminator" as the results from the SVM instead of labels, and adding a value to those results before labeling. Normally, a distance that is positive means that the sample is labeled positive and negative means that it is labeled as negative.

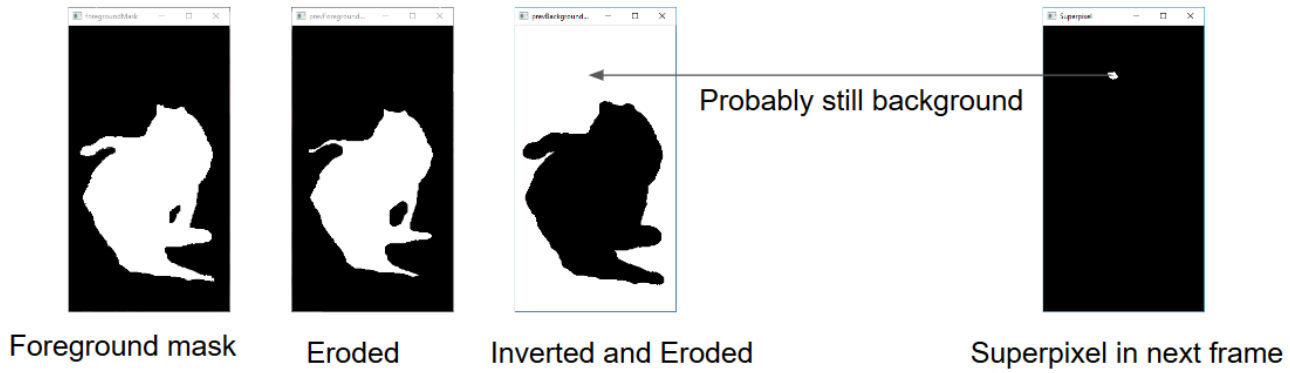

Figure 3: A superpixel in the new frame is found to be deep inside a background region at the previous frame

Noise Reduction

Since each superpixel is classified separately, there may be incorrectly classified superpixels in the middle of nowhere, reducing the noise in the result. Some of these superpixels can be detected by checking if there are any "friendly" neighbors around it. The assumption here is that a single superpixel is too small to be a complete foreground or background region, therefore it must have neighbors of the same label around it to not be considered as noise. So, if a superpixel does not have

any friends around it, it is converted to the opposite label. A demonstration of this can be seen in Figure 4.

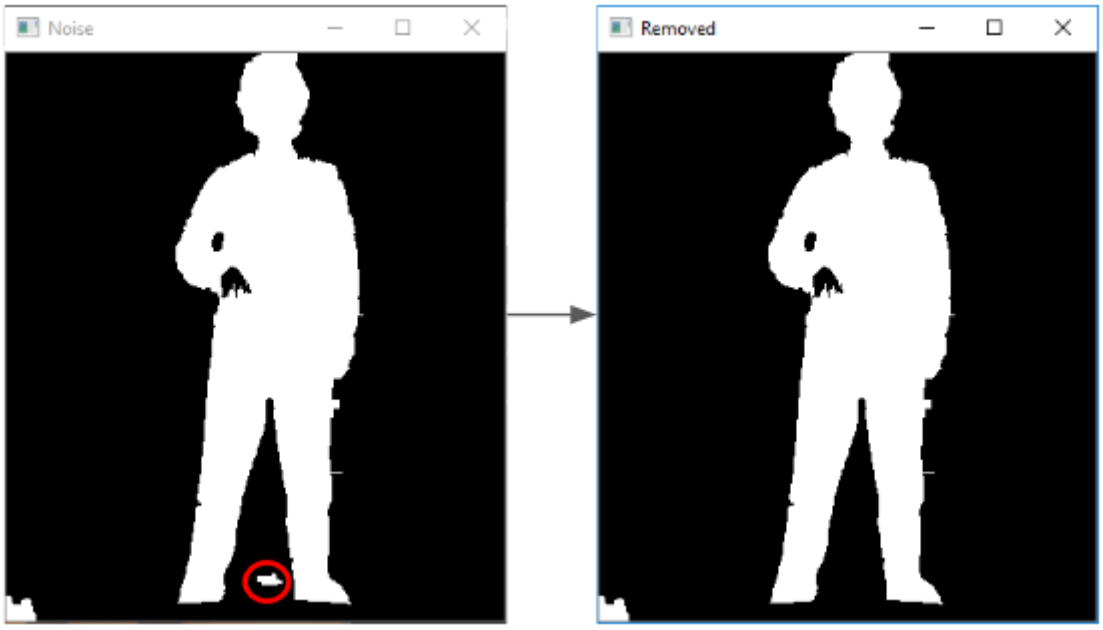

Figure 4: A suerpixel is considered as noise since it has no friends around

One way to detect these superpixels would be to build a neighborhood graph of all superpixels and use that. However, that would be very expensive to do at every frame, so instead a simple morphological method is designed. For each superpixel, its mask is dilated and the original is subtracted from the dilated version. This creates the outline mask of the superpixel. This outline mask is anded with the corresponding region in the foreground or background mask depending on the label of the superpixel. If the anded version is completely black, that means there are no friendly pixels around this superpixel, therefore no friendly superpixels either. This is also visually explained in Figure 5.

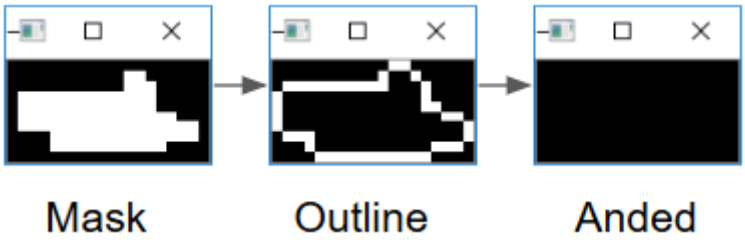

Figure 5: Showing the steps of finding that there are no friends of the superpixel in Figure 3

All the operations used in this feature are very simple and fast to not incur a performance penalty. And the operations only act on very small, superpixel sized binary masks. Paying attention to performance is a recurring theme in this project due to the nature of video segmentation.

Convenience Features

It is difficult and annoying to have to annotate a video over and over again when testing, so in order to make it easier, some convenience features are added.

- After training for a video, the SVM model is exported to an XML file. This file can later be given as a parameter to bypass the annotation and training procedures on the same video.
- If the features or some other parameters are changed between tests, the old SVM model will not work. For these cases, the ids of the labeled superpixels and their labels are stored on a text file. This text file can be later used to bypass the annotation procedure. But this will only work as long as the SLIC parameters are not changed.
- When densely annotating a video for optimal results, it can be difficult to annotate a small foreground and than having to annotate every other superpixel as background. For these cases, pressing F during annotation is made to fill the entire unlabeled superpixels as background. This saves time and wrist health for the user.

Implementation

The project is implemented in C++, using the OpenCV library. The “filesystem” feature of C++ 17 is utilized in the automated evaluation code, so the project requires a compiler that supports C++ 17. OpenCV is used with ffmpeg to make use of video reading and writing features.

Version of OpenCV used is 3.4.5. OpenCV was manually compiled along with opencv_contrib, so the project will not work on a normal OpenCV installation.

OpenCV Features Utilized

- Every image processing code is done using the *Mat* implementation.
- All the basic operations such as matrix bit wise operations, masking, ROIs, filtering, morphological operations, histogram generation
- Video reading and writing using *VideoCapture* and *VideoWriter* constructs.
- Reading from camera stream is made very simple with *VideoCapture* as well
- Image reading and writing such as *imread*, *imwrite*, *imshow*
- SVM implementation in the machine learning module
- SLIC implementation in the ximgproc module
- For the parallelization of feature computation of superpixels, the *ParallelLoopBody* feature

The SLIC implementation is an extra feature that is only in opencv_contrib, and not on the main version. Therefore, the opencv and opencv_contrib repositories were downloaded and manually compiled to make use of SLIC.

General Architecture

The project architecture is designed as a loosely object oriented system. The main construct is a class called *VideoSegmenter*. This class uses a *Superpixel* class and OpenCV features for its functionality. A main function creates an object of the VideoSegmenter type and creates a scaffold to make use of it in the following manners:

- Segment a video manually (the default functionality)

- Segment a real time camera stream
- Run automated evaluation

The Superpixel class holds a vector of pixels and computes and stores its features, including its centroid location and class label.

Annotation interface is implemented using OpenCV's window functions, including binding a mouse event.

Evaluation

The project is evaluated on the DAVIS 2016 dataset. DAVIS (Densely Annotated VIdeo Segmentation) is an organization for "In-depth analysis of the state-of-the-art in video object segmentation" [7] and provides a free dataset for the evaluation and benchmarking of video segmentation algorithms. The newer datasets are based on multi-label segmentation, that is why the 2016 version is used in this project.

The dataset consists of 50 videos (exported as image sequences) and their manual annotations as black and white binary images. Each video is available in 1080p and 480p resolutions. In this project, 480p versions are used for faster results.

Evaluation Method

The evaluation is completely automated. The dataset files are foldered in the filesystem as and ".../JPEGImages/480p/<name>/" ".../Annotations/480p/<name>/". In every such folder, the frames are named as "00000.jpg", "00001.jpg", etc.

Using the C++ 17 filesystem API, the list of sample names are extracted. For each name, the frames are taken as a video stream using the format string "%05d.jpg" for frames and "%05d.png" for annotations. The first annotation frame is used for training annotation, just like how a manual segmentation works. In this case, the superpixels whose centroids correspond to a white pixel in the annotation image are labeled as foreground and the others are labeled as background. After the training is done, segmentation commences as usual, but the results are written as PNG images in the same naming format as the manual annotations.

After the segmentation is done, another function performs the evaluation by comparing the results with the manual annotations. Two evaluation metrics are used:

- Dice score
- Jaccard index

These metrics are commonly used for segmentation evaluation. In the DAVIS challenge, Jaccard is used as well, so this allows comparison with the algorithms on their website.

Dice score is calculated as 2 times the area of intersection divided by the sum of areas. Jaccard is a similar metric that is calculated as the area of intersection divided by the area of the union. These scores return a similarity measure of the predicted and ground truth segmentations, so a value closer to 1 is good, whereas closer to 0 is bad.

A score is calculated for each frame of the video, so the actual score is calculated as the average of the score of all frames in a video.

Results

The results of running the algorithm on each sample are given below:

Name of sample	Dice score	Jaccard index
bear	0.835751	0.719164
blackswan	0.857412	0.75226
bmx-bumps	0.445926	0.323918
bmx-trees	0.399285	0.255957
boat	0.341597	0.206192
breakdance	0.627274	0.460104
breakdance-flare	0.707827	0.549685
bus	0.837937	0.728791
camel	0.561608	0.39645
car-roundabout	0.257521	0.148308
car-shadow	0.489819	0.347612
car-turn	0.293788	0.178525
cows	0.773713	0.632201
dance-jump	0.547253	0.381545
dance-twirl	0.32778	0.196737
dog	0.57669	0.415282
dog-agility	0.543805	0.377234
drift-chicane	0.0337383	0.0177236
drift-straight	0.0166238	0.00865827
drift-turn	0.170439	0.0996587
elephant	0.423284	0.269074
flamingo	0.293306	0.17246
goat	0.544386	0.374652
hike	0.655441	0.489101
hockey	0.60813	0.439677
horsejump-high	0.645197	0.486936
horsejump-low	0.612341	0.442136
kite-surf	0.710069	0.552675
kite-walk	0.839971	0.725464
libby	0.584041	0.429596
lucia	0.787211	0.650269
mallard-fly	0.364794	0.250115
mallard-water	0.681238	0.52728
motocross-bumps	0.217032	0.127159
motocross-jump	0.253224	0.149583
motorbike	0.505872	0.347185
paragliding	0.913546	0.841657
paragliding-launch	0.677005	0.528466
parkour	0.438558	0.286284
rhino	0.459005	0.298871
rollerblade	0.575499	0.40667
scooter-black	0.0539298	0.0280011
scooter-gray	0.267327	0.156604
soapbox	0.141286	0.0778644

soccerball	0.758866	0.619624
stroller	0.771538	0.628915
surf	0.798601	0.67993
swing	0.631297	0.46627
tennis	0.704756	0.545698
train	0.592788	0.423987

The histograms of these scores are given in the following figures 6 and 7:

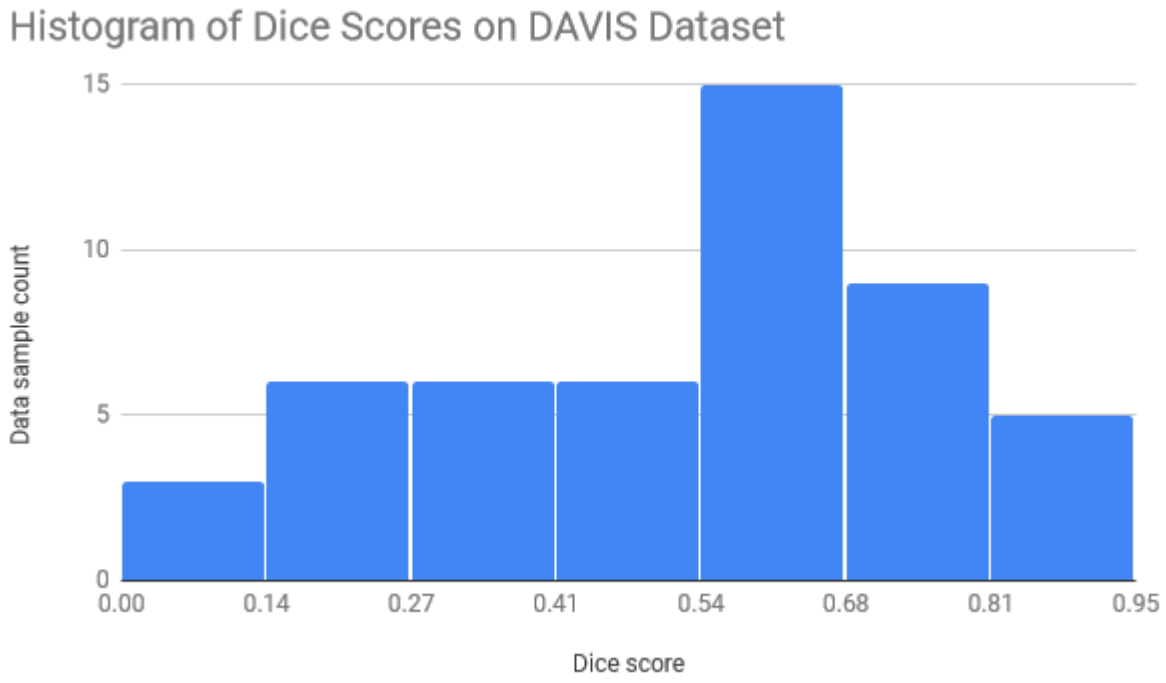

Figure 6: Histogram of Dice scores

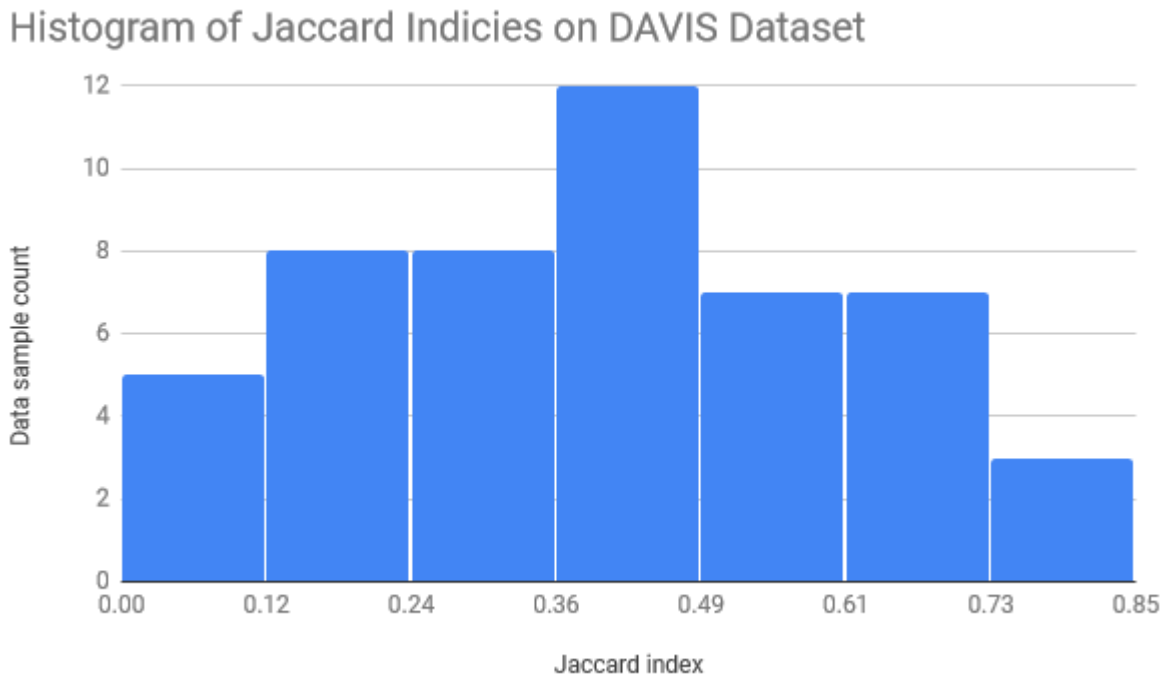

Figure 7: Histogram of Jaccard indices

As can be seen from the histograms, the distribution of scores has a high variance, with some samples getting very good scores (>0.8) and some getting terrible scores (<0.1). Overall, it seems that the Dice scores are higher than Jaccard scores and the Jaccard scores resemble more of a normal distribution, whereas Dice scores are clumped asymmetrically and there is a very large peak.

Overall, the mean Jaccard index is around 0.4. In the benchmark section of the DAVIS 2016 challenge, the mean Jaccard indices of semi-supervised algorithms that competed go from 0.849 to 0.504 [8], so they all did better than the implementation of this project. That said, the focus of the challenge is video *object* segmentation, and in the dataset samples, a singular object is segmented. This project however does not make the assumption of a single moving object, but looks for possible foreground regions around the whole frame. The accuracy on this dataset could be increased by making that assumption and only keeping the biggest connected components in the final mask.

Limitations

Testing of the algorithm showcases some limitations that are present. These can be listed as follows:

- If the foreground object, or some part of it is very similar in color and texture to some part of the background, it confuses the classifier and results in bad segmentation. This is due to the data points in the feature space not being separable.
- If the color or texture of a foreground object changes too much over time, for example due to differing lighting conditions or the object getting closer/farther as time goes on, the classifier may not recognize it as foreground anymore. This is due to the assumption that the regions will look similar from the beginning of the video to the end of it, since training is only conducted at the beginning.

- An unexpected object entering the scene may result in bad segmentation, if the object looks different from anything that was present during training. This is due to the classifier not knowing what the new object is supposed to be, since it was not trained for it.

Another limitation of this algorithm is that it is designed for binary segmentation, so it cannot be used to segment a video into more than two regions. However, adding multi-class support would not be very difficult, as the underlying SVM implementation supports multi-class classification, and that is the main place that labeling is done. The more difficult parts of this conversion would be changing the annotation interface, and updating the temporal inertia feature.

Conclusion

In the scope of this project, a video segmentation scheme has been designed, developed and evaluated on a public dataset. The algorithm makes use of well known methods such as superpixels and support vector machines to solve the problem. In the end, it has its limitations and there is definitely room for improvement, but the implementation works fine for non-complex cases and the running time performance is not bad.

References

[1] J. Luiten, P. Voigtlaender, and B. Leibe, “PReMVOS: Proposal-generation, Refinement and Merging for the DAVIS Challenge on Video Object Segmentation 2018,” p. 4.

[2] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[3] P. Neubert and P. Protzel, “Superpixel benchmark and comparison,” in *Proc. Forum Bildverarbeitung*, 2012, vol. 6.

[4] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, “Slic superpixels,” 2010.

[5] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, “SLIC superpixels compared to state-of-the-art superpixel methods,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, 2012.

[6] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[7] “DAVIS: Densely Annotated VIdeo Segmentation.” [Online]. Available: https://davischallenge.org/index.html. [Accessed: 12-Jan-2019].

[8] “DAVIS 2016 Benchmark” [Online]. Available: https://davischallenge.org/davis2016/soa_compare.html. [Accessed: 12-Jan-2019].