Video Object Segmentation based on Pixel-level Annotated dataset E6893 Big Data Analytics Final Project 201912-13

Hu Chong[*]
UNI: ch3467
Columbia EE
ch3467@columbia.edu

Yanlin Liu[*]
UNI: yl4238
Columbia EE
yl4238@columbia.edu

Abstract

This project tackles the task of video object segmentation, i.e., the separation of a foreground object from the background in a video, given the mask of the first frame, and builds a web application to visualize the result. We applied One-Shot Video Object Segmentation (OSVOS) to recognize and separate a foreground object based on DAVIS 2016 dataset. On the other hand, we build up a web interface for user to upload video and preview rendered video with segmented labelled object, along with additional function like data visualization of object's motion trajectory. Code is available at `https://github.com/JackSnowWolf/video-object-segmentation`

1. Introduction

The objective of Video Object Segmentation is to extract foreground objects from video clips. It has many applications such as: video editing, object tracking, video action detection, autonomous driving, etc. Object segmentation and object tracking are both fundamental research area in the computer vision area. Public benchmarks and challenges have been an important driving force in the computer vision field, with examples such as Imagenet for scene classification and object detection, PASCAL for semantic and object instance segmentation, or MS-COCO for image captioning and object instance segmentation. From the perspective of the availability of annotated data, all these initiatives were a boon for machine learning researchers, enabling the development of new algorithms that had not been possible before. Their challenge and competition side motivated us to participate and push towards the new different goals, by setting up a fair environment where test data are not publicly available.

According to above motivations, in our project, we are engaged in combining the objective of video object segmentation and the implementation of some big data techniques like data visualization and web application. That is to say, our project is composed of two parts, realizing the video object segmentation and visualizing the result in our web application. So, the first contribution is to successfully do video object segmentation based on DAVIS 2016 dataset and we will present our performance evaluation. The second contribution and innovation is to successfully design and build a web interface based on django framework and interact with back-end OSVOS model and flask api, visualizing the result and presenting it on the web. The third innovation is to do data visualization to describe the object offset

[*]equal contribution

and motion trajectory.

2. Related work

Semi-supervised Video Object Segmentation: Most of the current researches aim to segment video objects based on preliminarily provided foreground regions, and propagates them to the remaining frames. In [3], Budvytis *et al.* proposed a patch-based probabilistic graphical model, which uses a temporal tree structure to link patches in adjacent frames to exactly infer the pixel labels in video. Plenty work have been done to solve video object segmentation by using deep learning method. Märki *et al.* [8] used bilateral space to minimize the new energy on the vertices of a regularly sampled spatio-temporal bilateral grid. Perazzi *et al.* [9] also proposed a MaskTrack ConvNet and performed per-frame instance segmentation by using the detections of the previous frame, along with Optical Flow and post-processing with CRFs. Xu *et al.* [17] presented a Spatio-temporal CNN that used pretrained temporal coherence branch to capture the dynamic appearance and motion cues of video sequences to guide object segmentation and spatial segmentation branch extract semantic information from one single frame.

Different from those approaches, OSVOS is a simpler pipeline which segments each frame independently. Introducing temporal information only helps to improve the model accuracy a little but brings a lot of redundant computation to extract temporal information among continuous frames. Those methods that combine the spatial and temporal together can't be applied in industry directly. To learn temporal information among continuous frames also means the computation of few images should be processed separately and therefore occupy much larger GPU resources. However, the method we used is significantly faster and can be easily industrialized.

3. Data

DAVIS-2016[10]: The dataset, named DAVIS (Densely Annotated VIdeo Segmentation), consists of fifty high quality, Full HD video sequences. The dataset was collected by Perazzi *et al.* [10] deliberately. It contains multiple occurrences of common video object segmentation challenges such as occlusions, motion-blur and appearance changes. Each video is accompanied by densely annotated, pixel-accurate and per-frame ground truth segmentation. In addition, they also provide a comprehensive analysis of several state-of-the-art segmentation approaches using three complementary metrics that measure the spatial extent of the segmentation, the accuracy of the silhouette contours and the temporal coherence. In our project, we will also use those three metrics to evaluate our performance.

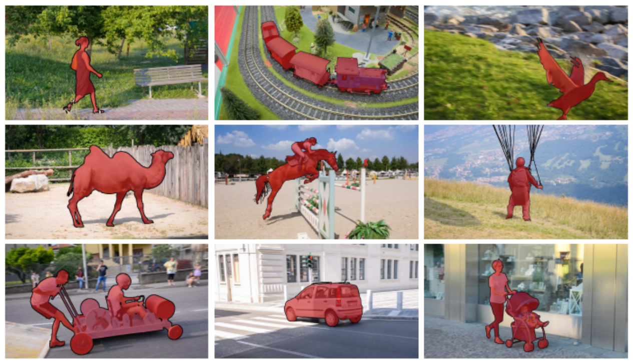

Figure 1. **DAVIS-2016**: Sample sequences with ground truth segmentation masks overlayed.

Complementary Dataset: Except DAVIS-2016, we also can also use other datasets, such as DAVIS-2017 [12], Youtube dataset [1]. Although they are not fully marked with every frames, they are designed for video object segmentation and we can use them as complementary dataset.

4. Method

4.1. One-Shot Deep Learning

The basic idea of One-Shot deep learning is inspired by human behaviors. Let assume that one

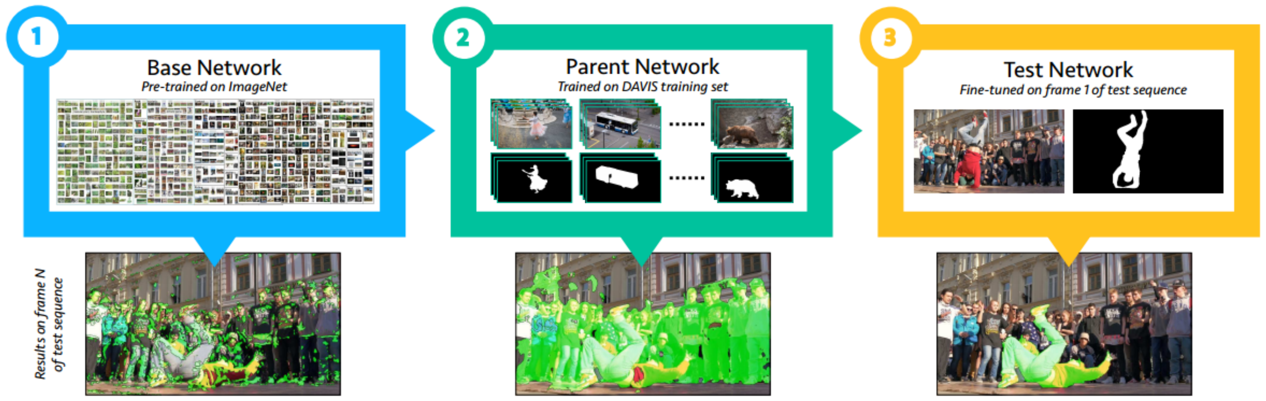
1
Base Network
Pre-trained on ImageNet
2
Parent Network
Trained on DAVIS training set
3
Test Network
Fine-tuned on frame 1 of test sequence
Results on frame N of test sequence

Figure 2. **Overview of OSVOS**: (1) The base network uses a pre-train base CNN for image labeling on ImageNet; the pre-trained weight here is only used for transfer learning. (2) Train on the DAVIS training dataset to segment common general objects . (3) Adjust for certain appointed foreground object to keep track the segmentation of that object.

human wants to segment one object from a sequence of continuous or a short video. Intuitively, this process is related to two kinds of behavior: human need to recognize a object from one frame, analyze the entity and memorize some features of that object, in order words, one need to generate a *model* to recognize a foreground object from background object; then, one can search for that object in the subsequent image sequence and track the object in the video. For human, we only need very limited amount of information to recognize the object from the first frame and locate the object in new frames even with changes in appearance, shape, occlusions, etc. One may easily find the object in the subsequent image by his previous memory about that object. This procedure is related to strong priors: one need to first realize that ”it is an object” and then ”it is the object that I want to focus”. The one-shot method is based on this logic.

Let’s first talk about the training steps of One-Shot deep learning. A Fully Convolutional Neural Network (FCN) is the backbone of the whole structure. It is used for binary classification to separate the foreground object from the background. According to the method provided by the Caelles *et al.* [4], the following training steps are: First, by using common foreground objects dataset for segmentation, construct a model that is able to discriminate the general notion of the foreground object, i.e., ”it is an object”; then, fine-tune the network for a small certain number of iteration so that the model can recognize the object in the subsequent image sequence, i.e., ”it is the object that I want to focus”. The overview of the whole training procedure is shown in Fig. 2

4.2. End-to-end Trainable Foreground U-Net

The network structure is based on the CNN architecture provided by Maninis *et al.* [7]. The original network structure is used for biomedical image segmentation and also has been proved that it is also valid in common images. It also provides some good features: 1. accurately localized segmentation output; 2. relatively small of parameters to train from a limited amount of annotated data; 3. Relatively faster inference time. The back bone of this CNN architecture is VGG [15], a very traditional structure used to extract information from image efficiently. Inspired by [13], a simple revised version is applied in our project. The network architecture extracts high dimensional information through down sampling through convoluational layer and pooling layer; and then, it uses up-sampling to restore the label of each pixel. The traditional network VGG has groups of convolutional plus Rectified Linear Units (ReLU) layers grouped into 5 stages. Our structure connects convolutional layers to form separate skip paths from the last layer of each stage (before pooling). In the up-sampling procedure, feature maps from the separate paths are concatenated to construct a volume with information from different levels of detail. We linearly fuse the feature maps to a single output which has the same dimensions as the image, and we assign a loss function to it. The proposed architecture is shown in Fig. 3 (1), foreground branch.

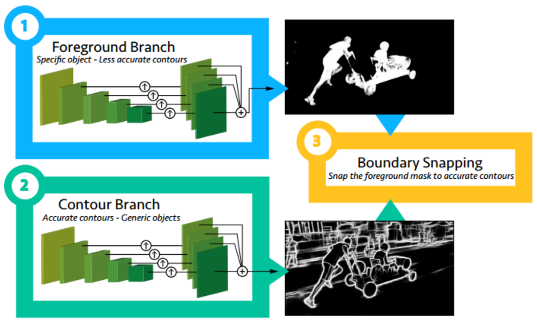
1
Foreground Branch
Specific object - Less accurate contours
2
Contour Branch
Accurate contours - Generic objects
3
Boundary Snapping
Snap the foreground mask to accurate contours

Figure 3. **Two-branch architecture**: The main foreground branch (1) is used to detect the rough location of the foreground object; Contour branch complements the main foreground branch and provides a more accurate boundaries (3).

4.3. Loss Function

Since we only consider to segment foreground object from background object, the pixel-wise cross-entropy loss for binary classification is used as optimizing target in this case. The original form is definied as:

$$\begin{aligned}\mathcal{L}(\mathbf{W}) = &- \sum_j \Big(y_j \log \mathbf{P}(y_j = 1|X;\mathbf{W}) \\ &+ (1-y_j)\log(1-\mathbf{P}(y_j=1|X;\mathbf{W}))\Big) \\ = &- \sum_{j\in Y_+} \log \mathbf{P}(y_j = 1|X;\mathbf{W}) \\ &- \sum_{j\in Y_-} \log(1-\mathbf{P}(y_j=1|X;\mathbf{W}))\end{aligned} \tag{1}$$

where $\mathbf{W}$ are the standard trainable parameters of a CNN, X is the input image, $y_j \in \{0,1\}, j = 1,\ldots,||X||$ is the pixel-wise binary label of X, and Y+ and Y- are the positive and negative labeled pixels. $\mathbf{P}(\cdot)$ is obtained by applying a sigmoid to the activation of the final layer.

However, in order to handle the imbalanced case between the two binary classes, we used balanced pixel-wise cross-entropy loss proposed by Xie *et al.* [16]. The modified version of the cost function, originally used for contour detection is:

$$\begin{aligned}\mathcal{L}_{mod} = &- \beta \sum_{j\in Y_+} \log \mathbf{P}(y_j = 1|X;\mathbf{W}) \\ &- \beta \sum_{j\in Y_-} \log(1-\mathbf{P}(y_j=1|X;\mathbf{W}))\end{aligned} \tag{2}$$

4.4. Contour Snapping

By only using the foreground branch, it is not enough to locate the foreground object accurately. After add skip connections to minimize the loss of spatial accuracy, the model performance is not satisfying. However, in terms of contour localization, the model performance can be still improved. There two strategies provided by original paper in this regard.

First, the Fast Bilateral Solver (FBS) [2] can be used to snap the background prediction to the image edges. It performs a Gaussian smoothing in the five-dimensional $colorlocation$ space, which results in a smoothing of the input signal (foreground segmentation) that preserves the edges of the image. It has two advantages: it is fast, which means it can be applied in practice; it is differentiable so it can be included in an end-to-end trainable deep learning. However, this method is not enough since it might preserve some naive image gradients.

To solve this question and improve more result, the second strategy is introduced. The basic idea is to use another branch, contour branch, as shown in Fig. 3 (2), to predict accurate boundary of a object. It works as a complementary CNN and it is trained to detect object contours. Those two branches have exact same structure, but trained for different losses. Finally, in the boundary snapping step Fig. 3 (3), the model computes superpixels that align to the computed contours (2) by means of an Ultrametric Contour Map (UCM) [11]. Then, then model can take a foreground mask (1) and we select superpixels via majority voting (those that overlap with the foreground mask over 50%) to form the final foreground segmentation result.

5. Experiments

5.1. Implementation Details

Data The experiments will basically conduct on DAVIS-2016 dataset. Inside the dataset, we use 30 video sequence as training dataset and 20 video sequences as test dataset. The image frame during traing is 480p (480×854) Evaluation will be executed on the 20 video sequence test dataset. During the training procedure, we also applied data augmentation, such as flip and crop, to increase the robustness of our model. We use threes metric to evaluate the model performance: Region Similarity $\mathcal{J}$, Contour Accuracy $\mathcal{F}$ and Temporal stability $\mathcal{T}$.

Training Details Based on the overview of OSVOS in Fig. 2, the training procedure can be divided into three steps: pre-training on ImageNet; offline training on DAVIS training dataset; online training/testing for specific foreground object.

The base model weight is pre-trained on ImageNet for image labeling [15], which has been proved to be a very good initialization to other tasks [14, 6, 5, 18]. We do not train a VGG network from scratch. Instead, We use the pre-trained weight on ImageNet provided by TensorFlow.

Then, we need to train the network on the training set of DAVIS, to learn a general common features that can help to segment objects from their background, *e.g.* their usual shapes, special color, etc. We use ADAM with momentum 0.9 for 50000 iterations, which is slightly different from original paper. The initial learning rate is set to 10^{-8}. This model is called *Parent Network* as shown in Fig. 2.

After we get the Parent Network model weight, it can be applied in online training/testing. In order to segment a particular foreground object, the model further adjusts (fine-tune) for the image and the segmentation of the first frame. Then the new weight can be used to predict subsequent video frames. The fine-tuning time (once per annotated mask) and the segmentation of all frames (once per frame) affect the total inference time. And the trade-off between quality and time should be also considered.

5.2. Result

After we get the parent network weights, we fine-tine that on test video sequences and make predictions on following new image frames. Fig. 4 shows some qualitative predication results. In the figures, we can compare the ground truth with prediction mask directly. The green mask represents the ground truth; the red mask represents the error segmentation. Those examples show us the prediction can match the ground truth very well, even after several image frames. We can see there are only some parts of area near the boundary are colored with red. This could prove that the model can have good performance on object localization and boundary detection. However, for some other video sequences, with high blur, or high motion, our model is not very satisfying.

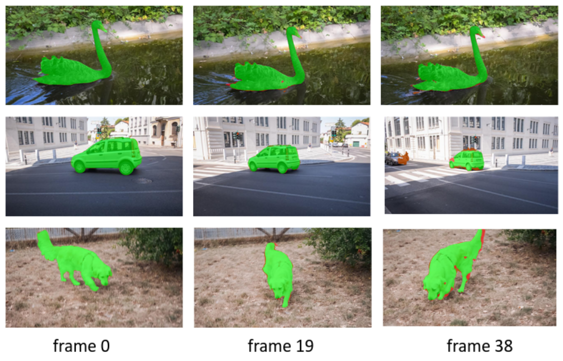

Figure 4. **Qualitative Predication Results**: Predictions on twenty test sequences; the green mask represents the ground truth; the red mask represents the error segmentation.

5.3. Evaluation

Metric	Ours	Original
Region Similarity $\mathcal{J}$	76.4%	79.8%
Contour Accuracy $\mathcal{F}$	78.2%	80.6%
Temporal stability $\mathcal{T}$	34.5	37.6

Table 1. Comparison with original paper on twenty test sequences.

For predictions on the test dataset, we also run the three metrics we mentioned before to evaluate model performance. Fig. 5 shows that Region Similarity $\mathcal{J}$, Contour Accuracy $\mathcal{F}$ and Temporal stability $\mathcal{T}$ per video sequences. From the plot, we can clearly see that the overall performance on test dataset is relatively good. However, we can also see that on some video sequences, such as Bmx-Trees (which is riding a bicycle across trees with high speed), Dance-Twirl (which is a person dancing in front of a bunch of people, and the dancing person is foreground), the model doesn't perform very good. That might because foreground object itself is not easily to separated from background objects, or the speed is too fast. Table. 1 shows a comparison result with the original paper. As we can our performance is still a little bit lower than original method. We believe that difference is not large and can be reduced by further adjusting hyper-parameters. Overall, our method to segment foreground method is valid.

6. System

6.1. Overview

Basically, our system aims to support previewing uploaded video with segmented foreground object, and expected outcome is to separate foreground objects with larger than 75% overlapping with ground-truth on average, and to provide API for video website and simple web front-end for demo. As is shown in Fig. 6, this simple overview diagram of system illustrates how system works and how front-end and back-end interacts. We will elaborate on the structure of both front-end and back-end separately, along with their interaction methods in details.

Environment	TensorFlow 1.14, CUDA 9.2, CUDNN 7.2.1, OpenCV 3.4.3, FFmpeg
Back-end	Flask
Front-end	django, video.js/D3.js

Table 2. Environment and Framework.

6.2. Front-end

We provide this web application to visualize the result. Using django to build a web interface. User can upload the original video on the web interface. After the interaction between django and flask api, finally, user can watch the rendered video on web interface. So, front-end structure is based on Django frameworks, and the interface is based on local host. So we start with 'python manage.py runserver', then we can open with localhost/homepage to view our website.

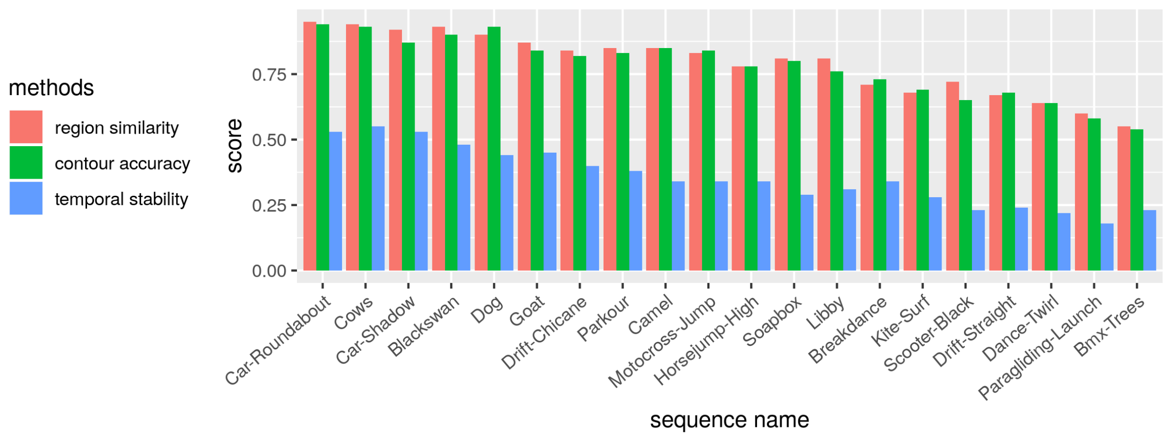

Figure 5. **DAVIS Validation**: Evaluate model performance on twenty test sequences using Region Similarity $\mathcal{J}$, Contour Accuracy $\mathcal{F}$ and Temporal stability $\mathcal{T}$.

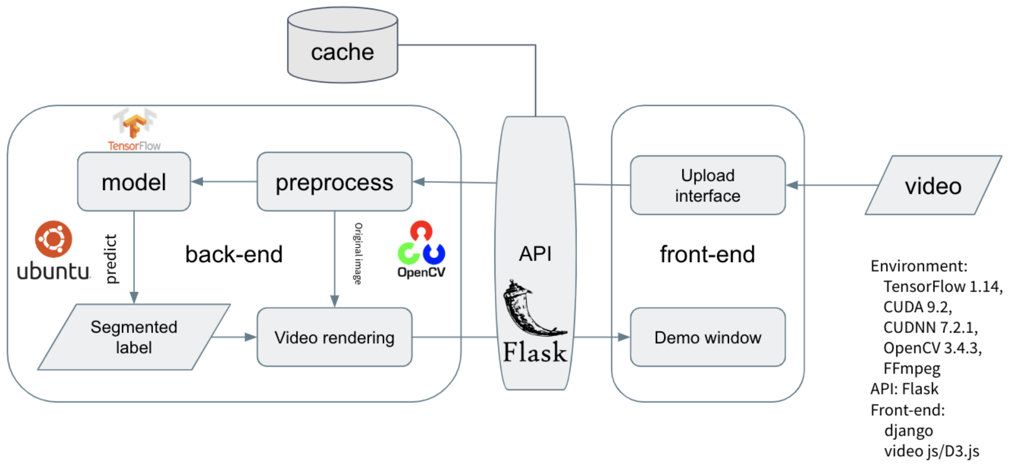

Figure 6. **Overview diagram of system**: (1) User can upload a video along with its name and first mask of this video on website upload interface. (2) Once video is uploaded, Django would send the video to Flask API. (3) Back-end uses model to predict segmented label and render the uploaded video. (4) API send back the rendered video to front-end so that user can preview the rendered video on the interface.

The user interface is actually a website that includes five parts in the navigation bar: homepage, video upload, video preview, data visualization and video gallery. Therefore, there are actually five html templates in our front-end structure. on homepage, we present some basic introductions of our project. on video upload page, user can upload a video file with a name and a first-mask picture file. On video preview page, once a video is uploaded by a user, Django would send requests.post to flask api. Api will get the uploaded video and return the rendered video. So, after user press the upload button on video upload page, our web will directly redirect to the video preview page so that user can watch the video and get the information about this rendered video at the video page once the rendered video is ready. For a small video file, for example, a video is less than 15MB as we used in demo, user can watch rendered video at once. On visualization page, we present a 3D scatter plot to describe the overall offset of the position coordinates of the object we are tracking, corresponding to the motion trajectory of the object in the video. X-axis and y-axis represent the horizontal and vertical coordinates of the object. Z-axis represents time, in unit of each frame. On gallery page, there is a gallery presenting all the cached rendered videos in our project.

6.3. Back-end

Back-end system is composed of the flask api and the model we discussed in method. Once the video is uploaded, django would send request post to the url that our flask api provides. Flask api then gets the uploaded video, implements data preprocess, uses osvos model to predict the segmentation label in the video(track object), render the video and return it back to django. As for the prediction part, what the model does is to predict the segmentation label of this video so that we can know whether each pixel belongs to the object we want to separate. So basically, all these back-end procedures are all done sequentially inside one program, which includes getting uploaded file, hitting video in cache if this video is cached before or using osvos model to predict and render if not cached, and returning the rendered video back to django.

6.4. Data Visualization

In data visualization part, our intention is to present some data behind the video object segmentation and visualize it. Once we predict the segmentation label of uploaded video, we can know whether each pixel is the object we want to separate. After that, we can obtain the position information (x, y) of the target object in each frame of the rendered video, including the midpoint position, upper left corner position, upper right corner position, lower left corner position, and lower right corner position of the object's rectangular frame. The data we get is a 80x6 txt data, the columns describe the x horizontal and vertical coordinates of midpoint, left-upper corner point and right-lower corner point of the object. So the time series is from 1 to 80, and we do data augmentation to get another two points, left-lower corner point and right upper corner point. Then we convert txt file into csv file using pandas in order to further use the data more conveniently. We use javascript especially d3.js to visualization the object position data. We implement a 3D scatter plot to describe the overall offset of the position coordinates of the object we are tracking, corresponding to the motion trajectory of the object in the video. X-axis and y-axis represent the horizontal and vertical coordinates of the object. Z-axis represents time, in unit of each frame. All the visualization is available on the web interface.

For example, we uploaded a video named goat.mp4 which is a video recording a goat. Once this video is rendered and goat is separated from the background, we can get this goat's coordinates position data of each frame. Then user can see the object offset 3D scatter plot on website visualization page. In Fig. 7, it is the object offset 3D scatter plot of goat video. Z-axis describes time in unit of each frame, x-axis and y-axis describes the horizontal and vertical coordinates of the goat in the video. Clearly, we can see in this video the goat motion trajectory is relatively unstable. Therefore, to some extent, it accurately describes the object offset and motion trajectory in this video.

7. Conclusion

In our project, we basically re-implement One-Shot Video Object Segmentation (OSVOS) to recognize and separate a foreground object based on DAVIS 2016 dataset. We also use visiualization and three metric, Region Similarity $\mathcal{J}$, Contour Accuracy $\mathcal{F}$ and Temporal stability $\mathcal{T}$, to evaluate the model performance. We proved that this way is valid to separate a foreground object from the background in a video, given the mask of the first frame. Beyond that, we also build up a web application to provide previewing rendered video with

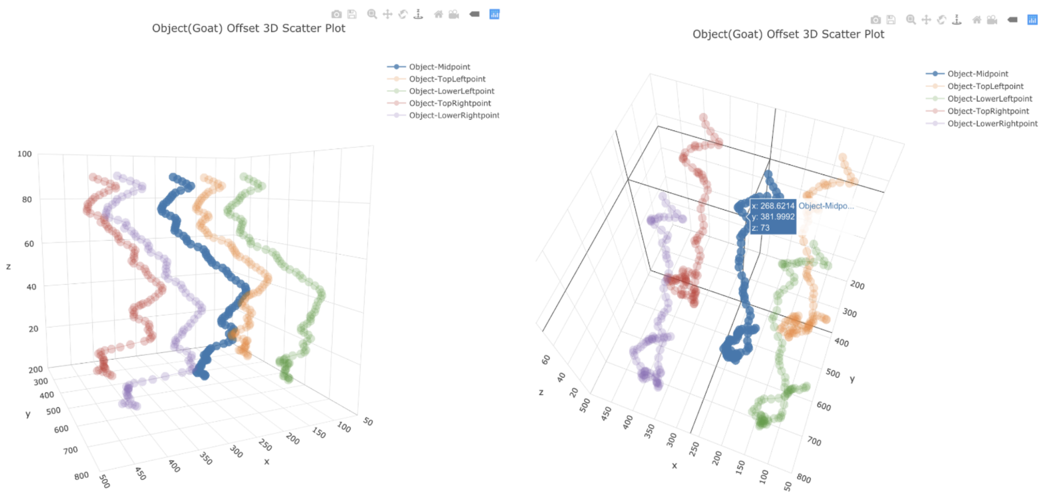

Figure 7. Object (goat) offset 3D scatter plot.

segmented labelled object, along with additional function like data visualization of object's motion trajectory. User could see how foreground moves, how to track the object and other visualizations.

8. Future work

Due to time limitation, we still have a lot of things to do. As for the method we applied to solve video object segmentation, we still could try to use other model structures, alternative loss functions or refer other innovations from other paper. Then we can improve the model accuracy or speed up inference time. As for the web application, we can optimize the logic and provide better experience for users. Also, we can provide other fancy functions, such as stabilizing or video editing. Beyond that, we can further consider how to use the foreground objects in the future.

Individual Contribution

Chong Hu and Yanlin Liu have equal contributions.

- Chong Hu: Re-implemented OSVOS, run evaluations, applied model to provide API; PPt slides and report.

- Yanli Liu: Re-implemented OSVOS, designed UI and deployed front-end, connected back-end API; PPt slides, report and video.

Acknowledgement

First, we would like to thank Professor Ching-Yung Lin, and all TAs, including Hung-Yi Ou Yang, Yunan Lu, Juncai Liu, Tingyu Li, Ashna Aggarwal and so on. Without you, we can never finish our final project from scratch. Thank you for the instructions and help of a whole semester!

Second, we would like to thank all the authors of the original paper and all the contributors of the datasets. They have done a great work before us.

References

[1] S. Abu-El-Haija, N. Kothari, J. Lee, A. P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. In *arXiv:1609.08675*, 2016.

[2] J. T. Barron and B. Poole. The fast bilateral solver. In *ECCV*, 2015.

[3] I. Budvytis, V. Badrinarayanan, and R. Cipolla. Semi-supervised video segmentation using tree structured graphical models. In *CVPR*, 2011.

[4] S. Caelles, K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool. One-shot video object segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[5] B. Hariharan, P. A. Arbeláez, R. B. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, pages 447–456. IEEE Computer Society, 2015.

[6] I. Kokkinos. Pushing the boundaries of boundary detection using deep learning. In *ICLR 2016*, 2015.

[7] K.-K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. V. Gool. Deep retinal image understanding. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2016.

[8] N. Märki, F. Perazzi, O. Wang, and A. Sorkine-Hornung. Bilateral space video segmentation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 743–751, June 2016.

[9] F. Perazzi, A. Khoreva, R. Benenson, B. Schiele, and A.Sorkine-Hornung. Learning video object segmentation from static images. In *Computer Vision and Pattern Recognition*, 2017.

[10] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] J. Pont-Tuset, P. Arbelaez, J. T.Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping for image segmentation and object proposal generation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(1):128–140, Jan. 2017.

[12] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung, and L. Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017.

[13] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[14] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, Apr. 2017.

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[16] S. Xie and Z. Tu. Holistically-nested edge detection. *Int. J. Comput. Vision*, 125(1-3):3–18, Dec. 2017.

[17] K. Xu, L. Wen, G. Li, L. Bo, and Q. Huang. Spatiotemporal cnn for video object segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[18] J. Yang, B. Price, S. Cohen, H. Lee, and M. Yang. Object contour detection with a fully convolutional encoder-decoder network. In *Proceedings - 29th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 193–202, United States, 12 2016. IEEE Computer Society.