Sequence-to-Sequence Video Object Segmentation Using Video Representations

Kartik Ramesh Jain
University of Central Florida
Orlando, Florida
kjain@knights.ucf.edu (4810577)

Abstract

This project was undertaken primarily to re-implement "YouTube-VOS: Sequence-to-Sequence Video Object Segmentation [9] as part of an assignment for the course CAP 6412: Advanced Computer Vision. As shown later on, this has been successfully accomplished. Incremental improvement is also made to original architecture by adding two networks in the architecture that should, in principle, capture better video representations. Experiments are performed on the YouTube-VOS dataset [10]. The reported results include both qualitative visualization and quantitative metrics.

1. Introduction

The paper "YouTube-VOS: Sequence-to-Sequence Video Object Segmentation [9] (referenced as "base paper" from here on) learns video object segmentation using a sequence-to-sequence learning approach. The task of object segmentation in a video is as follows: We are given a video, and the exact location (segmentation mask – pixel level details) of an object from the first video frame and we're tasked with the problem of tracking that object throughout the video. The YouTube-VOS dataset [10] is the largest Video Object Segmentation dataset so far and is suitable for this task. This becomes a challenging task particularly because of YouTube-VOS dataset where there is a lot of variation in the dataset and neither the object nor the camera is static.

The base paper used a sequence-to-sequence learning approach in that it treats the videos and their corresponding segmentation videos (ground truth) as a sequence of frames. These frames are then processed by a recurrent network. The video frames are encoded first into a three dimensional tensor and processed by an LSTM network designed for images – ConvLSTM [5]. The hidden states are updated at each time-step and, at each time-step, they are decoded to a segmentation mask.

We will get into the implementation details later but one thing to be noted here is the base paper's architecture/model sees only one frame at a time and hence lacks video-level understanding. My architecture (referenced as "updated architecture" from here on) instead uses both frame and video-level representations to generate segmentation masks at each time-step.

2. Related work

Most of this section follows work up until the base paper: Talking about the dataset first,

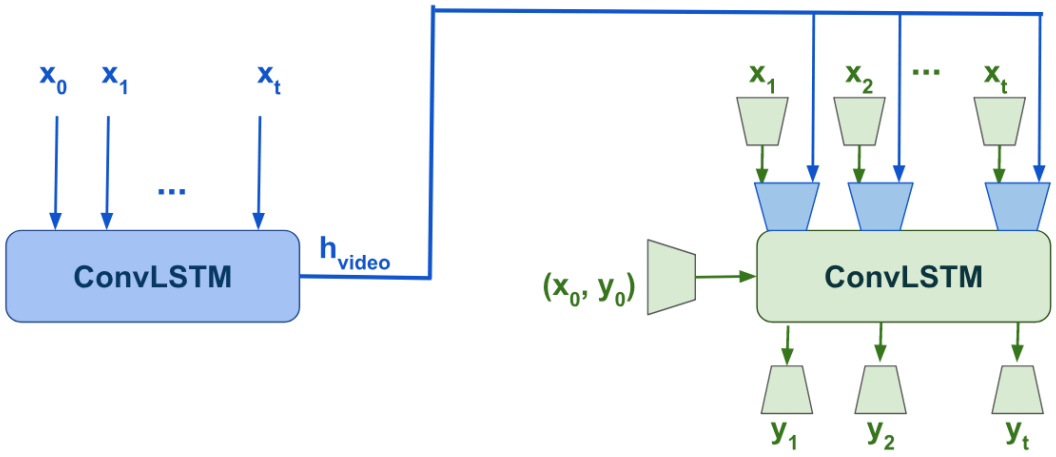

Figure 1. Proposed approach: Uses a ConvLSTM for capturing video representation (h_{video}) and passing them with frame-level encodings through a convolutional block to a second ConvLSTM. The parts in green indicate the base paper's setup while the parts in blue indicate components that we added.

before YouTube-VOS, almost all the video object segmentation datasets were considerably small in size with the best one being DAVIS [3]. The 2017 version of DAVIS has about 90 videos with multiple objects per video. This is considerably small than YouTube-VOS which has more than 3000 videos.

Pertaining to the methods used so far, many approaches use hand-crafted features. Recent deep learning appraoches to this challenge include model build on image segmentation networks (e.g. :- [1]) further using Online Learning to improve their performance. Another approach [2] uses spatio-temporal consistency to propagate objects over time. Others used Optical-Flow based techniques. Sequential learning has not been explored in this domain so far. Our base paper takes inspiration from that and hence employs a recurrent network to learn Video Object Segmentation (VOS).

3. Method

In this section, we will discuss the base paper's model architecture. Since there is no code accompanying with this paper, we will try to replicate their original implementation by following the text in their paper as closely as possible. We chose not to include model summaries in this report to keep the document's length within given page limit. The summaries can be easily viewed by running individual network's files in the `networks` directory from the accompanying code.

3.1. Pre-processing

Since the videos are stored as frames in distinct directories at their native resolution (mainly 720×1280 with some exceptions), we need to resize them to a smaller dimensions for memory considerations. The paper went with 256×448 and so did we. In the same spirit of memory considerations, we train the network using only 5 frames per video similar to what the authors did. Finally, we obtain a $5 \times 3 \times 256 \times 448$ video. We normalize this with ImageNet `mean` and `std` since (as we'll see in what follows) we are using encoding networks pre-trained on ImageNet.

For the segmentation video $5 \times 1 \times 256 \times$

448, we do not normalize with ImageNet since it makes little sense to do so. Also, based on how the model is designed, even if we have multiple objects in a video, only one object can be tracked at a time in a single forward pass. So, for a video with N objects, we have to sample it N times which different object each time. The dataloader was particularly difficult to write since each instance had different pixel values which we had to rescale to 1.0 and later map back to it's original value.

3.2. Initializer

The initializer was designed to take the first video frame concatenated with the first segmentation mask and produced initial cell state and hidden state for the Convolutional LSTM since we want the sequential processing to be conditioned on where the object was in the initial frame. For this network, we convert the four channel input to a three channel output as we empirically found it was the best way to work a pre-trained VGG-16 model [6] (since VGG16 is what the base paper used).

Following the convolutional layers of VGG-16, we add 1×1 convolutions to bring output precisely to $512 \times 8 \times 14$ dimension. We perform these convolutions independently to obtain both c_0 and h_0 since these serve as initializers for ConvLSTM.

3.3. Encoder

The encoder simply takes one RGB images and uses pre-trained convolutional layers of VGG-16. A further 1×1 convolutional layer is similar to initializer's setup which outputs $\tilde{x}$ of size $512 \times 8 \times 14$. This is used as input to the Convolutional LSTM. The same network is used at each time-step to produce $\tilde{x}$ for that time-step.

3.4. Convolutional LSTM

Instead of using a repository for ConvLSTM, we implemented it ourselves referring to the original paper's [5] equations for LSTM gating mechanisms. This part of the network is responsible for sequential updates of it's hidden states (representations) that allows the model to learns to track an object over future time-steps.

Particularly, in our implementations, we kept the padding to `SAME` meaning that the working dimensions of the entire ConvLSTM is $512 \times 8 \times 14$. At each time-step, the updated hidden state (h_t) is used by decoder.

3.5. Decoder

The base paper was ambiguous for this network in that the exact upsampling procedure was not clear. Empirically, we found out that using transposed-convolutions produced poorer segmentation masks compared to the traditional convolution and interpolation-based upsampling (scale factor = 2). In our case, we have used five such blocks with convolutions of kernel size 5 as mentioned in the paper with the network ending in a sigmoid to produce a $1 \times 256 \times 448$ grayscale image.

3.6. Training

The loss used was Binary-Cross-Entropy (BCELoss) since we ideally want the segmentation mask to be a binary image i.e., either a pixel belongs to the instance/object or it does not. This loss is computed for all the time-steps (at 5 FPS where we have ground truth available). Adam optimizer was used and while the base paper says they used learning rate of `1e-5` we found out that the training convergence was too slow so we carried further experiments on a learning rate of `1e-4`.

3.7. Post-processing

Since the obtained masks are output of a sigmoid function, after the model converges, the values tend to be closer to 1.0 or 0.0 but are not exactly those values. Therefore, we have to threshold them. We went with the theoretically justified

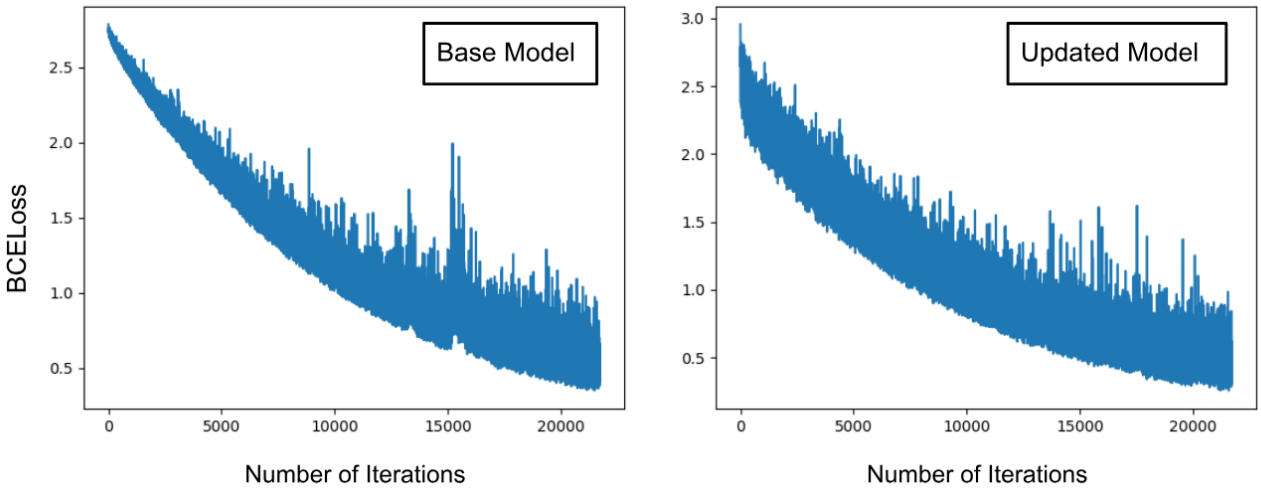

Figure 2. Binary Cross Entropy loss vs. Number of Iterations for base model (left) and updated model (right).

0.5 even though other threshold may hurt or boost our performance. In addition, since the leaderboard submission asks the resolution of masks to be 720×1280, we use Bi-Linear interpolation and remove the anti-aliasing artifacts so that the final image contains exactly $N + 1$ unique pixel values for N objects plus one background (0.0).

3.8. Proposed Approach: Video Representation

In the base paper's sequence-to-sequence model, each time-step operation is performed independently with the only relation between them being the h and c tensors. Nowhere in the entire architecture is the entire video representation being captured. This video-level representation may be useful to capture the motion of the object/instance accurately and could be useful for generating segmentation mask at each time-step.

Our motivation to use another ConvLSTM solely on the video-frames was so that the model can capture video representations and use those for frame-level segmentation. As seen from figure 1, our updated model combines the video representation (h_{video}) with the encoded frame at each time-step so that the second ConvLSTM has access to frame-level as well as video-level information at each time-step so that it can help generate better segmentation masks.

4. Experiments

A series of experiments were performed on the YouTube-VOS dataset with both the base model and the updated model which will be discussed in this section.

5. Quantitative Evaluation

From figure 2, we see that as the training progresses, both of the models converge. Although our updated model's loss value is consistently lower than the base model's loss after a few thousand iterations, we do not make any concrete claim that our model is better than the base model. Regardless, in our interpretation we believe that the learning capability of our model is certainly higher than that of the base model.

Since we're dealing with segmentation masks, a quite popular and appropriate metric for this task is Jaccard Similarity Index (J) which calculates the Intersection Over Union of predicted mask and ground truth. Another metric named $F-$Value is used to measure contour similarity via calculation of precision and recall. Since, as

Metric	Base Model	Updated Model
Overall	0.0299089895762	0.0446368230621
J_{seen}	0.00494019075989	0.0105933667722
J_{unseen}	0.00301982829088	0.0371834866469
F_{seen}	0.0704420090413	0.0648245982349
F_{unseen}	0.0412339302126	0.0659458405945

Table 1. Obtained J and F scores for both seen and unseen categories in validation set.

part of this assignment, we were required to submit the validation results to the online leaderboard for this competition hosted on CodaLab [8], we do that as well. Obtained scores for both the models can be seen in table 1.

5.1. Qualitative Evaluation

The first frame in the segmentation masks is the ground truth for all the figures belonging to this section. In figures 3 and 4, we visualize a video selected randomly from the training and validation set respectively. A sample for tracking multiple objects from the validation set is shown in figure 5. We see how both models perform on the VOS task for these samples.

In figure 4, the object (a tiger) is progressively occluded by a person. We see that while the base model was unable to predict mask in the last frame (heavy occlusion), the updated model was able to segment at least the general area of the object.

In figure 4 and 5, since we do not have the ground truth for validation data, we show only the predictions. In figure 4, while the base model considered the hand of the person as part of the object (bird) the updated model couldn't segment the height of the bird accurately.

The two different gray levels in figure 5 indicate two different objects where, more or less, both of the models had a difficult time segmenting the small object properly. In these samples and generally over all the samples, we see that both the models have trouble capturing the exact shape of an object.

6. Discussion

In our experiments, we observed that both the models, given enough training (trained for more than 80 epochs), can segment objects pretty accurately on the training set but cannot localize objects very well in the validation set. We're unsure of the precise reason for this but our interpretation is that the model over-fitting on the training set.

It was expected that models would perform relatively worse on unseen categories and the base model's numbers in table 1 agree with our expectations. What's surprising though is that the J_{seen} score is lower than the J_{unseen} for our updated model. Although, since the overall score is low, making bold inferences from these quantitative metrics may not be the best idea.

Running an object detection algorithm like YOLO [4] on the location where the object is initially present and then passing only the detected region-of interest through a recurrent network can help reduce computation time and improve model performance. A more conceptually powerful improvement could adding self-attention [7] to get precise localization of objects in the video frames and help to model learn shapes better. We look forward to exploring these ideas in future work.

7. Acknowledgements

Thanks to Jyoti Kini for detailed presentations on this topic. It made our job of programming easier. Thanks to the Dr. Vyas and Prof. Shah for providing access to Newton without which running this project would have been impossible. Finally, thank you for a wonderful course, I really loved learning in CAP 6412.

Figure 3. Single training sample ($object = tiger$) with video frames (first), segmentation masks (second), base model prediction (third), and updated model prediction (fourth) in their respective rows.

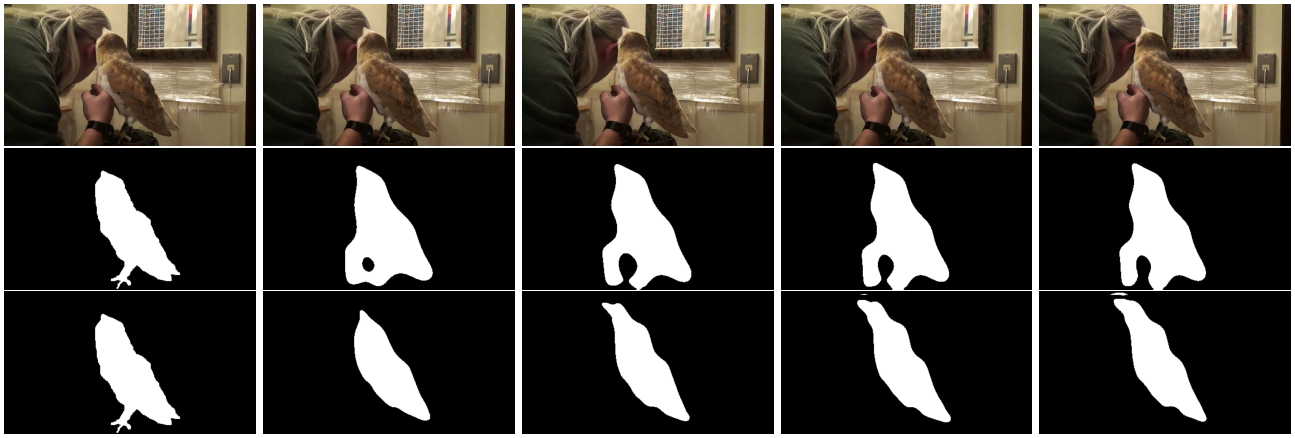

Figure 4. Single validation sample ($object = bird$) with video frames (first), base model prediction (second), and updated model prediction (third) in their respective rows.

References

[1] S. Caelles, K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. V. Gool. One-shot video object segmentation. *CoRR*, abs/1611.05198, 2016.

[2] V. Jampani, R. Gadde, and P. V. Gehler. Video propagation networks. *CoRR*, abs/1612.05478, 2016.

[3] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbelaez, A. Sorkine-Hornung, and L. V. Gool. The 2017 DAVIS challenge on video object segmentation. *CoRR*, abs/1704.00675, 2017.

[4] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[5] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[6] K. Simonyan and A. Zisserman. Very deep con-

Figure 5. Single validation sample ($object_1 = person, object_2 = person$) with video frames (first), base model prediction (second), and updated model prediction (third) in their respective rows.

volutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[8] N. Xu. https://competitions.codalab.org/competitions/19544.

[9] N. Xu, L. Yang, Y. Fan, J. Yang, D. Yue, Y. Liang, B. L. Price, S. Cohen, and T. S. Huang. Youtube-vos: Sequence-to-sequence video object segmentation. *CoRR*, abs/1809.00461, 2018.

[10] N. Xu, L. Yang, Y. Fan, D. Yue, Y. Liang, J. Yang, and T. S. Huang. Youtube-vos: A large-scale video object segmentation benchmark. *CoRR*, abs/1809.03327, 2018.