Using Temporal and Spatial Features to Track Fast Moving Objects

Midterm Project Report

Kristi Bushman, Clara Cardoso Ferreira, Avery Peiffer

April 1st, 2021

1 Approach

Our group is researching a method to recognize, track, and - if time permits - predict the trajectory of Fast Moving Objects (FMOs). FMOs are objects that move with a velocity greater than that which can be captured by the exposure rate of a camera [1]. We are using computer vision and machine learning methods to recognize spatial and temporal features in videos to follow the objects. There are several documented challenges associated with tracking FMOs. The high speed of the FMOs means their appearance is often small, blurry and inconsistent between frames, making them difficult to detect confidently [2]. As mentioned in our proposal, a volleyball at rest can appear round with colored stripes. However, when it is moving fast enough to be classified as an FMO, it can appear as a single colored streak. We propose using a convolutional recurrent network to track FMOs and then introducing an additional recurrence between frames to predict the placement of the FMO.

Our method's goal is to track FMOs by segmentation, not only through the use of visual features to ensure the same object is in the next frame, but also temporal features to identify the pattern by which the system of pixels within an object moves from frame to frame. Since FMOs vary in their visual and dynamic characteristics between frames, we expect that introducing temporal features will improve how well the model fits to the data. Furthermore, although our method is not meant to draw and predict a smooth trajectory, it is expected to determine the objects placement at a future frame. Our method is expected to computationally faster than previous methods because spatial and temporal features may override the need for extra computations. With temporal features, the network would be able to cross check predictions with previous predictions.

There are only two existing machine learning methods used for FMO detection. Both papers chose a Convolutional Neural Network (CNN) and were recently published in 2020. The first method, referred as Learning-Based Tracking, uses a segmentation method to track and estimate trajectories of FMOs. This method suffered from many false positives, which was diminished through emphasizing shape and connected component size. Sequences of detected FMOs were used to extrapolate the trajectory[2]. In contrast, our method avoids extrapolation calculations by learning expected displacements and attempts to minimize false positives by aiming to improve upon our current results. The second machine learning method used was the FMODetect method, which estimates trajectories of FMOs by first using a deblatting technique, which is simplified into matting and deblurring. Then trajectory fitting is fed into the network to predict continuous trajectories[3]. In this case, trajectory fitting refers to fitting a smooth curve to the path of the FMO after viewing the entire video. It makes the assumption that the FMO follows a straight path or a parabolic path. In contrast, our goal, a much more difficult task, is to develop a notion future location from viewing the beginning of the video, using it to predict FMO location at a later frame through a Convolutional Long Short Term Memory (ConvLSTM).

Although deblurring generally results in better fit to predicted trajectory (measured in Trajectory Intersection Over Union TIoU), it can cause the runtime to slow down from 20 fps to 0.4 fps as is shown in the results table in [3]. Our method minimizes computations, including deblurring and trajectory fitting, to achieve future position estimations. Both papers rely only on spatial features found through a UNet convolutional neural network architecture. It is expected that incorporating temporal features through an

ConvLSTM will raise accuracy and reduce computational time by avoiding deblurring, trajectory fitting and extrapolation as the previous methods; skipping these steps saves computational time needed for realtime tracking applications. Our intuition is that because there is temporal information available, the network will be able to learn a concept of trajectory (rather than fitting a curve as a post-processing step).

1.1 RVOS Overview

Our approach is based on the Recurrent Network for Multiple Object Video Object Segmentation (RVOS) proposed by Ventura et al. [4]. Unlike previous methods for object detection and tracking, the recurrent neural network (RNN) used in RVOS captures features of an object in both the spatial and temporal domains. The RVOS model predicts a match for each object instance of the image at each step of the recurrence, eliminating the need to make a forward pass for each object present in the video. The neural network has an encoder-decoder structure which allows it to make pixel-wise single frame predictions through segmentation [4]. RVOS is based on the architecture of RSIS [5], a Recurrent model for Semantic Instance Segmentation. RSIS provided the basis of predicting a mask for each instance of the image at each step of the recurrence. RVOS is robust to both one-shot segmentation, where masks are provided at the first frame of the video sequence, and zero-shot segmentation, where no such mask is provided. We use zero-shot segmentation for our task in order to be comparable to existing FMO tracking algorithms.

Figure 1 shows the encoder-decoder architecture of RVOS for a single frame. The encoder efficiently grasps features from the images to feed into the decoder as shown in blue. The decoder makes use of the ConvLSTM architecture to perform spatial and temporal recurrence as shown in green. Since Figure 1 only shows one frame, the temporal recurrence is not depicted; the masks for the previous frames are passed into the ConvLSTMs for the current frame to achieve the network's temporal recurrence.

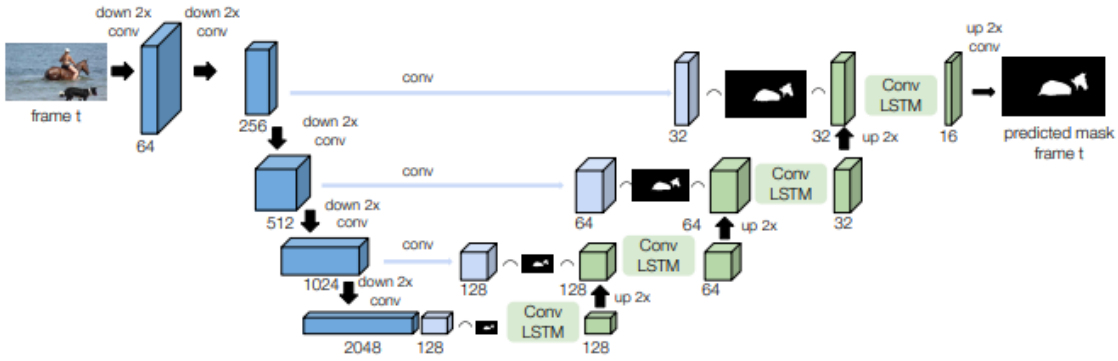

Figure 1: RVOS encoder-decoder architecture for one frame of tracking [4].

Figure 2 on the next page shows a ConvLSTM in further detail. This machine learning method computes the future state of a given cell by using the inputs and takes into account the past states of its neighbors. As seen on the image, there are three inputs to the ConvLSTM: two dimensional spatial features and time sequence. Similarly, it outputs predicted spatio-temporal states through weighed activation functions [6]. In RVOS, the ConvLSTM cells in the decoder architecture allow for both spatial and temporal recurrence, as opposed to [5], which only spatial recurrence.

1.2 Dataset Generator

There is no existing real-world FMO dataset that is large enough to support training of a deep neural network. Because of this, a synthetic dataset is necessary. For our project, we build upon a dataset generator from the FMODetect method [3]. In this section, we describe the high-level ideas of the dataset generator. We will discuss our current and planned modifications to the dataset generator in the Progress and Plans sections, respectively.

The FMO dataset generator works by generating a disc shaped object with a random color and texture as the foreground. It uses a random image from the VOT dataset as the background. The foreground is

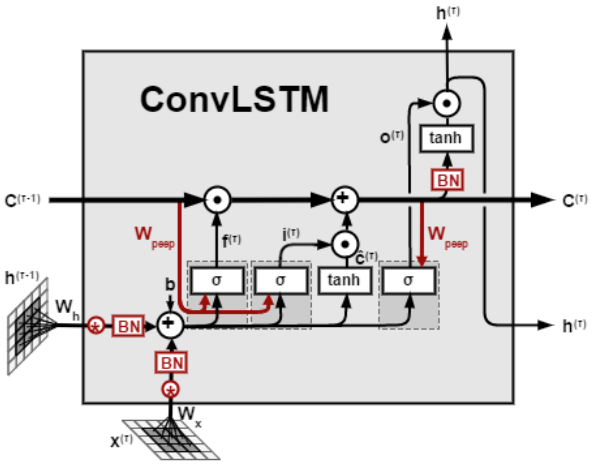

Figure 2: A single ConvLSTM cell used for spatial and temporal recurrence in RVOS [7].

superimposed on the background based on the FMO image model $I = H * F + (1 - H * M)$ as formulated in [1]. The image, I, is computed by combining motion blur with the background. Motion blur is represented with a convolution of a blur kernel H, and the undistorted disc object F. The amount that the background gets incorporated into the blur depends on an indicator function M. Different trajectory types are created, including straight lines, broken lines, and parabolas.

1.3 Novelty

As described in our project proposal, none of the existing learning-based methods for FMO detection and tracking have incorporated temporal information into the neural network. RVOS, the basis of our method, was built only for segmentation of conventional videos with relatively slow moving objects. We hope that by making sufficient changes to RVOS, we will be able to adapt it to detect FMOs.

Our method requires large amounts of data as any other machine learning method does, so we developed our own version of an FMO video dataset generator. The idea of an FMO video dataset generator is not completely novel since it has also been created in [2]. However, we do not have access to this previously developed code, as will be explained in the Progress section. We implemented an FMO video dataset generator ourselves by modifying a previously developed FMO image (not video) dataset generator [3]. We can now control the variety within the videos, how long the videos are, and how many videos are available for training, testing, and validation phases. This method is further discussed in the Progress and Plans sections. Our specific logic of creating new frames with moving FMOs is original since we plan to specify the motion of the objects to mimic realistic motions and backgrounds.

Our plans for predicting the future relative spatial location of FMOs at specified timestamps based upon its trajectory is also novel. This task has not been explored with FMOs before, to the best of our knowledge. More information about this method discussed in the Plans section.

2 Data and Metrics

2.1 Datasets

For training, we created a synthetic dataset using a dataset generator. The generator is a modified version of the generator from the FMODetect method [3]. More details about the dataset generator is discussed in the Progress section of this report.

We will be using the FMOv2 dataset for testing [1]. This dataset consists of 19 videos of FMOs in different environments. This dataset is not large enough for training, plus we need to be able to test on the entire dataset in order for our results to be comparable with the existing methods. The FMOv2 dataset contains ground truth segmentation masks of the FMO in each video frame.

2.2 Metrics

We will be using Precision, Recall, and F1 score as the metrics for evaluating our work. In our original proposal, we mentioned using Trajectory Intersection over Union (TIoU) as our main metric of evaluation. However, after further research, we now understand that TIoU was used for a slightly different task (segmenting the trajectory - a line that traces the center of the object rather than the entire object). We will be comparing these three scores to those obtained in the existing work [1, 2, 8].

The precision, recall, and F1 score metrics are calculated from the respective totals of true positives, false positives, and false negatives. Staying consistent with the existing work, a true positive is defined as having an $IoU > 0.5$. If there is an FMO in the ground truth, but the model predicts no FMOs, then it is a false negative. If there is no FMO in the ground truth, but the model predicts an FMO, then it is a false positive. If there is an FMO in the ground truth, but the model predicts an FMO in an incorrect location (i.e. the $IoU < 0.5$), then it is both a false positive and false negative.

3 Progress

3.1 Environment setup

Our first step was setting up the computing environment for training and testing our models. We wanted to make sure that we could get the unmodified RVOS code working using the dataset originally used in the RVOS paper before we made any modifications or swapped out the existing dataset for the FMOv2 dataset. We had originally planned on using Pitt's CRC instead of Google Colab since the RVOS codebase consists of regular Python files, and not Jupyter notebooks. However, we later realized that we can execute Bash commands in Colab in order to run Python scripts that are not in the Jupyter format. After this realization, we decided to try both environments.

After a few days of trial and error, we were able to get the RVOS code running on both the CRC and Colab. We decided that we will stick with Colab for most of experimentation in our project. The CRC is not ideal for quick experimentation/feedback when editing code, since we must submit jobs and wait in the queue before our code will be executed. Colab is much more intuitive in this sense. However, we still plan to use the CRC at the end of our project to calculate the inference time of our method (seeing if it can process in real time). We will use the CRC for this calculation instead of Colab since performance metrics may be affected by the virtual environment/hardware in Colab.

3.2 Acquiring a dataset

We originally wanted to use the FMOv2 dataset described in [1]. However, we concluded that this dataset is not large enough for training a neural network. Thus a dataset generator is necessary to create our own data source for training. Our plan was to use an FMO dataset generator previously developed in [2]. This generator creates synthetic videos by using a static background and adding foreground objects that simulate the trajectory, motion, and blur of FMOs. We contacted the creators, Dr. Zita and Dr. Sroubek about their generator code and were initially told that they would share it with us. However, despite follow up emails we were not able to obtain the code. At this point, we are no longer considering their video dataset generator to be a viable option; as a result, we developed our own video dataset generator.

To create our video dataset generator, we expanded upon an FMO image dataset generator that was developed for the FMODetect method [3]. The code for this generator was available on GitHub [9]. However, this dataset generator does not exactly fit our needs out of the box. It produces single images instead of video sequences. In each frame, there is a different background and the FMO has a different size, color, and/or shape. Each video also has a new location and trajectory for the FMO. This was sufficient for the FMODetect method, as that method only uses visual features to detect FMOs [3]. However, since our proposed method also makes use of temporal information to detect and track FMOs, it is important that the generated dataset maintains some continuity between frames.

We decided to modify the dataset generator code in order to address this issue. The dataset generator code is complex and challenging to follow. However, we were able to modify it so that it outputs a proper sequence of frames instead of disjoint images. In our modified code, the background stays consistent in all

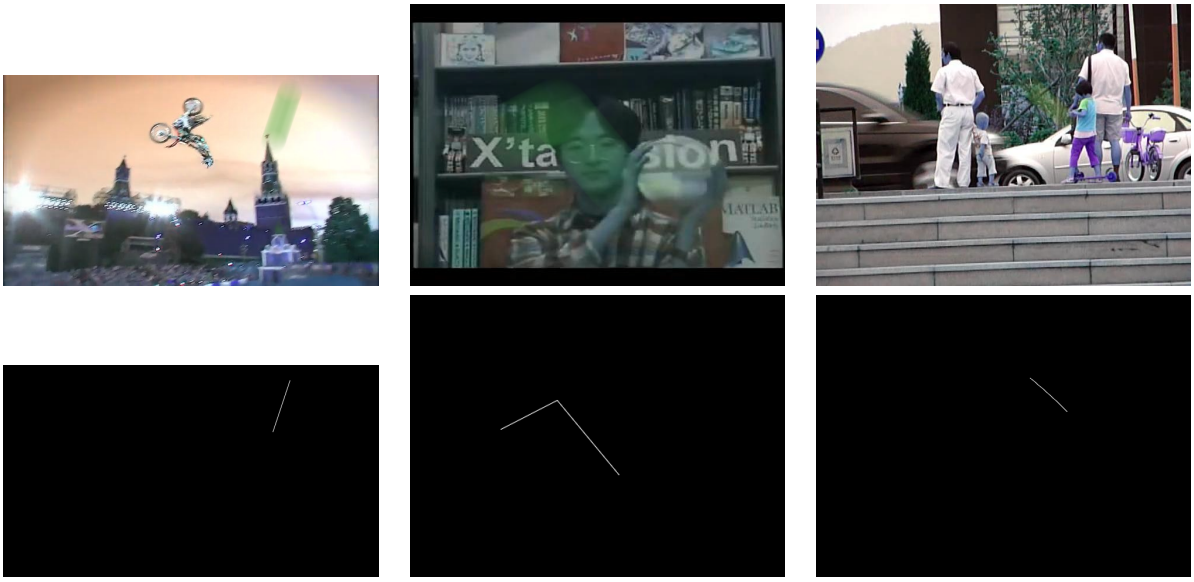

Figure 3: Output from unmodified dataset generator

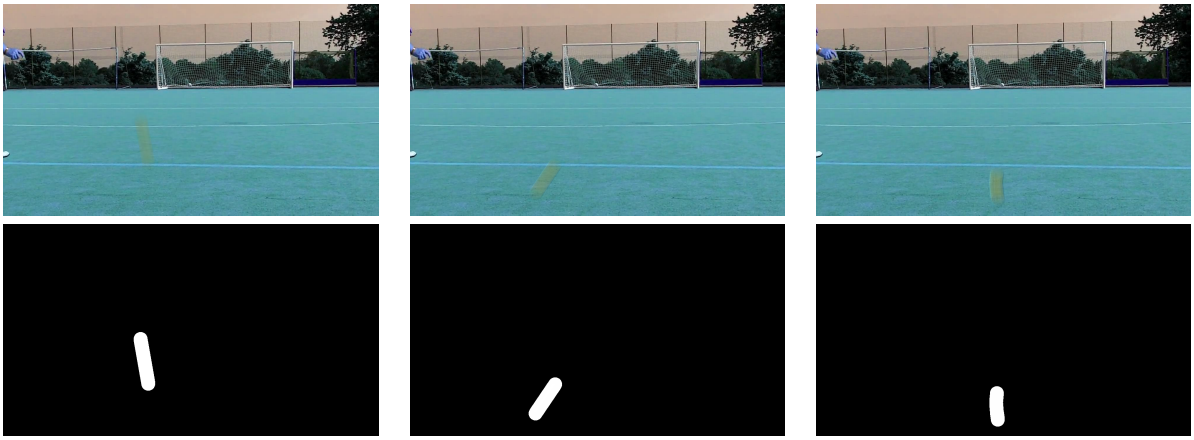

Figure 4: Output from our modified version of the dataset generator

frames of the video but changes from video to video. Additionally, the shape, size, and color of the FMO stay consistent in all frames of the video but change from video to video. Finally, we made it so that the end point of the FMO path in one frame is the start point of the FMO in the next frame. This creates some temporal information that can hopefully be used by RVOS to localize the FMO.

Another issue with the FMODetect dataset generator was that the provided segmentation masks were for trajectories (e.g. a line that follows the center of the object), instead of the full segmented object. This had to be changed to work for our task, since our ground truth segmentations are for the entire object. Figure 3 shows the output from the unmodified dataset generator from [3], while Figure 4 shows the output after our modifications to the dataset generator.

The remaining issue with modifying this dataset generator is in preserving the trajectory of the FMO from frame to frame. We have not yet figured out how to make this change in the code. It is especially difficult because in the original FMOv2 dataset, the trajectory of an FMO should remain consistent between some frames (e.g. a ball is flying through the air) but should also sometimes change abruptly (e.g. a person hits the ball or it bounces). Incorporating a good balance of consistency and sporadic changes to the FMO's trajectory will prove difficult.

3.3 Training the model

It took some time to get our data into the format the RVOS network was expecting (see challenges section). However, after fixing this, we trained RVOS on the synthetic dataset. The validation set was a subset of the synthetic dataset. Then we tested this model on the FMOv2 dataset. With the default setting of 10 epochs, we found that the weights had not yet converged. We increased the number of epochs to 30 which led to convergence. Figure 5 shows the resulting loss curves. There is a significant gap between the training and validation losses, which indicates some overfitting. The predicted segmentation masks and the metrics from our trained model are shown and discussed in the Results section.

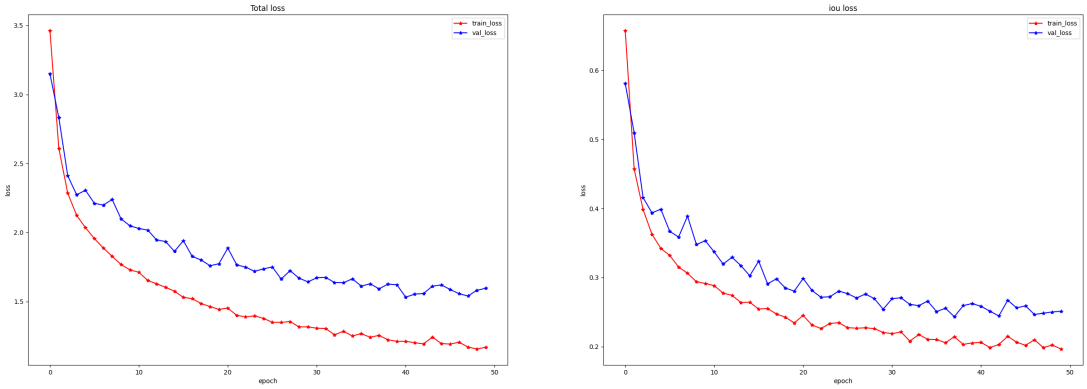

Figure 5: Loss for training and validation set. The loss is lower on the training set than the validation set, indicating overfitting.

Video	Ours			Rozumnyi [1]			Zita - small [2]			Zita - large [2]		
	Pr	Rc	F1	Pr	Rc	F1	Pr	Rc	F1	Pr	Rc	F1
blue	0.64	**0.76**	**0.70**	**1.00**	0.52	0.69	0.40	0.27	0.32	0.58	0.44	0.50
volleyball1	0.07	0.23	0.10	**1.00**	**0.46**	**0.63**	0.00	0.00	0.00	0.33	0.43	0.38
squash	0.00	0.00	0.00	0.00	0.00	0.00	**0.26**	0.84	**0.40**	0.22	**0.76**	0.34
ping_pong_paint	0.03	0.04	0.03	**1.00**	**0.89**	**0.94**	0.59	0.67	0.62	0.00	0.00	0.00
softball1	0.00	0.00	0.00	0.67	0.15	0.25	**0.83**	**0.83**	**0.83**	0.55	0.67	0.60
more_balls	0.00	0.00	0.00									
tennis2	0.00	0.00	0.00									
ping_pong_top	0.00	0.00	0.00	**0.93**	0.88	**0.90**	0.56	**0.99**	0.72	0.00	0.00	0.00
hockey	0.00	0.00	0.00	**1.00**	0.16	0.28	0.24	0.87	**0.38**	0.20	**0.92**	0.33
darts_window1	0.00	0.00	0.00	0.25	**0.50**	**0.33**	**0.33**	0.33	**0.33**	**0.33**	0.33	**0.33**
tennis_serve_back	0.00	0.00	0.00	0.29	0.06	0.10	**0.35**	0.69	**0.47**	0.26	**0.70**	0.38
volleyball_passing	0.24	0.24	0.24	0.22	0.10	0.14	0.20	0.16	0.18	**0.85**	**0.98**	**0.91**
frisbee	0.00	0.00	0.00	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	0.95	0.95	0.95
tennis1	0.00	0.00	0.00	0.00	0.00	0.00	**0.33**	0.41	**0.37**	0.20	**0.92**	0.33
darts1	0.00	0.00	0.00	**1.00**	0.27	41.7	0.37	0.63	0.47	0.33	**1.00**	**0.50**
atp_serves	0.00	0.00	0.00									
ping_pong_side	0.00	0.00	0.00	0.12	0.07	0.09	**0.45**	**0.79**	**0.58**	0.00	0.00	0.00
tennis_serve_side	0.00	0.00	0.00	**1.00**	0.59	**0.74**	0.67	0.77	0.71	0.35	**0.86**	0.50
william_tell/archery	0.00	0.00	0.00	0.00	0.00	0.00	**0.25**	0.20	0.22	0.18	**1.00**	**0.32**

Table 1: Our current results compared to the existing methods. Best results are highlighted in bold.

4 Challenges

We have faced several challenges on our project so far. The first challenge was with the synthetic dataset generator. As mentioned above, we had originally planned on using existing dataset generator code from the authors of [2]. Although we received initial confirmation from the authors that they would be willing to share the code, we have unfortunately not yet received the code or any response to a follow-up email. As a result, we are not considering this to be a viable option going forward. To overcome this challenge, we will continue modifying the dataset generator provided online by the authors of [3] to represent the FMOs correctly.

Setting up the computing environment for running the models also proved to be a challenge, albeit a less significant one. Since we would like to use the CRC for calculating the inference time of our method, we wanted to make sure that we felt comfortable navigating its servers and submitting jobs to its machines. However, there was a more considerable learning curve with the CRC than with Colab, which has a much more intuitive feel that we came to understand through working on Homework 2. As a result, more time was spent figuring out how to use the CRC machines than we had originally allotted on our Gantt chart. An updated Gantt chart can be seen in the appendix.

Another challenge faced was getting the data into the proper format that the RVOS code was expecting. The code was expecting the segmentation masks to be in the DAVIS format. Although the masks appear to be RGB where the object is red and background is black, the actual format is a PNG image with only one channel but a palette that specifies the colors. It took us some time to realize this and figure out how to convert our data to the correct format. The model also expected the filenames and directory to be set up in a very specific manner, which took a nontrivial amount of time to fix. Finally, we had an issue with our images being too large. We were getting "CUDA Out of Memory" errors, even after decreasing the batch size to 1. Eventually, we realized that the issues were due to our images were about four times larger than those in the DAVIS format. We had to resize our images to be smaller in order to avoid this error.

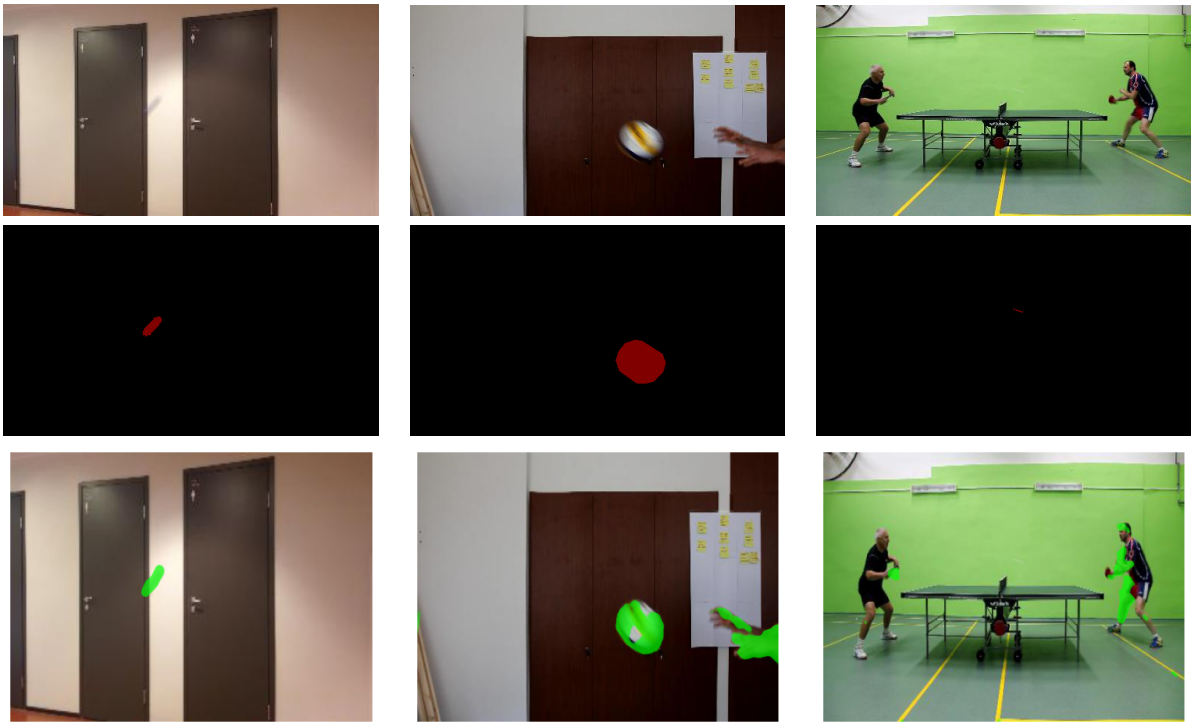

Figure 6: Current results. Top row: input video frame. Middle row: ground truth segmentation. Bottom row: result from our method.

5 Current Results

The results from our trained model are summarized in Table 5. In some cases, our method performs well, particularly for the blue ball video. Our method has higher recall and F1 score than the existing methods for this video. A sample frame from this video is shown in the left most column of Figure 5. In many cases, however, our method detects the FMO, but also predicts other non-FMO objects to also be FMOs. An example of this is that the network detects the moving hands in the volleyball1 video in addition to the ball (shown in the middle column of Figure 5). Unfortunately, in many cases our method does not detect the FMO at all. This is especially true for small FMOs. An example of this is shown in the right-most column of Figure 5 with the ping_pong_side video.

Although the quantitative results look very poor at the moment, we do see some promising results qualitatively that are not reflected in the metrics. For example, in the darts1 video, our model picks up the dart in almost all the frames. However, it also picks up the moving arm of the dart thrower. Because each of the metrics is based upon IoU to calculate TPs (rather than a pixel-wise definition), the predicted segmentations do not meet the IoU threshold. This example is representative of a noticeable trend in our results. We have observed that our model is detecting moving objects that aren't the object of interest. Intuitively, this makes sense, as the original RVOS code is designed for image segmentation of any moving object. Additionally, in our current synthetic dataset, there are no examples of moving objects which are not the object of interest. Thus, it makes sense that the model has learned to find *any* moving object rather than just *fast* moving objects.

6 Plans

6.1 Model Tuning

We will be generally adjusting different hyperparameters for training the model, such as the number of epochs and the learning rate. We will also try changing the kernel size of the convolutional LSTM. A larger

kernel should be better at capturing fast motion [10, 11]. We will also be exploring specific components of the RVOS architecture including the impact of each activation function, such as hyperbolic tangent and sigmoid. Changing the image sizes and padding inputs to each activation function could have an impact on the overall architecture.

6.2 Dataset Generator

We will try to improve upon the synthetic dataset that we have used so far. During training, we found that our model was overfitting. Figure 5 shows the loss curves we observed during training and validation. We will try to address this issue by generating a larger synthetic dataset. More data should allow the model to learn more general features instead of "memorizing" the training data.

We will also try to improve upon the dataset generator to make it more realistic. We will explore how to make the trajectories more consistent from frame to frame so that motion is more reasonable from a physics standpoint. In addition, we will also incorporate slow moving objects into that background which are not the objects of interest. This will hopefully address the issue of the model picking up all moving objects rather than just fast moving objects. We will accomplish this by using successive video frames from the VOT dataset as the background, rather than a single frame.

6.3 Trajectory Prediction

The Convolutional LSTM in the RVOS model currently uses segmentation to predict a single frame. We can to expect the network to predict the location of an FMO at a certain number of frames ahead relative to the initial frame. The simplest way to predict where the object will be in a few frames is to break down the input and ground truth videos into multiple corresponding videos, by reordering the frames (e.g. instead of using two dimensional data from a frame and temporal data from the history of the frames to predict where the ball is in the next frame, we set $k = 2$ to predict two frames ahead. We use frame 1 to predict frame 3, and frame 1 and 2 to predict frame 4, etc.). We plan to analyze how well our modified RVOS performs when the spatial features are further between each frame (introducing k means FMOs are further apart each frame). Training a network to predict object locations multiple frames ahead resembles the idea of predicting the location of an object that has moved far ahead in a frame. The idea is to remove the frames in between the input frame and the predicted frame.

A more sophisticated alternative would be keeping the video inputs as is but modifying the encoder so that the features for every specified number of frames are reordered similarly to how the frames were reordered. The encoder would be given a certain number frames to skip, k, and it would reorder the features, f_i. The decoder would correspondingly reorder the comparisons with ground truth, so we plan to update the loss function to account for the frames skipped. This method would reanalyze the frames and initially disregard the frames in between the input frame and the predicted frame. Complexity and potentially accuracy can be introduced by connecting the LSTM to the frames in between the input frame and the predicted frames. In other words, learning from the previous prediction would be beneficial to the network. The first frame can be the seed to predict the object k layers ahead, but the second frame prediction of $k + 1$ frames ahead can be optimized with the knowledge of how far off in both vertical and horizontal directions the previous predictions are from the ground truth. This bias, which relies on the history of the trajectory, adapts the network to the trajectory of the detected FMO.

Figure 8 intuitively shows how ConvLSTM can have an additional RNN connection to predict trajectory. The inputs of the network rely on the hidden states from the previous frame and the spatial hidden state to identify the object of interest in each frame. Our method's goal is to change the order of theses input features as described above, and connect each frame prediction to learn from the previous prediction using the same network. After the logic is rearranged, the next challenge would be training the weights in the model to predict accordingly.

6.4 Loss Function

Since we have noticed that our model is segmenting multiple other objects instead of simply the FMO, it would be worth further changing the loss function to address the confusion within the network. We plan to

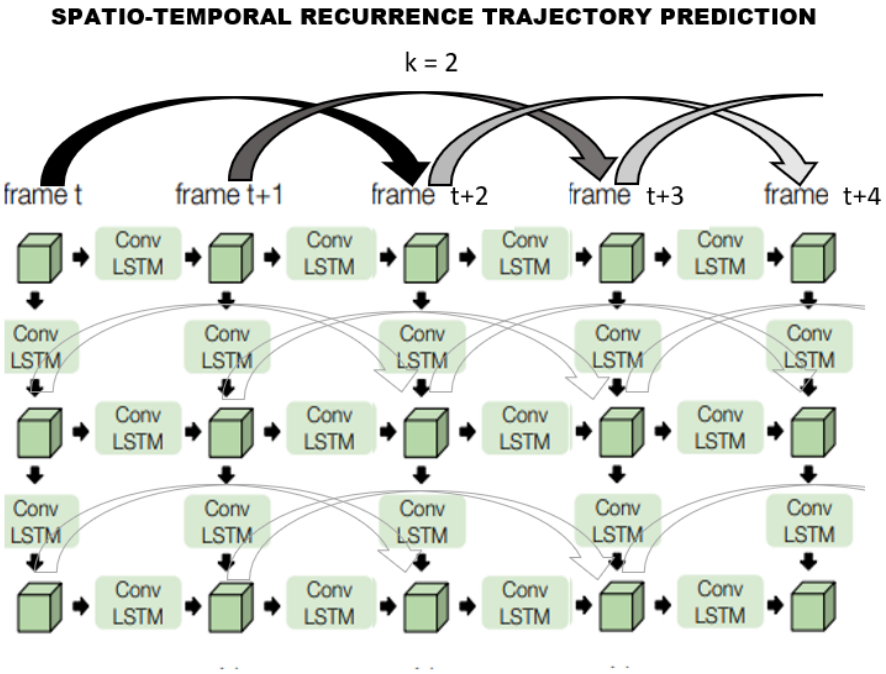

Figure 7: Visualization of predicting two frames ahead while updating predictions within an Conv LSTM [4].

add a penalty to the loss based on the number of connected components exceeding 1. This penalty will be added to the existing IoU loss. Our intuition is that this will force the model to learn to only segment one (or zero) connected components, because the ground truth data in the FMO dataset used focuses on single FMOs at a time.

References

[1] Denys Rozumnyi, Jan Kotera, Filip Sroubek, Lukas Novotny, and Jiri Matas. The world of fast moving objects. 07 2017.

[2] Ales Zita and Filip Sroubek. Learning-based tracking of fast moving objects, 2020.

[3] Denys Rozumnyi, Jiri Matas, Filip Sroubek, Marc Pollefeys, and Martin R. Oswald. Fmodetect: Robust detection and trajectory estimation of fast moving objects, 2020.

[4] Carles Ventura, Miriam Bellver, Andreu Girbau, Amaia Salvador, Ferran Marques, and Xavier Giro i Nieto. Rvos: End-to-end recurrent network for video object segmentation, 2019.

[5] Amaia Salvador, Miriam Bellver, Victor Campos, Manel Baradad, Ferran Marques, Jordi Torres, and Xavier Giro i Nieto. Recurrent neural networks for semantic instance segmentation, 2019.

[6] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

[7] Alexandre Xavier. An introduction to convlstm, Apr 2019.

[8] Jan Kotera, Denys Rozumnyi, Filip Sroubek, and Jiri Matas. Intra-frame object tracking by deblatting. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct 2019.

[9] Denys Rozumnyi. rozumden/fmodetect.

[10] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[11] Sebastian Agethen and Winston H. Hsu. Deep multi-kernel convolutional LSTM networks and an attention-based mechanism for videos. *CoRR*, abs/1908.08990, 2019.

7 Appendix

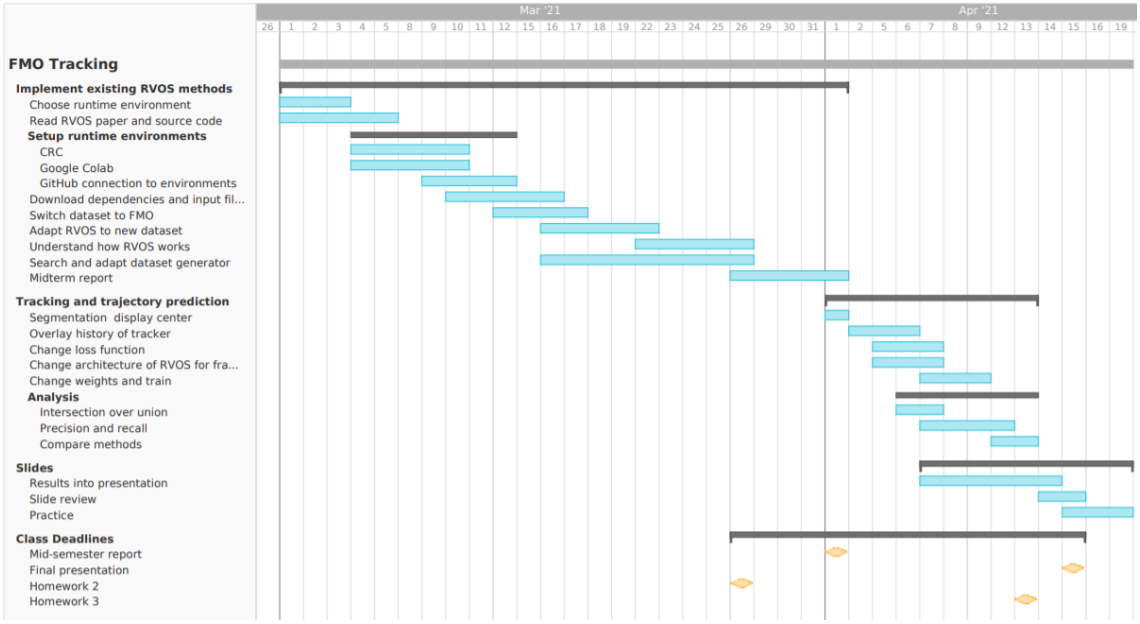

Figure 8: Updated schedule.