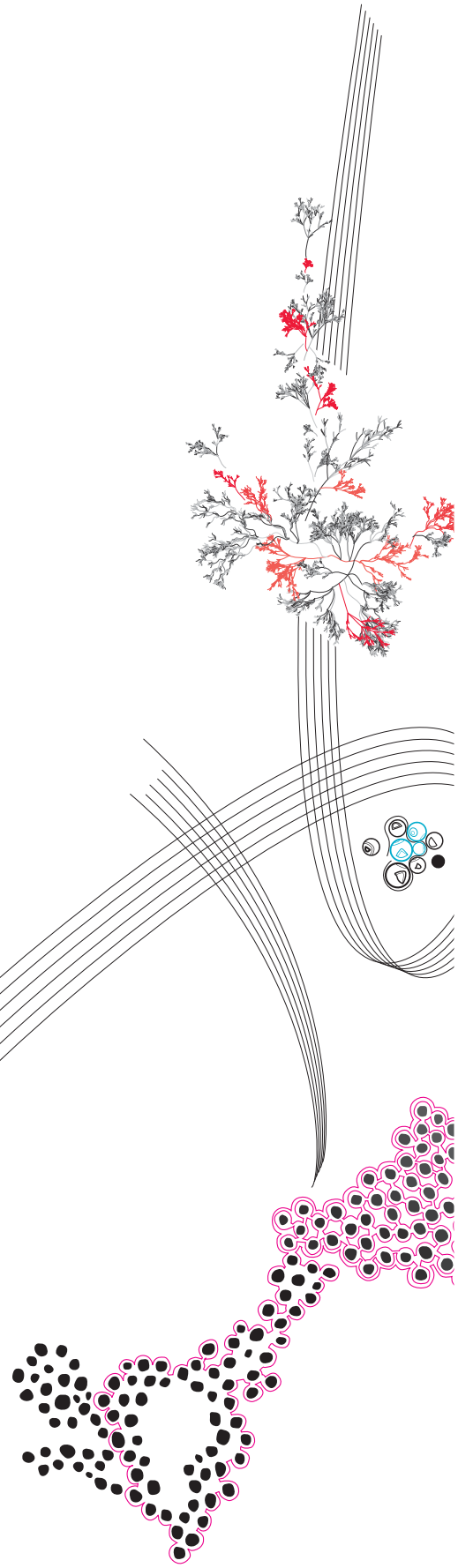


MSc Thesis Systems & Control

Interactive Segmentation in Space-time Memory Networks

Jochem Hendriks

Examination committee:
Dr. ir. Mark Vlutters (daily supervisor)
Prof. dr. ir. Massimo Sartori (chair)
Dr. ir. Momen Abayazid

April, 2023

Document number:
BE-919

Department of Biomechanical Engineering
Faculty of Engineering Technology
University of Twente

UNIVERSITY OF TWENTE.

Abstract

The introduction of memory networks was a major development in Semi-automatic Video Object Segmentation. In all these networks, frames are encoded into an embedding space using a convolutional neural network. Due to this, the encoded features are abstract and cannot be changed easily afterwards. Modifying the initial selection while retaining information from the already made memory to help in segmentation was previously not possible.

In this research, two novel methods were devised that allow for the combining of memories from pre- and post modification of the tracked object. This allows the user to define (partial) objects that can be removed or added to the initially selected object. One of the methods combines in the encoded feature space whereas the other combines in the probability space of the decoder outputs.

Tested on a made data set, both methods were able to segment in specific cases where the unmodified network fails, although the larger memory of the methods generally did not result in more accurate segmentation. The first devised method proved too inconsistent for practical use but the second performed well.

The validity of using memory from one stream of stereo vision to segment both streams was also tested. Although the made data set was limited in size, results indicate this is possible.

1 Introduction

Humanoids are robots resembling the human body in various factors, such as shape. As they are mainly designed to do physical tasks humans normally do, they have wide applications in many fields like healthcare, security or factory line work. Up until now robots have mostly been used for specific tasks but they are steadily becoming more complex and better at more generalized tasks. One such humanoid located at the University of Twente's Nakama Robotics Lab is Eve [1]. Eve is a full-sized humanoid from the company Halodi designed for use in a wide range of fields, including retail, logistics, security and healthcare.

FIGURE 1: Humanoid Eve

Similar to humans, perhaps the most important way these robots obtain information from the environment is through vision. Optic sensing can yield information for locating, environment mapping, object detection and more. Extracting useful information from an image or a sequence of images through the use of software is one of the major tasks in the field of computer vision. Although the field is not new, it is an area of study in which the use of neural networks has revolutionized the field, and many tasks which were previously considered very hard are now performed easily. Neural network are able to extract complex features without these features needing to be handcrafted by humans.

In many environments, a robot will encounter a variety of objects, some of which it may have never seen before. Classifying objects using traditional neural networks requires training on known objects, which means that previously unseen objects cannot be classified. It is impossible to train using all objects, yet in many cases the tracking of these unknown objects is required. Even without classification there are many uses to being able to track arbitrary objects, regardless of their class. Detecting objects without object-specific knowledge required a different training method: a network has to learn to recognize the 'objectness' instead of specified features. For example, a bottle has a distinct colour and shape compared to a table it is standing on and thus could be seen as a distinct object

even if one has never seen a bottle before.

Detecting objects and knowing their spatial location in an image (e.g. by a bounding box) already provides useful information but a more detailed representation is a mask indicating for every pixel in the image whether it belongs to the object or not. Such a segmentation has many uses, for example in video editing, background removal or shape estimation [2]. Segmenting an object from the background throughout multiple frames, either in a video or a stream, is called Video Object Segmentation (VOS), which is further divided into categories depending on the input by the user needed to specify the object to be tracked.

Automatic Video Object Segmentation (AVOS) methods require no initialization and segment based on the salient object throughout a video. Semi-automatic Video Object Segmentation (SVOS) methods require a user input on the first frame that specifies which objects need to be tracked. Typically this is in the form of a pixel-wise mask but some methods require other inputs. Other common inputs are bounding boxes defined by two clicks at the corners, short scribbles drawn by the user, or a list of positive/negative clicks which roughly indicate which regions need to be included or excluded. Interactive Video Object Segmentation (IVOS) takes this further and can handle additional user input throughout the video. These are often corrections on output masks of the network. IVOS methods usually incorporate SVOS networks into a larger framework that allows for easy segmentation of videos.

One of the main long-term goals of Eve is teleoperation wherein a person controls her and performs tasks from a distance. For this, an IVOS method is needed which allows this user to select objects live from the video stream. The user also needs to be able to correct mistakes in the output while propagating, or modify the selection in case parts of the tracked object must removed or other parts added. For ease of operating, it needs to be simple for the user to define input while the selected object must be tracked accurately.

Specifying the object to be tracked (initialization) and subsequently outputting the location of the object for all frames (tracking) can be seen as two different tasks. Some methods [3] combine both, but using a separate neural network for each of these tasks is the most common solution. Many tracking networks only take as input fully annotated masks, in which for each pixel in the initialization frame is specified whether it includes the object. The task of initialization such an initial mask can be done by many specialized networks existing [4][5][6]. These translate user input in the form of clicks, scribbles or bounding boxes to a binary mask output.

SVOS methods can be roughly classified depending on the principle behind their tracking strategy, with fine-tuning, propagation-based and matching-based being the major classes [7]. Methods based on online fine-tuning learn generic features from data sets and then fine-tune network parameters from the supplied initial mask to learn object-specific features. This online training often takes a long time, rendering these methods not very useful when speed is a requirement. Propagation-based methods use the previous computed mask to infer the mask for the next frame. The major problem for these methods is that they generally cannot handle occlusion or very large frame to frame changes properly, as in these cases the object is not clear in the previous frame. Matching-based methods construct an embedding space from the supplied mask to capture the object's features after which the new frame's features are matched against this representation to label pixels based on simi-

larities. Yet these methods often fail when the object of interest rotates or shifts such that the correspondence to its initial shape is low.

A major development in SVOS methods was the introduction of the Space-Time Memory [8] (STM) network which encodes information from not one or two, but many frames throughout a video into its memory. This allows both old and newly seen features of the object to aid in the prediction of new masks. Almost all currently best performing SVOS networks are improvements upon the original STM network [7]. Enhancements are often made on STM's non-local matching [9], frame rate [10] and lacking robustness [11]. A glaring issue of STM is its ever-expanding memory bank. Its GPU usage and a major part of the computing time scale linearly with the number of frames used for building the memory and thus the original STM cannot handle long videos (500+ frames) using consumer-grade hardware. The main focus of subsequent papers, such as [12], is thus on the handling of (potentially infinitely) long videos.

In all memory networks, frames are encoded into an embedding space using a convolutional neural network. Due to this, the memory consists of encoded features which are abstract and cannot be changed easily afterwards. This leads to issues for IVOS, such as when part of an object needs to be added or subtracted for subsequent frames, as the existing memory can not be used anymore. Using the same memory would cause the network to incorrectly believe some parts need to be tracked or some parts do not, leading to wrong predictions. Resetting the memory bank is an option but this leads to the loss of all information from the past which was the key benefit from memory networks.

For Eve, a feature that would let a user add or remove objects while still maintaining the state of the art segmentation accuracy provided by memory networks would be very useful. A user could then select and deselect items on the fly without resetting the network or running the same network multiple times in parallel. Currently, memory networks are not able to modify the object selection on the fly without either resetting their memory or giving false predictions.

The main goal of this research is to develop an architecture that allows for expanding or decreasing the selection during video object propagation in a memory network, while still using the previously made memory bank to aid in future predictions.

As Eve has not mono-ocular, but stereo vision, there may be additional benefits in using a memory network. In theory, from the memory made with images from one camera, predictions for the other camera could also be made. Segmentation on both streams can then be done in an efficient way. Such a segmentation of an object in both cameras has uses like a better or more efficient depth estimation, or 3D reconstruction of objects.

Thus, a secondary goal is to test whether memory networks can accurately segment both stereo vision streams, using a memory made from only one stream.

2 Background

At the start of this thesis the objectives were not fully defined yet and the stated research goals were decided at only after substantial literature search. This section first details the process of arriving at these objectives.

The initial aim of the research was to develop a method to segment objects throughout a video in an unsupervised manner. By looking at edges, corners, depth or the consistency of information throughout successive images, it is possible to distinguish objects [13]. Many of these methods require the object to move or they focus on segmenting the salient object in a frame [14][13], both of which are not suitable for Eve's environment in which objects can be static or many in the field of view. Also, as there is no user input to define the object to track in these methods, the exact selection cannot be modified easily. For example, at some point one may want to track just the hand instead of a full person, or a bookshelf instead of an individual book in it.

Focusing on just the initialization and tracking of objects simplifies segmentation and is better suits Eve's environment. The focus of the research shifted towards designing a framework that allows for the tracking of objects from user input where the object selection can be changed during propagation.

There exists extensive literature on the problem of tracking objects throughout frames. Thus the choice was made to modify an existing network for Eve's environment, where objects must be segmented from simple user input. For the selection of this network, a list of requirements was made to which the network must conform:

- RGB: the network must take as input a sequence of $H \times W \times 3$ input arrays, in which $H \times W$ is the image resolution and the 3 represent the three colors in RGB images.
- Segmentation: the output of the network must be a $H \times W$ binary array indicating for every pixel in the input frame whether it belongs to the tracked object or not.
- Occlusion handling: if an object disappears and reappears, the network must be able to track it without requiring initialization again.
- Speed: the network must be able to track online, meaning it can handle streaming where the output is given before the next frame arrives. For this a minimum of 5 output frames every second is set.

Besides these, a number of points are important considerations:

- Mask initialization: what does the network take in as input for defining the object to be tracked. Does it require a full binary mask, clicks/scribbles or a bounding box?
- Mask modification: is it possible to modify the object selection during propagation?
- Stereo: as Eve has stereo vision, can the network take in a pair of stereo images or is there some other way stereo images can offer an advantage?
- Number of instances: can the network handle more than one object to track? How does this influence its speed and accuracy?

Using the two lists above, a literature search resulted in three different major types of network. The first way is to use a box-initialization network, that requires only two corner points as input from the user. Such an approach is relatively simple but due to the bounding box, the object selection is less well defined and thus the tracking suffers. SiamMask is perhaps the most well known example of such a network [3]. A second option would be to use two different networks for initialization and propagation. The user can use the first network to define an initial mask which is then propagated using the second network. The separation of these two makes both more accurate as the networks can specialize more. Given a binary mask as input, memory networks are state of the art SVOS methods which show high accuracy in tracking. Lastly, some networks separate the entire field of view into different objects and possibly walls or floors, from which then the user can select one. Such methods are called Unknown Object Instance Segmentation (UOIS). Stereo vision or depth data are often used in such networks to aid in their predictions. In table 1 an overview of these three methods is given, including their relative performance on a number of criteria.

Method	**Type**	**Examples**	**Initialization**	**Complexity**	**Accuracy**	**Depth utilization**	**Speed**	**Multi-object support**
Box-initialization methods	SVOS	[3]	Bounding box	Low	Medium	No	High	No
Track using memory networks	SVOS	[8][10][12]	Binary mask	High	High	No	Medium	Partial
Segment all objects	UOIS	[15][16][17]	None	High	Low	Yes	Medium	Yes

Table 1: Overview of videos in data set

Box-initialization methods are less complex and fast compared to the other methods but suffer in accuracy. Also, for more than one object the entire network would have to be run multiple times. Memory networks provide the highest accuracy in tracking albeit run slower and require a full binary mask for object selection. For multi-object tracking, most but not all operations need to be done twice so they too suffer in speed and GPU usage, albeit less. UOIB methods excel at this but are relatively poor at accurately segmenting individual objects.

Because of their superior tracking the choice was made to focus on the memory networks method and specifically on how to modify their implementation to be able to change the tracked object on the fly. Existing networks for the task of mask initialization [4][5][6] are assumed to work well enough such that these can be used in Eve and are outside the scope of this research.

Memory networks

Memory networks are SVOS networks first introduced in [8], in which they showed major improvements in segmentation accuracy over existing methods. Previously, SVOS networks used either the previous frame or the initial user-defined frame to compare to the new frame for which a mask prediction is required. Some methods even use both, but the key idea behind memory networks is to use information from many frames to build up a memory bank such that the resulting matching is very robust. By using this richer information, the network is better at handling problems such as appearance changes and

occlusions.

The network finds similarities between the query frames and the memory frames by encoding both into a feature space which are then matched against each other. Every frame that needs to be added to the memory goes through a convolutional network after which the made feature space is simply concatenated to the previous ones.

In figure 2 an overview of the base space-time memory network is shown. Multiple frames from the past are encoded with their masks and these outputs are concatenated. The query frame is also encoded from which then a memory read operation determines which pixels in the query frame contain the object embedded in the memory.

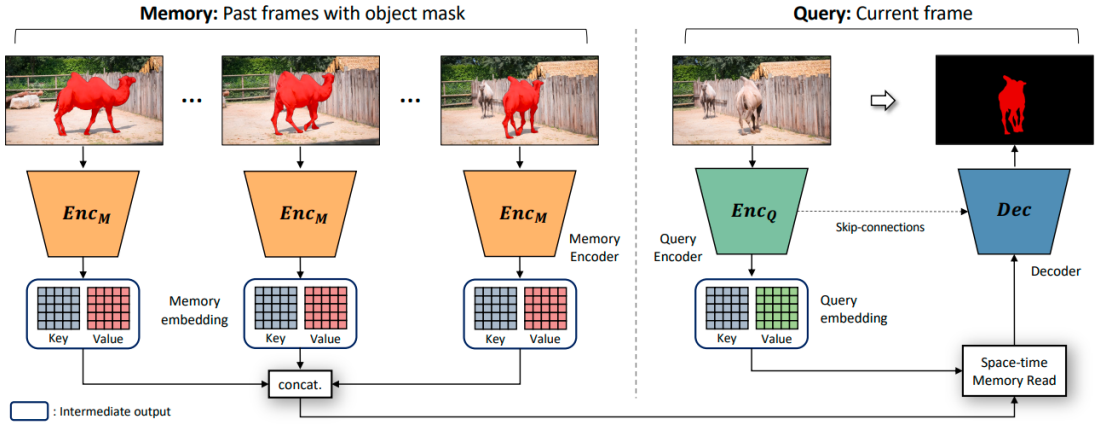


FIGURE 2: Overview of base space-time memory network, from [8]

Encoding

Encoding an image into the feature space gives a high-level representation of the image such that the network knows which features are shown where in the image. Features here can mean to simple colors or edges but also complex combinations of these such as digits or faces. Two different encoders are used: one for frames that are added to the memory where the binary segmentation mask is known, and one for incoming query frames. Both encoders are often small modifications on the structure of existing convolutions networks such as [18], as these have been shown to encode complex features well. For the first encoder, an additional input layer is added to the existing three input layers (for RGB) to incorporate the input mask. For both encoders, two parallel convolutional layers are added at the end such that they output two feature maps, called **keys** and **values**. These are 3D tensors with shape $H \times W \times C$ where H and W are the height and width of the output feature channels and C is the number of channels. H and W are often of size 54x30 to retain the approximate common display ratio of 16:9 while down scaling enough such that the resulting feature space does not become too large. When these variables are too big, computational speed suffers and hardware memory limits may be exceeded. Yet, a higher resolution would result in more accurate segmentation so a balance has to be found. Keys and values from multiple frames are added to the memory by concatenation to a fourth dimension such that the total keys/values in the memory are of shape $H \times W \times C \times T$. This T is then equal to the number of frames encoded in the memory.

Memory read operation

Keys are computed to see which places in the query image corresponds to which places in the memory image. For example, a channel might represent green edges from which it is used to see if green edges in the query image are present somewhere in the memory too. A visualisation of one key channel for two different images is shown in figure 3. It can be seen the channel represent some abstract feature present in the input images but it is unclear exactly what specific feature.

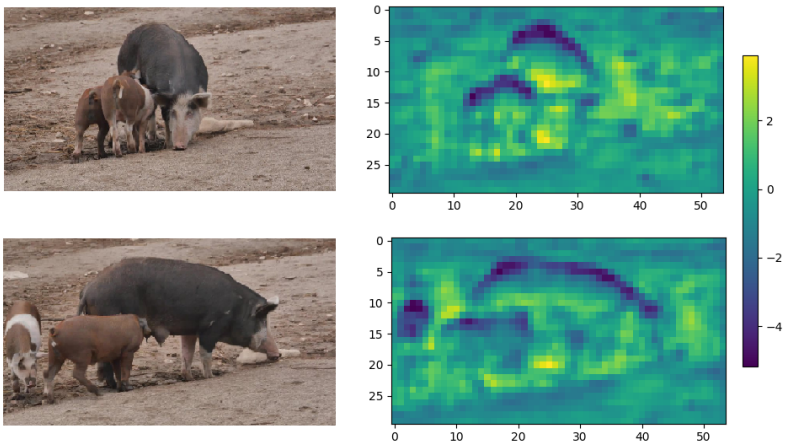


FIGURE 3: Example of an encoded channel from two different input images

Reading the memory to see which features in the query image correspond to the object of interest in the stored memory can be seen as a combination of basic tensor operations. A schematic overview is shown in figure 4 with four steps shown in red. First, at step 1, every pixel in the query key is matched against every pixel in all the memory keys across the channels, resulting in a $HW \times THW$ matrix. This is then normalized by the softmax function (step 2) such that the output is a matrix of similar shape but in which now every element is between 0 and 1 and every row sums up to 1. Now for every pixel in the query frame it is known which pixels in the memory corresponds with it.

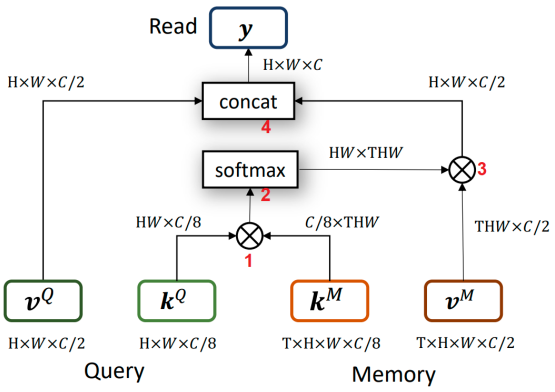


FIGURE 4: Overview of read memory operation, adapted from [8]

The memory values are used to store information of whether features belonged to the mask. Pixels in the channels have (depending on the channel) either highly positive or negative

values if they were inside the mask when the image was encoded. At step 3, all the values in the memory are multiplied with the output of the key matching such that the result is a $H \times W \times C$ matrix where for every pixel in the channels represents where in the query image the objects encoded in the memory is located. This is concatenated to the query values (step 4) as these aid in accurately reconstructing the object shape.

Decoding

The final output of this read operation is the input to the decoder. This module compresses the channels while upscaling the pixels through a series of modules such that the final output is a mask with similar height and width as the input images. The final values are probabilities between 0 and 1 which serve as the one of the inputs to the memory encoder for the next iteration in case the image is subsequently added to the memory. For visualisation and evaluation the values are rounded so the final output mask is binary with all pixels categorized either as belonging to the object(s) or not.

Evaluation

SVOS methods are commonly evaluated on two different metrics [2].

- **Intersection over Union** (IoU, also known as the Jaccard Index): a value given in the range of [0,1] where 1 is the best possible score. It is calculated by $\mathcal{J} = |\hat{Y} \cap Y| / |\hat{Y} \cup Y|$ in which $\hat{Y}$ is the estimated binary mask and Y the ground truth. This is calculated for every frame from which then the average is taken. A visualisation is shown in figure 5. A limitation of the IoU is that poor segmentation of thin long objects without much surface area has a relatively small penalty compared to improper segmentation of large objects.
- Frames per second (sometimes seconds per frame): a measure of speed which shows how many frames the network can handle every second. In practice these are not deterministic and heavily dependant on the hardware used. For a proper comparison, different networks must be run on the same computer.

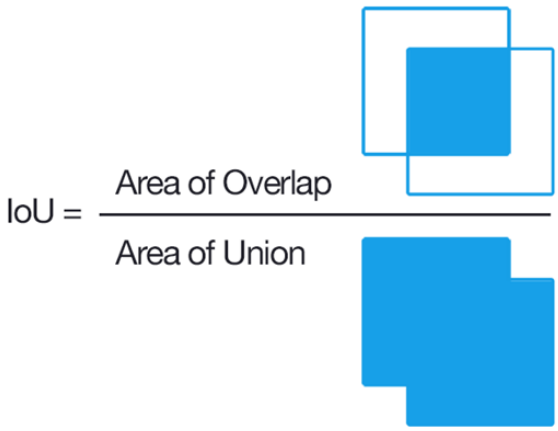


FIGURE 5: Visualisation of Intersection over Union

A third metric that is commonly used is the boundary accuracy. The value indicates how well the outer boundaries of the estimation and ground truth match. An IoU of 1 will

always have a maximal boundary accuracy but for some imperfect outputs it can give a better representation of how well objects are segmented. It addresses the above mentioned issue with the IoU although it suffers from a different problem. Very small mistakes in segmentation may be penalized heavily as these mistakes can cause large changes in the boundary. For this reason sometimes the mean of the IoU and the boundary accuracy is used. However, these are not as common as IoU because computing the boundary accuracy is more difficult as it requires bipartite graph matching. For this reason, the boundary accuracy is not used in this research.

3 Methods

The goal was to develop a memory network framework that allows for modification of its object selection while retaining the memory from the past. For this task, there are many possible approaches. The choice was made to build solutions by altering the simplest memory network, STM. Subsequent networks include additional features, complicating the process of implementing changes. Although the final resulting network might have better performance using a different backbone network, the simplicity helps in showing the proof of concept works.

The first step was to make an overview of which methods are possible and what advantages and disadvantages they offer. The result was a list of five options. The first three options aim to combine memories made from both pre- and post object modification. When adding an object, the memory combining must be done such that the output mask contains both the initial and the added object. The fourth and fifth methods aim to alter the existing memory such that the added object is now also included, causing the network to segment both objects.

1. **Combine in feature space.** Make a separate memory each time the selection is altered, and combine these tensors somewhere in the memory read operation. Slicing a tensor across the T dimension at frame number s results in one tensor that includes frames $(1, 2, ..., s-1, s)$ and one tensor with frames $(s+1, s+2..., T-1, T)$. Then multiple different memory banks are obtained, each having a different object encoded. If these tensors and thus different (sections of) objects can be combined somewhere in the read memory operation, there is no need for additional encoding and/or decoding. However, it is not known yet if there is a relatively simple way of combining the key value tensors from different objects, and whether such a method would retain high accuracy.

2. **Combine after decoding.** Slicing similar to the above option, keeping a separate memory bank for each (partial) object but now decode each one. This way, multiple segmentation masks are obtained. Combining these by taking the union for addition and the intersection for subtraction gives the final output mask. In this method the combining is done after the decoding step in figure 2. The major disadvantage is that decoding and subsequent encoding would have to be done multiple times. Especially if additional corrections are made to the tracked object and many different memory banks are obtained, leading to a major drop in fps performance.

3. **Combine using additional small network** Train a small neural network that takes as input two tensors which are inputs to the decoder from different memories and make the network output a tensor that when decoded, fully segments both objects. This combining of different memories is done at the final step of the read memory operation. Although this solution would require the design and training of an an additional neural network, it likely can be a very shallow and small net. This solution would have little impact on the overall accuracy and speed of the main network.

4. **Re-encode memory frames with insertion.** Storing the original RGB frames, manually inserting ('photoshopping') the added second object in each of these frames and re-encoding everything would likely yield very good segmentation. Although the re-encoding of many frames at once will cost significant time (buffering), after this

is done the network will run as fast as it originally did. Encoding takes roughly 30% of the total computation time in the original STM [11] so as long as the number of frames in the memory is not too large, the buffering time is manageable. In the read memory operation every query pixel is matched against every memory pixel (non-local matching), thus the location of inserting the second object in the original frames has no influence. However, this is true for the original STM but not necessarily true for some extensions which do use local matching. Another problem is subtraction of (partial) objects in the tracked selection, which is unclear how to handle using this method.

5. **Run backwards in time.** Keep the original RGB frames, reset the network's key/value memory and then using the latest mask as the initial mask, run the network backwards in time trough the past frames. When this is done, use the new memory as normal to predict the next mask. This may lead to issues if the added object is not visible in a significant fraction of the memory frames, as it may then not be represented enough in the memory for proper estimation of the next query frame. A possible option is to not include past frames in the memory when the added object is not present which can be done by comparison with the original masks in the memory. There may also be mistakes in the predictions of the past frames when running backwards, leading to the memory containing wrong information.

Due to time constraints it is not possible to implement all of these options. Thus, from the above list two methods were picked for implementation. The first option of combining in feature space has little to no loss of speed, and thus seems the preferable option. Combining after the decoder is relatively straightforward to implement and offers the tracking accuracy of the original network. This also provides a good baseline to compare the other method to.

The third option requires generating a sizeable data set, training on this data and subsequent tuning of network hyperparameters which is considered too elaborate for this research. Modifying the mask selection in both the fourth and fifth option leads to large buffering times before the next output can be given, especially if the memory is large. This is a significant disadvantage for Eve's live teleoperation goals.

Method 1: Combining in feature space

For the addition of a second object (here called object B) to an object already in the memory (object A) somewhere in the feature space, multiple attempts were made.

The first attempt was just encoding only B and adding it to the memory. The total memory now contains keys/values made from frames of A and keys/values made from frames of B. The problem with this approach is that is leads to inconsistent outputs. Memory values from both A and B are read but this causes the decoder to often assume that the objects are both not in the memory. This was tested on multiple videos/objects and the results were very inconsistent: sometimes the mask was the correct output (both objects segmented) but more often it was partial, as shown in figure 6, or even no segmentation occurring at all.

The number of memory frames of A relative to the number of memory frames of B seemed to influence the output, as having little frames of one object and many of another object in the memory causes only the second object to be segmented. Another attempt was then to weigh both keys/values based on the number of memory frames, but this still had issues. If a certain pixel in a memory frame belonging to object A has a value of 2 and if one in

FIGURE 6: Wrong output from simple addition

another frame belonging to object B equals 3, and they're both equally weighted by the output of the softmax, the final segmentation may only show object B. In the memory values, all high values contain the object but they can still differ somewhat. Even if they were equal, the softmax may cause some to be values to be weighted more than others, such that one may not be segmented.

Increasing all memory values to ensure the input to the decoder has high values for both objects also failed to work as the output becomes 'jittery' with small spots of both objects being segmented. Decreasing all memory values causes the little to no segmentation at all. The decoder apparently segments based upon relative values instead of absolute. Simple addition somewhere in the key/value feature space does not seem to work and a more complicated method is required.

A possible way to combine the tensors is to use a mathematical maximum method. Right after step 3 in the memory read operation (see figure 4), there is a $H \times W \times C/2$ tensor where each feature channel C shows which pixels in the query image match the object embedded in the memory. High values mean the object is present and low values indicate it is not. By taking the maximum of both memory tensors for every pixel, the resulting tensor will have both objects. An illustration of this is shown in figure 7. Here, the first two images are input tensors from which combining results in the right tensor where both objects can be seen present. For this specific channel the objects are clearly visible, but for most others the encoded features are more abstract yet the principle of combining using this method stays the same.

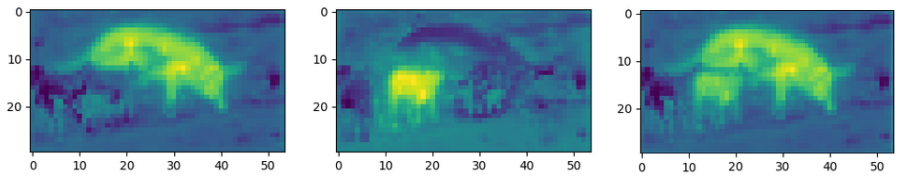


FIGURE 7: Addition of channels by taking the maximum

There are two main issues with this solution. First, the maximum between pixels of both tensors is taken for noise too, which means it will be amplified. For the objects themselves it is also imperfect as one object may have higher values than the other causing the de-

coder to only output the highest object. A second issue is that not all channels have high values for the object being present and low values if it is not. Roughly half of the channels show the opposite behaviour where positive and negative are switched. In these cases the minimum can be taken instead of the maximum. However, the specific channels in which this occurs are not consistent and differ somewhat based on the input image. This means that when combining the memory, a check will have to be done to find out over which channels the maximum function and over which channels the minimum function should be applied.

A simple check is to find the median value of all pixels in a channel. Channel pixels of the object have an sign opposite of pixels outside the object. Assuming the tracked object is not encompassing more than half the of the query image, the median should give an indication of whether the maximum or minimum function must be used.

Below the pseudocode of the implementation of this method is shown. It starts with two $H \times W \times C/2$ tensors called mem_1 and mem_2, and outputs a tensor mem_{new} of similar shape.

function Combine1(mem_1, mem_2);
$\text{mem}_{new} \leftarrow$ zeros(H,W,C/2);
for *c in channels* **do**
  if *sign(median(mem_1)) < 0* **then**
    mem_{new}[:,:,c] $\leftarrow$ maximum(mem_1[:,:,c], mem_2[:,:,c]);
  else
    mem_{new}[:,:,c] $\leftarrow$ minimum(mem_1[:,:,c], mem_2[:,:,c]);
  end
end
return mem_{new}

Algorithm 1: Combining memories method 1

Not shown in the pseudocode, but in case the memories need to be subtracted (for removal of an object) the maximum and minimum switch. For deciding on which channels to apply which function on, other methods than the median were tried. On simple test videos, using the mean or top-k values($k = 1, 2, 5, 10, 100$) showed poorer consistency in output masks compared to using the median value.

When a third memory is added or subtracted the above algorithm can be used multiple times with the memory from the previous iteration as one of the inputs. This allows for sequences where some objects are added and some objects subtracted.

The full steps of the final modified read memory operation are shown below.

- Calculate key similarity (similar to step 1 in figure 4)
- Split resulting tensor into slices, based on which frames in which memory
- Apply softmax function to each slice (step 2)
- Split memory values into similar slices
- Calculate product of each key slice and memory slice (step 3)
- For the resulting slices, apply *Combine1*

- Concatenate query values (step 4)
- Decode to obtain mask

Once memory banks of object A and B are combined the resulting mask contains both objects (A+B). For subsequent storing of frames the input mask is now A+B. This would mean that for the iteration afterwards there are now 3 memory banks (A, B, A+B). Ignoring the last part would mean no new frames can be added to the memory but using it makes combining involve an additional step.

To solve this issue, no separate memory bank is made for B, only for A+B. For example, when a glass is added to an already tracked table, the second memory bank will be of glass+table instead of just table. Due to the maximum function, false negatives of the table in one of the memories will be compensated by the other but false positives in either will end up in the output. For channels in which the minimum function is applied, false negative predictions may end up in the final result whereas false positives are ignored. Because there are now multiple memory banks the chance of wrong predictions may increase but other mistakes may be now be compensated. For the subtraction of objects, the new input mask is the resulting object after removal of a part, not the removed part.

Solving the issue this way also means that when an object is added, both the original and the object to be added must be present in the frame. However, when removing an object it is not necessary for it to be visible as only the remaining parts need to be.

Method 2: Combining after decoding

The second method consists of again keeping a separate memory bank for each added object, but now decoding each one. Then, there are multiple output masks which can be combined to obtain the final segmentation mask. Two binary masks containing objects that need to be added together can be combined by taking union of both masks. Similarly, subtraction can be done by taking the intersection. However, the actual output mask of the decoder is not a binary but consists of probabilities lying between 0 and 1. For the visualisation and analysis of outputs, these get rounded to transform it into a binary map, but the output mask fed back into the network is not binary. To combine two or more probability masks, the maximum and minimum functions can again be used. If two objects in these masks need to be added, taking the maximum across both masks for every pixel results in a mask yielding both objects. If this is then converted to a binary map by rounding, the result is the same as when the union would be taken from both individually rounded masks. The same is true subtracting objects: taking the minimum yields the intersection of both objects.

Below the pseudocode of the implementation of this memory combining method is shown. It starts with two probability masks with height and width equal to the input images, and outputs one probability map of similar shape.

function Combine2(mask_1, mask_2);
if *Addition* **then**
  $\text{mask}_{new} \leftarrow \text{maximum}(\text{mask}_1, \text{mask}_2)$;
else
  $\text{mask}_{new} \leftarrow \text{minimum}(\text{mask}_1, \text{mask}_2)$;
end
return mask_{new}

Algorithm 2: Combining memories method 2

Similarly to method 1, when a third mask is added or subtracted the above algorithm can be used multiple times with the mask from the previous iteration as one of the inputs.

The finalized steps of the going from keys and values to output mask are alike that of method 1 with some differences:

- Calculate key similarity (similar to step 1 in figure 4)
- Split resulting tensor into slices, based on which frames in which memory
- Apply softmax function to each slice (step 2)
- Split memory values into similar slices
- Calculate product of each key slice and memory slice (step 3)
- Concatenate query values to every product (step 4)
- Decode all these
- Apply *Combine2* on each output mask from the decoder

A strategy similar to that of the first method was chosen for which mask to add to the memory bank. Instead of using just the added part for the input mask, both the original and the added part (A and A+B) are used. This comes with the effect that in case of addition false positives from one memory bank end up in the final output but false negatives do not. For subtraction of objects the opposite is true.

Evaluation

For the evaluation of the two methods, a small data set was created consisting of ten videos. These videos are approximately 50 to 100 frames long and are selected such that different kinds of challenging situations are included. In contrast to SVOS data sets, these videos have not just one user defined mask at the start but multiple throughout the video, simulating the addition or removal of (partial) objects by a user. The first 5 videos are self-made whereas the other 5 are modified from the DAVIS16 [19] benchmark set by changing the input masks. Both methods are compared to the baseline STM network, which memory is reset every time a new input mask is given. It is common to only put every fifth or tenth frame into the memory, as frames that are only a few hundred ms apart look very similar and this decreases memory build up and computing time. In this evaluation every frame is added, because for speed the networks are compared only relative to each other and memory overloading is no issue with videos of less than 100 frames.

An overview table with example frames of all videos is shown in table 2. The column addition/subtraction indicates in which frames an object is added (+) or subtracted (-). A description of every video is also given.

- **Video 1:** A guitar is tracked from which then its body is removed, leaving the neck and headstock remaining. This tests if the partial removal of an object works without cut sections reappearing again.
- **Video 2:** Three objects on a desk are added after each other, but when the third is added the first object is just outside of the image. When it appears again it must still be segmented. As the memory of the original STM network must be reset it is expected it will fail in this task whereas the two devised novel methods should succeed.
- **Video 3:** A cup and a phone case on top of it are tracked while the camera comes closer until the objects cover almost the entire screen. The camera then zooms out again. This tests whether the methods are robust such that both small and large objects are segmented correctly.
- **Video 4:** The fourth video includes a small plant in a pot standing on a piece of paper. The objects and their boundaries are more complex, increasing the difficulty of accurate segmentation.
- **Video 5:** A paper cup is stacked on top of a white block and both are tracked. A paper slowly covers the objects starting with the paper cup. When the cup is fully invisible, it must be removed from the memory such that only the white block remains. Thus after it reappears it must not be tracked anymore. This is similar to the first video but now the object to be removed is not visible during removal.
- **Video 6:** A miniature train consisting of 4 carts is riding and all four carts are first added one by after which the first 3 are removed one by one. This tests whether the segmentation is still accurate if many objects are added and subtracted.
- **Video 7:** Three moving pigs are added to the tracking separately. As the objects are somewhat similar and constantly moving, these are likely more difficult to segment correctly.
- **Video 8:** A person is sitting on a bike as it slowly moves. First only the bike is tracked, then the rider is added and finally the bike itself is removed. At the start of the video the bike is seen from the front and at the end from the side. Because of this rotation, the bike and person change shape heavily complicating segmentation.
- **Video 9:** A man grabs a cardboard box and lays it on top of two similar boxes. First only the grabbed box is tracked after which the other two are added. This video also presents difficulties as the object rotates and the man partially occludes the box.
- **Video 10:** A dog is chasing a group of four ducks. First only the dog must be tracked, then also two ducks from which one is then removed. The ducks move, are partially occluded at some times and look very similar to each other. These combined test the accuracy of the methods under complex conditions.

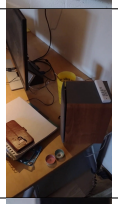
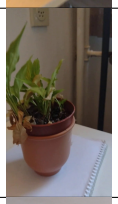
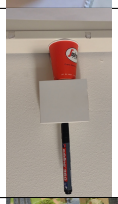
Nr.	Nr. of frames	Addition/subtraction	Example frame	Ground truth
1	100	0+,27-		
2	57	0+,13+,28+		
3	94	0+,15+		
4	56	0+,22-		
5	76	0+,13-		
6	78	0+,5+,11+,17+,37-,45-,51-		
7	78	0+,22+,44+		
8	78	0+,34+,58-		
9	49	0+,28+		
10	85	0+,19+,44-		

TABLE 2: Overview of videos in data set

Visual inspection of output frames compared to ground truth masks can give a good impression of the overall accuracy and failure cases. However, the evaluation will be also quantified using two metrics. The IoU will be used to measure the accuracy of the segmentation and the relative speed will be compared using frames per second averaged across a whole video. For the IoU, frames with masks that are user defined are not taken into the calculation.

Stereo data set

To measure how well predictions for one camera in a stereo setup can be made using memories of the other camera, a small data set was created using a Stereolabs ZED camera. Although only three videos are included, this can still give a decent indication on the validity on stereo segmentation from one stream's memory.

- **Video 1:** A large plastic pair of scissors is moved in front of the camera. It changes shape and is also out of the field of view multiple times.
- **Video 2:** A trolley with multiple objects laying on top is seen. A second object added while the initial object is just outside of the screen. Similar to video 2 in the first data set, this is a case where the original STM is expected to fail as it cannot retain information from the first object while the second is added.
- **Video 3:** A small box and a bottle are both tracked from which then the box is subtracted. What complicates segmentation in this video is the fact the both are rotated and at some point one occludes the other.

An overview table with example frames of all videos is shown in table 3. The example frames are from the left stream.

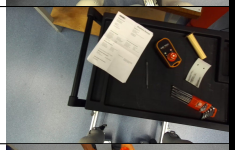
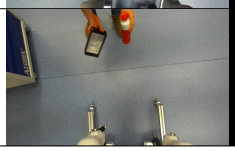
Video nr.	Nr. of frames	Addition/subtraction	Example frame	Ground truth
1	73	0+		
2	73	0+,35+		
3	73	0+,27-		

TABLE 3: Overview of videos in stereo data set

Frames from the left camera are used as input to the encoders from which the memory is used to segment the streams from both cameras. Ground truths were defined for both streams from which IoUs can be calculated. This way a comparison can be made between the left and right side to see whether the (slight) change in shape and place of the tracked objects reduces the accuracy of the predictions.

4 Results

This sections details the results of the developed methods on the two data sets. For reference, method 1 is the combining in feature space method and method 2 combines after the decoder.

The situations presented in the videos are meant to be difficult or illustrative cases and thus not proportional to typical usage. Because of this, the average IoU of all videos combined is not an ideal metric and it is more useful to look at the IoU of each video individually. Also, as the metric itself does not show what exactly goes wrong, visual inspection of the output of each method on each video provides additional insight.

The results of the evaluation on the first data set are shown in figure 8.

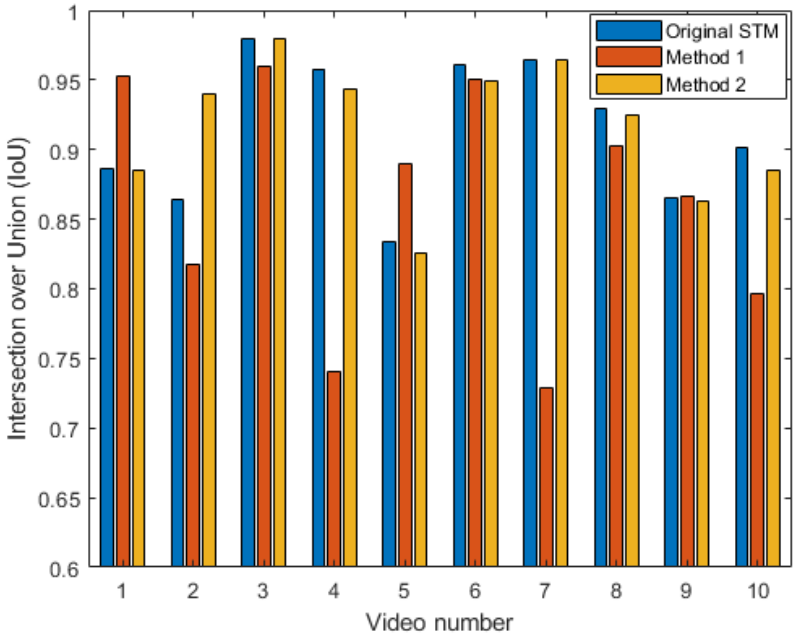


FIGURE 8: IoU results on made data set

In the first video the second method performs on par with the original network whereas the first method outperforms both. The original STM wrongly predicts part of the of the body guitar belonging to the neck and the same mistake is present in method 2 too. This can be clearly seen in some frames as shown in figure 9. However, the subtraction of the majority of the body works in all cases.

The second video is a case where the original network was assumed to fail, which it does. As its memory is reset on addition or subtraction, it is not possible to recover the object outside the image when it becomes visible again. Method 2 does succeed in this, showing its intended improvement. Even though it is present in its memory, method 1 fails to segment this object.

All three networks score IoUs above 0.95 on the third video, showing an object going from

(A) Original STM (B) Method 1 (C) Method 2

FIGURE 9: Example evaluation from video 1

small to large to small again is still segmented correctly. The second method scores slightly lower as there are slightly more small fluctuations in its output masks.

Whereas method 2's IoU was only off by 0.02 compared to the other networks in video 3, it is plain wrong on video 4. As shown in figure 10, only a small part of the paper is segmented and even some spots on the ground are false positives. The second method also segments the same spot falsely, whereas the original network does not.

All 3 networks experience difficulty on the fifth video. The original STM only segments half the paper block and also has some small regions of false positives. Method 2 suffers from the same issues, although the false positives occur in slightly different frames. The first memory combining method however, segments the white block but also a part of the paper cup, showing the subtraction was not done thoroughly. The outputs are shown in figure 11.

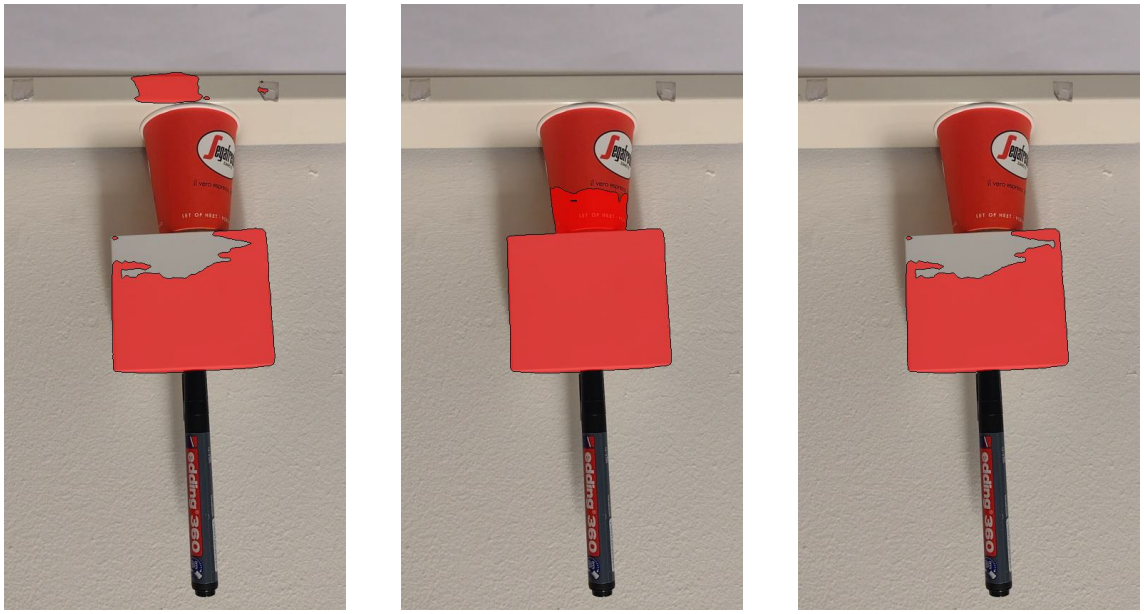
The trains in the sixth video are all segmented very well. Mistakes made due to complex shape of the individual wagons are small, leading to high IoUs. This shows both addition and subtraction of many objects works well for each of the methods.

Differences in IoUs of over 0.2 are present between the outputs of both combining methods in video seven. The original STM and the second method perform well although both show some false positives due to the complexity of the objects. Method 2 performs poorly as it does not segment the added second and third pig at all. After the user defines the extra objects they immediately disappear, as if they were not present in the memory.

All three perform decent on the eighth video. Each method made some small mistakes due to the complex shape and rotation of the objects. The second method seems to suffer slightly more from this.

(A) Original STM (B) Method 1 (C) Method 2

FIGURE 10: Example evaluation from video 4



(A) Original STM (B) Method 1 (C) Method 2

FIGURE 11: Example evaluation from video 5

The outputs on video nine are very similar for each network. Before addition, the original network (and thus the two other methods) have trouble tracking the box due to it changing shape and being occluded. Still, in all cases the addition of the other boxes works well.

The final video again shows major differences between method 1 and 2. The original STM and method 2 both achieve good segmentation on the dog but have issues with the relatively small and identical looking ducks. Method 1 suffers from the same issue as in video seven where the object to be added disappears after a few frames.

Overall, the results of the second combining method are very similar to that of the original network with two differences. It performs slightly worse with differences in IoUs <0.02 on some videos, which are exactly the videos that involve the removal of objects. Yet it has the major advantage of being able to segment objects that were deleted from the memory in the original STM, as the second video shows. The first combining method is very inconsistent. It outperforms in terms of IoU on some videos, but this is due to the original network and the second method having large false positive or false negative areas. It performs equally or even better than the other networks in some videos but suffers severe issues in other videos as entire added objects are sometimes not segmented at all.

Speed

The speeds of the three tested networks are shown in figure 12. These are calculated from the total run times and the number of frames in the video. Thus, these indicate an average across all frames. In practise, the speed of the two memory combining methods depends on the number of objects inside the memory which means these are slower towards the end of a video.

The original STM suffers no loss of speed when additional objects are added or subtracted and is therefore always the fastest option. Due to the expanding memory, it is slower on long videos when many frames are already encoded and the key and value tensors are large. The two combining methods are both slower, although in videos where only one modification is made to the tracked object (videos 1, 3, 4, 5, 9) the difference is smaller, roughly a 30% reduction. This is opposed to video 6, in which both memory combining methods suffer heavily in speed. Although method 2 needs to decode multiple times, it is seems to be faster than method 1 in all cases.

Stereo

The IoU results of the stereo vision tests are shown in figure 8.

In video 1, all methods output exactly the same but the right stream seems to have a slightly lower accuracy compared to left. All IoUs are not very high (< 0.75), as the scissors are often blurry, and thus the exact definition of where the object starts is not as well defined as in other videos.

In the second video, the IoUs of both combining methods (0.96) are much higher than that of the original network (0.86). Visual inspection shows both track the initial object well but, as shown in figure 14, the original network is unable to segment the second object at all.

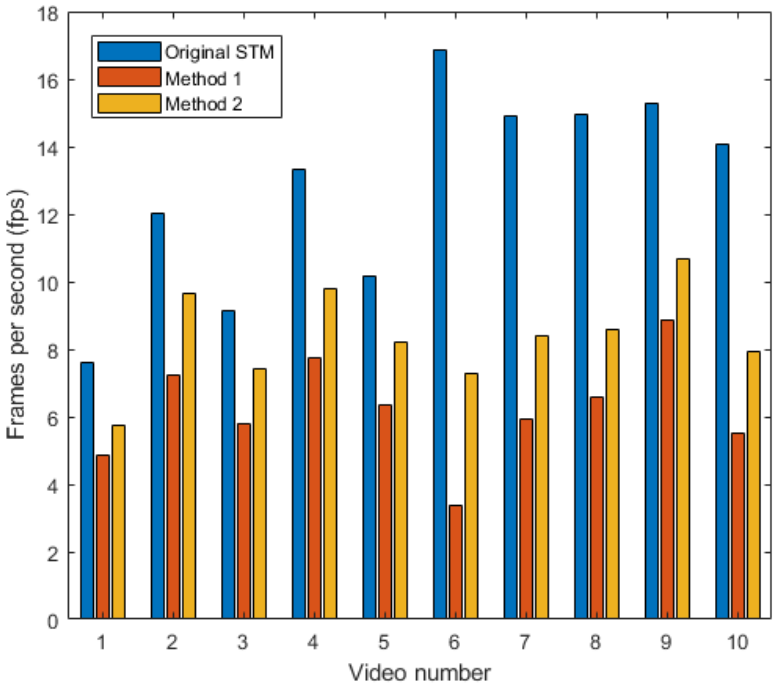


FIGURE 12: Average frames per second of methods on data set

In the third video, all methods perform well at the start and track the initial object correctly. All three also have trouble accurately segmenting the remaining bottle in the final frames. STM performs best with method 2 being close as it segments the bottle only slightly worse. Method 1 fails to fully remove the first object in one of the final frames. It also has significant trouble fully covering the second bottle. Figure 15 shows an example frame in which the differences in output are visible. For the right stream, the IoUs of the original STM and method 2 are slightly higher than than that of the left stream, whereas method 1's accuracy slightly suffers.

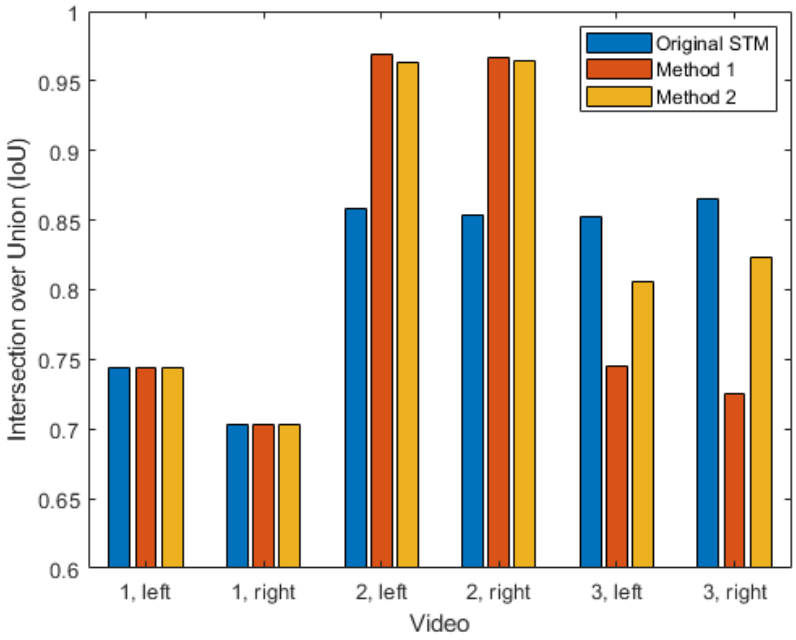
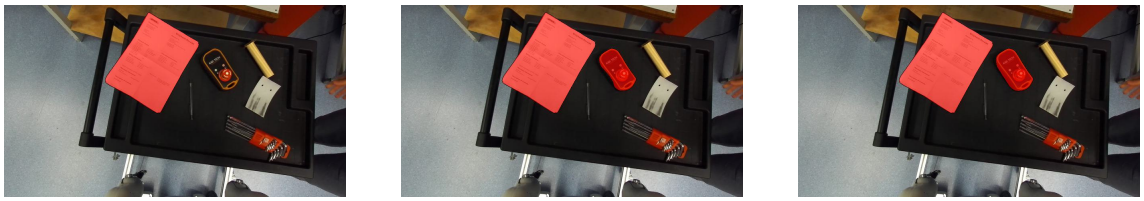


FIGURE 13: IoU results of stereo test



(A) Original STM (B) Method 1 (C) Method 2

FIGURE 14: Example evaluation from stereo video 2

5 Discussion

The main goal of this research was to design an architecture that allows for the modification of the user's selection during mask propagation while still using the old memory bank for aiding in predicting. Previously, this was not possible as memory networks cannot handle such modifications of their memory or allow multiple memory banks to be combined. The results indicate the two devised methods are capable of generating outputs such that multiple objects can be combined, showing the initial object selection can be modified successfully.

In terms of accuracy, STM and the second memory combining method are very close to each other. This difference being small makes sense as the decoder output mask from method 1's last memory slice is the same as the final output mask of STM. The advantage of combining is clearly shown in video 2 where the original STM is incapable of segmenting all objects whereas method 2 does this correctly. However, besides video 2, the IoU of the second method is always equal or slightly lower compared to the original network. When adding objects in method 2, if one of the memory banks causes the decoder to output false positives, these also end up in the final mask. The same is true for false negatives when

(A) Original STM (B) Method 1 (C) Method 2

FIGURE 15: Example evaluation from stereo video 3

subtracting and as a result, the segmentation is often slightly worse.

In spite of that, this loss in accuracy could be offset by the combining methods having more frames 'stored' and thus having more material to compare to, yet this does not occur. There seem to be no cases where after addition the initial object that is now present in multiple memory banks is segmented more accurately. One reason for this could be is that the rotating objects included in the videos (video 6 to 10) do not change shape severely. If objects are static, not many frames are needed to segment the object as it always looks similar and additional memory will have little use. There are no videos in which there is a sequence of an object rotating, addition of a second object and the original object rotating back. In such a situation, the additional memory of the object before its first rotation is more likely to aid in the prediction of the final frames.

The inconsistency of the first method indicates the combining of memories does not always result in both memories being present in the new tensor. This means the min-max operation shows severe deficiencies. For finding out over which channels the maximum needs to be applied and which the minimum, using the median value of pixels is an imperfect way. The median can be positive while the encoded object also has positive values, causing the wrong operation to be applied. Also, even when the right operation is chosen, one object could have relatively higher values than the other causing the other object to not appear after decoding.

What is interesting is that in video 3 method 1 still performs well. At some point the object encompasses over half the screen meaning the median may be negative when it should be positive or vice versa. However, when two memory slices are combined, the median is calculated from the first and in the first of these slices the object was still small.

Speed

As expected, the original STM is always the fastest network. It involves no additional computations for combining and after the modification of an object its memory is reset so the memory tensors are small. This is also the reason it runs faster in videos with more frequent resets and explains the variation between the 10 videos.

The developed methods are not very far off in terms of speed compared the original STM, considering that they keep every frame of the video in their memory. The difference is the largest in video number 6 as it has the most user defined masks and thus requires the most combining operations.

Although it has to decode multiple times when multiple objects are in its memory, the second method is nonetheless faster than the first method. This is because finding the maximum or minimum and also calculating the median requires iterating over every single pixel. This needs to be done for every channel, which takes a long time, seemingly even longer than decoding. It must be said here that the coding of the implementations is not optimized for speed. Decoding takes a set amount of time which cannot easily be reduced whereas the tensor combining of method 1 can likely be sped up. For adding three or more memories, the maximum operation can be applied once over three tensors instead of two times across two. Also, calculating the median of every channel likely does not have to be done multiple times for combining three or more memories.

Stereo test

A second goal was to test whether memory networks can accurately segment in both stereo vision streams, using a memory made from only one stream. Although in theory this should be possible due to the differences between streams being small, it had not been experimentally confirmed yet. The results show the differences in accuracy are small and thus it confirms it is possible to segment both streams from one memory.

The first stereo video involved no addition or subtraction and thus no memory combining. In that case all three networks behave similarly and thus all output masks and IoUs are the same. The segmentation on the right stream seems to be slightly less accurate compared to the left stream but it is difficult to find the exact reason for this. As the view of the object differs to some degree between the two streams, predictions made with the memory of the other stream may not be ideal. However, the object is sometimes visible in only one of the streams or it may be more blurry in one of them, so a direct comparison between the IoU of left and right is not perfectly fair. This means that with minor differences it is not possible to say what exactly causes the variance between left and right.

The second stereo video again involved a case where addition was done while the originally initialized object was out of view. Therefore, it is no surprise the two combining methods perform better than the original network, which is inherently unable to segment correctly here. No differences between the segmentation accuracy of left and right are visible which shows again that segmenting using memory frames from the other eye works well.

It is surprising the accuracy of the right stream seems slightly higher than that of the left stream for the original STM and method 2. This is probably due to noise from earlier mentioned reasons. Method 1 again shows its inconsistency due to not properly removing an object, again likely caused by the maximum/minimum on the tensor not combining good enough consistently. Method 2 performs slightly worse than STM, due to similar reasons as in the mono vision data set.

These findings show that memory networks are a good option to segment stereo streams as they provide high accuracy while requiring much less time than double that of segmenting one stream. This can be done in Eve but also in any other robot or application where segmenting both stereo streams has use cases.

6 Conclusion

The initial research objective was to develop a method for Eve to segment objects throughout a video in an unsupervised manner. After a literature search a more specific goal was arrived at: to design an architecture that allows for the modification of object selection in a memory network while still using previously made memories for predictions. For this, two different methods were devised. The focus was exclusively on combining memories, not on initialization or the exact implementation details in Eve. One method uses the feature space of keys and values to combine whereas the other combines in the probability space that the decoder outputs. These were implemented on top of the existing memory network STM.

Both methods are able to segment objects in cases where the original network fails. Out of the two approaches, the first one shows promising results on some videos but severely lacks consistency and fails to segment simple objects in others. This makes it unsuitable for practical use. The second method does succeed in providing high accuracy while showing one advantage of memory combination: addition when the original object is out of view. A larger memory bank resulting in more accurate segmentation is not seen in other cases but this may be due to the limited objects and situations in the made data sets.

Both methods suffer in speed when compared to the original network but it is difficult to quantify the exact extent of this. If the number of memory banks is not too large, this likely does not pose an issue for usage in Eve. Also, the implementation of the second method is not optimized for speed and there may be significant improvements possible in computation efficiency.

A secondary goal was to test whether memory networks can accurately segment both stereo vision streams, using a memory made from only one stream. The results of the stereo test indicate it is likely possible to do this without a major reduction in accuracy, but the limited size of the data set means it is hard to quantify the exact extent of this. These results align with the expectation that accurate predictions are possible as objects look very similar in both eyes.

Recommendations

First of all, a larger data set could provide more insight into the changes in accuracy the second implementation has. Especially videos of cases where good segmentation requires the larger total memory of the combining methods. When looking at larger data sets, calculating the boundary accuracy can also yield more insight as a combination of boundary accuracy and intersection over union is more telling than just one of both.

Implementation of the second proposed method in another baseline network can be beneficial as many such improvements upon STM exist, although it differs on a case by case basis how straightforward this implementation is. In, for example STCN [11], the network structure is very much alike, meaning the implementation could be similar as in STM while offering better speed and accuracy due to an improved key affinity calculation.

Directly evaluating the time it takes for certain computations in the combining of memory banks could yield more insight into the additional computing time a method takes. Comparing the frames per second as done in this work can give indications but a truly fair

comparison is hard in this way and measuring the exact time spent on combining operations would give a better indication of speed.

The stereo data set is also limited in size and can thus can not provide more than indications. In videos where the object is always visible in both eyes, switching the streams and averaging can remove the influence of some factors. Currently, some differences in IoU between the left and right eye can be caused by the object segmenting more accurately due to chance. If the same video is run through the network again, but now with memories made from the other eye, averaging across both ways provides a more robust evaluation.

Lastly, instead of using just one for input to the memory encoder, using both streams may improve the accuracy. Slightly different views of the object are not incorporated and this richer representation in the memory may allow for more precision during segmentation. Another option is to use the stream in which the object's previous mask was the largest. Then, if the object is only party visible in one stream, the other stream is used as input to the memory.

References

[1] Halodi Robotics. Eve homepage. `https://www.halodi.com/eve`, 2022. Accessed: 2023-01-20.

[2] Mingqi Gao, Feng Zheng, James JQ Yu, Caifeng Shan, Guiguang Ding, and Jungong Han. Deep learning for video object segmentation: a review. *Artificial Intelligence Review*, pages 1–75, 2022.

[3] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 1328–1338, 2019.

[4] Konstantin Sofiiuk, Ilia Petrov, Olga Barinova, and Anton Konushin. f-brs: Rethinking backpropagating refinement for interactive segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8623–8632, 2020.

[5] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5559–5568, 2021.

[6] Xi Chen, Zhiyan Zhao, Yilei Zhang, Manni Duan, Donglian Qi, and Hengshuang Zhao. Focalclick: Towards practical interactive image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1300–1309, 2022.

[7] Wenguan Wang, Tianfei Zhou, Fatih Porikli, David Crandall, and Luc Van Gool. A survey on deep learning technique for video segmentation. *arXiv preprint arXiv:2107.01153*, 2021.

[8] Seoung Wug Oh, Joon-Young Lee, Ning Xu, and Seon Joo Kim. Video object segmentation using space-time memory networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9226–9235, 2019.

[9] Hongje Seong, Junhyuk Hyun, and Euntai Kim. Kernelized memory network for video object segmentation. In *European Conference on Computer Vision*, pages 629–645. Springer, 2020.

[10] Haochen Wang, Xiaolong Jiang, Haibing Ren, Yao Hu, and Song Bai. Swiftnet: Real-time video object segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1296–1305, 2021.

[11] Ho Kei Cheng, Yu-Wing Tai, and Chi-Keung Tang. Rethinking space-time networks with improved memory coverage for efficient video object segmentation. *Advances in Neural Information Processing Systems*, 34:11781–11794, 2021.

[12] Ho Kei Cheng and Alexander G Schwing. Xmem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *European Conference on Computer Vision*, pages 640–658. Springer, 2022.

[13] Emanuela Haller and Marius Leordeanu. Unsupervised object segmentation in video by efficient selection of highly probable positive features. In *Proceedings of the IEEE international conference on computer vision*, pages 5085–5093, 2017.

[14] Xiankai Lu, Wenguan Wang, Chao Ma, Jianbing Shen, Ling Shao, and Fatih Porikli. See more, know more: Unsupervised video object segmentation with co-attention siamese networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3623–3632, 2019.

[15] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 37(5):1343–1359, 2021.

[16] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[17] Maximilian Durner, Wout Boerdijk, Martin Sundermeyer, Werner Friedl, Zoltán-Csaba Márton, and Rudolph Triebel. Unknown object segmentation from stereo images. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4823–4830. IEEE, 2021.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Computer Vision and Pattern Recognition*, 2016.