Bachelor Thesis

Video Object Segmentation with Deep Learning

Manuel Jüngst

Examiner and Advisor:
Prof. Dr. Michael Möller

University of Siegen

School of Science and Technology

Department of Electrical Engineering and Computer Science

Chair for Visual Scene Analysis

February 25th, 2019

Writing Period

2018-10-26 – 2019-02-26

Examiner

Prof. Dr. Michael Möller

Second Examiner

Hartmut Bauermeister

Advisers

Prof. Dr. Michael Möller

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Netphen, 2019-02-26 | M. Jengst

Place, Date | Signature

Abstract

This thesis primarily focuses on *One-Shot Video Object Segmentation* (OSVOS) by Caelles et al., that deals with the task of segmenting a foreground object from the background, over all frames of a video. Given a ground truth segmentation of the first frame of a video, a (fully convolutional) neural network trained for general object segmentation is then specialized for that video. The first half of this thesis mentions general video object segmentation approaches, including classical ones, followed by some deep learning approaches leading up to and including OSVOS. After understanding architectural considerations and training phases of OSVOS, and detailing how to measure the quality of such approaches, the second half covers all practical aspects. Using and extending the available code as needed, the results presented in the OSVOS paper can be reproduced, resulting in robust segmentation of desired objects in a wide variety of videos. Numerical experiments on the trade-off between computation time and quality indicate an amount of training iterations, past which the segmentation quality of OSVOS can be improved by a greater batch size, i.e. an optimization step is done over a batch of multiple transformed versions of the one-shot image. Visualizations of the internal representations of the network, and of the transition between the two training phases, indicate that both of these so-called parent and online trainings are important. Finally, the limits of the OSVOS framework are studied by creating particularly challenging video sequences, that reveal difficult to handle cases for one-shot learning approaches, such as drastic appearance changes of the object, or very dynamic backgrounds.

Contents

List of Figures

List of Tables

1 Introduction

Through advances in the field that is now known as deep learning, many impressive applications have surfaced in recent years; yet new innovations still continue to be released frequently. This is not merely due to the improvements in well established machine learning tasks, but also due to the increasing amount of digital data. Our lives continue to be digitalized and accumulated into usable datasets—with every Google search or with any public image upload. Deep learning thrives increasingly with bigger datasets, seemingly without a limit. Human level performance (or even greater) has been achieved in some [1], and is predicted to be achieved on increasingly more tasks—some even believe a significant percentage of jobs might be overtaken by AI [2].

Currently the process of accumulating data is often bottlenecked by low bandwidth input from humans—with two thumbs being a common input method (on a phone display). This reduces the size of datasets, that would otherwise have been available. One exception to this is image data, which is already being used widely in many deep learning applications. Even higher bandwidth is video data, which is the logical next step after mastering tasks on images.

At this time, working with video data is still an active area of research, with a large quantity of available video data representing a (yet) largely untapped potential for the fields of computer vision and deep learning. A big general goal for algorithms here is to generalize well from specific tasks to ranges of others.

For instance, some algorithms are designed to play games, such as DeepMind's AlphaStar learning to play Starcraft II [3], or OpenAI's Five learning to play Dota 2 [4]; the hopes for such algorithms are that they can generalize to applications outside of games [4, 5]. Similarly, advances on computer vision tasks may provide useful insights for deep learning in general. After all, computer vision tasks use convolutional neural networks, which are modeled after the human vision system, and the human brain is the prime example of a highly general neural network.

Again, this thesis is primarily focused on a paper titled *One-shot Video Object Segmentation* (OSVOS) [6]. The task there is to take *one* frame of a video sequence, together with a segmentation mask that covers an *object*, to then produce the segmentations for the rest of the video. The paper makes most of the used code available, and has also already been superseded by an extended version of the original algorithm (though this thesis remains focused on the original) [7]. While the results of OSVOS were produced with the source code using *Caffe*, they also made *TensorFlow* and *PyTorch* implementations available, and this thesis will use the latter for reference, unless stated otherwise. Further, the primarily used literature source here is the freely available *Deep Learning Book* by Goodfellow et al. [8].

Seen as a computer vision task, this paper provides some insights into deep learning in general. As a task working with video data, it contributes to unlocking the potential of available high bandwidth datasets. As an object segmentation task, it may be useful for image processing applications, such as a synthetic depth-of-field effect, as it can be found in smartphone camera applications [9]. Or for another image processing example, it could be used for tracking objects in robotics tasks.

Specific goals of the paper are to improve the state of the art for video object segmentation on three accounts. Firstly, by bringing in similar advances from boundary/edge detection tasks (to the segmentation task), which have already successfully been using pretrained networks (prior to OSVOS) [10, 11, 12, 13, 14]. Secondly, it shows that leaving out temporal consistency, traditionally a core aspect of working with video data [6], is a viable strategy and even improves the quality of the results in some cases. Thirdly, it presents an algorithm that has a freely adjustable speed/quality trade-off. All of these contributions are a push towards online applications, where a user actively waits for the results and possibly interacts with the application. These require acceptable quality, while also working with limited resources. In this case, even part of the training is done online, which represents an unusual challenge that typically not many machine learning algorithms have had to deal with. Both, an emphasis on speed, and a reduction on the amount of required training data, is something that any neural network algorithm could profit from.

1.1 Who Should Read This Thesis?

This work assumes a basic understanding of neural networks. Familiar concepts should be convolutional neural networks (CNNs), the basic challenge of (non-convex) optimization and using gradient descent to solve it, the relationship between number of parameters and training examples, and the role of training and validation/test datasets in their respective phases during the lifetime of a neural network. Terms such as ReLU, Pooling, Overfitting, SGD and Data Augmentation should be familiar in that context. Furthermore, a basic understanding of working with image and video data in general should be useful.

As such, this work may be useful to students aspiring to get one step further into the field of deep learning, or software engineers trying to gain a deeper understanding for a similar task. Hopefully, this will help the reader with the process of bridging the gap between the numerous available deep learning tutorials, and the highly specialized network architectures of today.

Some concepts that will be explained in here are: Fully Convolutional Networks (FCNs), using and adapting pretrained networks, deep supervision approaches, and methods for comparison of state-of-the-art algorithms for video segmentation.

2 Related Work

2.1 Classical Approaches

Video object segmentation has been around since before neural network approaches dominated computer vision tasks [15]. Typically, those rely on temporal consistency, the assumption that image contents of consecutive frames remain largely similar. Ideally, most background pixels stay the same, while (small) changes are only caused for object/camera motion and of course the always existing noise in natural images.

Many approaches utilize the temporal dimension by combining it together with the spatial dimensions, in form of a (spatio-temporal) graph structure [16, 17, 18, 19, 20]. This lends itself well to the segmentation task, as already existing graph cut algorithms can be used to solve it. The challenge there is the easily overwhelming complexity, arising due to the large number of possible vertices and edges in the graph. Overcoming this problem on the edge front typically requires having a sparse graph, limiting connections to a local neighborhood. The resulting limitation is that big jumps (e.g. caused by fast movement or occlusions) may put the corresponding object (or some of its parts) outside of the locally detectable range. To reduce the number of vertices, the problem can be overcome by using regions rather than pixels as the graph elements, also called pixel aggregation methods. These include superpixels [19, 20], which are small uniformly sized (though not shaped) regions that follow color/texture contours, or patch based methods [21, 22], which (coherently) split the scene along spatial and temporal dimensions. A method that produces overlapping regions is object proposals [18], where a possible trade-off could be region size against number of regions (though [18] in particular use a fully connected graph and a large number of proposals, with different optimizations instead).

One approach that uses these graph structures as well is bilateral spaces [16]. The name comes from the idea of bilateral filtering, which uses kernels for image processing that aim to avoid edges, e.g. a Gaussian blur kernel that preserves object outlines. For bilateral spaces, in this instance, the video data is cast into a higher dimensional space (such that it respects edges), and then snapped to a uniform grid structure—in order to make it traversable with computational efficiency.

Another traditionally important tool for working with video data is optical flow. It deals with the problem of mapping pixel or feature locations from one frame to another. If done for every pixel, this *dense* optical flow can be used to sometimes precisely determine boundaries of moving objects, based on their movement vectors. For this algorithm, many variations and extensions exist, since extensive research has been done on it over decades. For this reason, for a more abstract task such as object segmentation, an out-of-the-box algorithm is often used. It can then for instance be used to help establish temporal connections in a graph, or as a feature for region detection (within a frame) [19]. Detrimental to the use of such algorithms can either be their computational cost, or the otherwise limited quality.

No matter whether a classical or neural network approach is used, it is always an option to refine the segmentation results via post-processing. For instance, some detection artifacts or noise could be smoothed out. One such method is Conditional Random Fields (CRF) [23], a very much open method as far as possible configurations go. It is a statistical method based on user defined rules, called feature functions, which in this context typically model smoothness constraints or contextual relationships between classes. Each rule has a weight, and given an input pixel and some of its neighbors, the probability for it belonging to a certain class (e.g. background) is calculated. The effectiveness of a CRF depends very much on how it is used in a given context, though those with very general rules may not grant much improvement, while still adding computational cost (including the training of weights).

As mentioned in the beginning, the task of video object segmentation has been overtaken by deep learning, perhaps rendering much of the previous knowledge obsolete. Still, there is some value in knowing methods that have worked before, for instance to find creative ways in which to improve upon a basic CNN model.

2.2 Deep Learning Approaches

As a reminder, the fundamental difference between *classical machine learning* and neural network (or more accurately *representation learning*) approaches is the way in which features are learned [8]. The former requires to at least hand-design the features (if not the whole program), which may take decades of research [8]. Afterwards, the obtained algorithms are often domain-specific, meaning they do not generalize well to other fields. The latter is focused on having the machine learn the features itself (given enough training examples).

In our case, hand-designing the features basically entails catching the contours of objects, with the help of movements. This is usually a combination of a low-level approach attempting to catch similar colors/textures in an image (e.g. superpixels), with a high-level approach on top, attempting to connect object (or background) parts (based on motion). While this may produce good results in many cases, it is not difficult to come across cases where it breaks down. The problem is that such an algorithm has no (general) high-level concepts about what an object looks like. This lack of information is problematic on both levels: On the low level, for soft object contours, i.e. regions of similar color and texture between foreground and background—and on the high level, when trying to piece together object parts.

It is easy to conceive of such examples. Take a person with dark hair walking past a black speaker at head level, and at some point two similar regions will overlap. A possible explanation could be that this box belongs to the head of the person, and was perhaps revealed through a rotation, such that it was not visible before. How would you discern whether a newly appearing shape is part of a previously unseen side of that object, without a concept of what that (or any) object looks like? Relying heavily on movement to solve this causes issues on other fronts, e.g. for occlusions and fast movements. If the person were to walk behind the speaker instead, part of the body may be hidden from view. Once an object (part) has been lost, it is difficult to recover from that state [6]. Temporal consistency assumes only gradual changes in consecutive frames, but once the person is visible again, there is a jump between the last known and the newly visible location.

Neural network based approaches seem like the solution to this problem. Not only do they skip the need to hand-design features, they also promise to overcome this issue of classical approaches by being able to learn what objects look like. Looking a step ahead of video object segmentation, lies the task of semantic segmentation, which is to identify multiple object classes, rather than merely foreground and background. In that case, relying on temporal consistency to identify the foreground objects is not enough, and the concept of what certain objects look like has to be encoded in some form anyway. Caelles et al. list some concurrent examples using deep learning [6]: MaskTrack using Optical Flow and CRFs [23], and an instance of using the idea of bilateral filtering in form of a Bilateral Network [24]. This points towards some prevailing relevance of concepts from classical approaches.

Further, they list a visual tracking network, that deals with the related task of finding just the bounding boxes of objects in videos [25]. This, as many other deep learning algorithms working with videos, still makes use of temporal consistency, by largely using the network output of the previous frame for a new output. Interestingly, this approach has a network architecture which still has fully connected layers (due to the nature of the task—not requiring dense predictions). In contrast to that, current works on the video object segmentation task, or more generally tasks using CNNs for dense predictions (e.g. image outputs), deal with so called Fully Convolutional Networks (FCNs), which come with their own unique set of challenges.

2.2.1 FCNs and Pretrained Networks

As mentioned in Chapter 1, since deep learning tasks aim to be general, advancements for specific tasks may benefit the field in general, or at least a greater subsection of it. This holds for both directions, as image segmentation tasks have benefited from recent progress of CNNs as well [6]. Also mentioned was the surpassing of human-level performance—in particular on the ImageNet classification challenge [1]. This is at least in part due to having a very large dataset of 1.2 million training images in 1000 categories, which is then leveraged successfully by some of the top architectures. An obvious inspiration from that for the task here are the successful network architectures, or at least the design choices for some of their layers. Much of it goes even further though, also including their already trained weights [6]. For approaches based on such pretrained networks, this means new progress in the field can benefit them directly, by simply swapping out the underlying base network. If such an architecture does not fit the task, it may still be possible to modify it accordingly. A CNN can be adapted, e.g. as a FCN for the segmentation task, by removing the last layers that are fully connected. An FCN modification like this lends itself particularly well together with pretrained networks (which aim to simplify the training process), as the omission of fully connected layers removes a lot of trainable parameters, thus leading to a further simplified training. While there are better performing ImageNet architectures by now, a popular choice for use as a *pretrained* network remains VGG (from 2014) [26]. This may be due to the good trade-off it offers between architecture/training complexity and performance—as newer architectures are perhaps becoming deeper or more specific [27], or maybe they are just not yet tested enough. Pretrained VGG instances are used commonly outside of the segmentation task [10, 11, 12, 13, 14, 28], as well as inside [29, 30, 31, 32]; although custom architectures or other pretrained models are still used often as well.

One particular problem that arises when using FCNs in a context where potentially large objects inside images should be recognized (e.g. for segmentation) is resolution loss. It occurs due to the receptive field of the network increasing through the layers, which enables abstraction from low-level image features to the object level. In CNNs this is typically achieved through pooling operations or convolutions with a stride, which simply reduce the resolution—while the depth is usually also increased. When the segmentation output mask is desired to match the input image resolution, some form of upscaling has to be implemented when using this typical CNN structure. The issue is addressed by related work in different ways. Deconvolutional layers for upscaling with trained parameters are used in [31, 33]. Skip connections from early layers are used in [34]. OSVOS takes inspiration from some works that use side outputs, which help bring together early features that are accurately localized but abundant, and the coarse but object-corresponding features from later layers [12, 10, 35]. Lastly, besides architectural solutions, the localization accuracy of the predicted pixels could of course be further improved by post-processing, for which there are many possibilities.

3 One-Shot Deep Learning

The video object segmentation task aims to provide annotation masks for arbitrary videos, which can have any number of uses—from video editing to getting closer to understanding visual data. This is unlike the more established related tasks under the names of background/foreground subtraction/extraction, that rely on a priori information about the background (and camera motion) [18]. An important distinction for object segmentation is that it typically requires annotation (save for maybe a one-time calibration of the system for the other task family). Note that we refer to annotation only in the online context here, e.g. annotation may also be done to train a network for general segmentation, but this can be ignored from the perspective of an end user. For the related task of semantic segmentation, multiple objects may be segmented, thus requiring no online annotation. Alternatively, one could make an a priori assumption about the video data, that it is usually centered on just one object, also eliminating a strict need for annotation. But we would like to be able to follow a single object through unknown backgrounds and foregrounds in any case (e.g. with multiple similar objects), so some starting point for selecting that object would be a reasonable requirement. Also note that OSVOS evaluates multiple unsupervised segmentation methods on the primary dataset, some of which even beat semi-supervised methods (i.e. those using annotation); although the sequences indeed often contain one *moving* object that may stand out from the background. An annotation may come in form of a rough guidance, e.g. a pixel within the object, or a boundary area, but it may also be a complete mask containing every object pixel.

OSVOS takes on the latter case, which contributes to its ability to work well without the use of temporal information from the video. Lastly, the term *One-Shot* refers to the annotation being only given for a single video frame. Another option would be to annotate multiple frames, e.g. to refine the results as needed in a video editing application. This is also something that is easily possible with the non-temporal algorithm of OSVOS, as it has no dependencies between frames; though doing too much annotation would lead away from what the network tries to accomplish.

Having the goals established now, the following sections will cover the steps that the OSVOS approach takes in order achieve them. This includes describing how the architecture was modeled, the distinct stages of training, and the methodology for evaluation of the results. The next section deals with the adaptations to the selected pretrained network architecture, and how the localization accuracy issue is addressed.

3.1 Architecture

At the heart of the one-shot approach is the combination of a FCN architecture with a pretrained network instance. Firstly, the pretraining takes care of the object recognition for a large part, which would otherwise be impossible with just a single image of training data. Secondly, the fully connected layers hold a majority of the trainable parameters, so the lack thereof further reduces the amount of training needed. The desired result is that the network is able to adjust all of its parameters from a very general object recognition capability into a state where the activations strongly correspond to the appearance it has learned just by looking at one image of an object. This will require an additional training phase, which will be covered in the next section, but the architectural considerations remain the same.

The pretrained architecture that is used is VGG16 (where 16 refers to the number of layers with trainable parameters) [26]. Note that this is called configuration D in the paper, which was released alongside VGG19 (E)—both performing comparably well. The most straightforward modification to turn it into a FCN (with dense predictions) is to remove the fully connected layers. Included in the removal is also the last Pooling layer, which would otherwise reduce the resolution too much to give usable results [12]. What remains are five stages, each consisting of multiple Convolutional and ReLU layers, followed by a Max-pooling layer that halves the resolution for the next stage (if it exists). Although is somewhat arbitrary, the Pooling layer conceptually being at the end of a stage means that all layers within a stage work with the same image resolution.

An important building block for a FCN is a **Deconvolutional layer**, otherwise the network output image would be several times smaller than the input. More accurately called transposed convolution, this operation conceptually takes one input element at a time and applies it to each kernel element to produce multiple outputs. This is in contrast to convolution, that takes multiple input elements at a time and applies them to the kernel elements to produce a single output. Strides (greater than one) on a convolution reduce the resolution by decreasing the number of times the input is sampled, whereas for a transposed convolution they increase the resolution by decreasing the overlap between the outputs. The implementation for such a deconvolution can by done by simply swapping forward and backward pass of the convolution [30].

In the simplest case, a 1×1 Convolutional layer down to a single channel could be added to the model, followed by a Deconvolutional layer that upscales the resolution back to the original one. This example would produce an output of the desired format, i.e. a (single channel) mask with size of the input. But this model would not be ideal as it could e.g. only address the localization accuracy problem to a very limited degree (in the single Deconvolutional layer).

Inspired by *Deep Retinal Image Understanding* [36], which in turn seems to be inspired by *Holistically-Nested Edge Detection* [12], the next modification aims to improve localization accuracy by employing a multi-scale architecture. The idea is to combine information from different scales, where larger scales hold more accurate localization and smaller scales have a larger receptive field (i.e. greater object recognition capabilities). HED gives an overview over various such architecture configurations, ranging from simple skip connections to using multiple different networks. For their approach (as well as inspired works), a **side output** configuration has proven itself useful, which is also what DRIU and consequently OSVOS follow. The general idea of side outputs is to take intermediate outputs from the network, at locations where the scale is reduced, adding a (1×1) convolution that reduces the output to a single channel, and then usually also upscaling this output to match the input size. Those last two steps are exactly what the simple example from earlier did, just now applied to intermediate outputs. Finally, the outputs can be **fused** together into a single output (if they were all upscaled to the same size), e.g. by concatenating them before sending them into the final 1×1 Convolutional layer (in place of its previous input). This is exactly what HED does, with side outputs being after every stage.

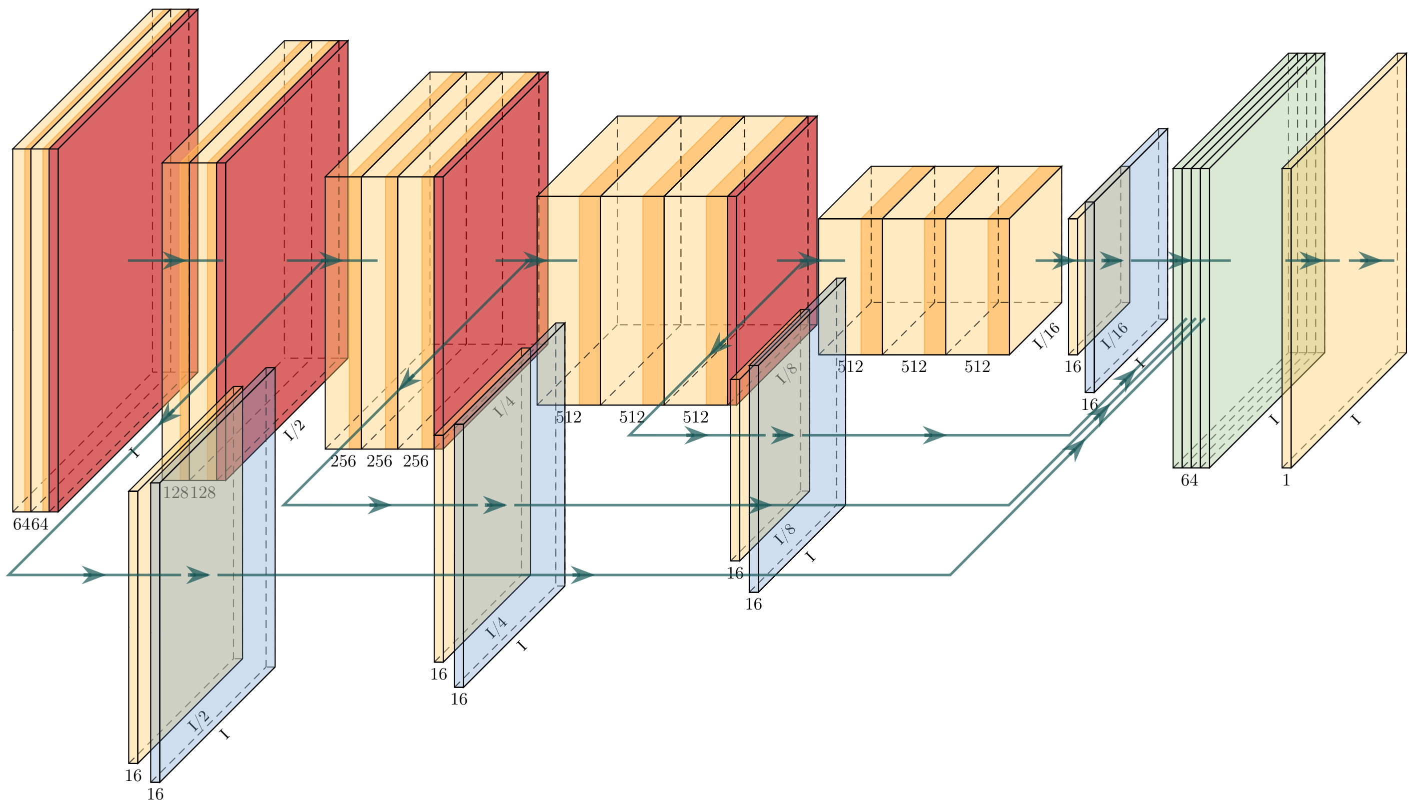


Figure 1. **OSVOS architecture for online training:** The 5 stages of the VGG16 network can be seen in the back, consisting of several Convolution (yellow), ReLU (orange) and Max-Pooling (red) layers. The flat side of a layer is labeled with the channel depth, and the slanted side with the image dimensions I; e.g. the second stage has two Convolutional layers of depth 128 and works with half of the original image resolution. The kernel sizes for convolutions are always 3×3, except for network outputs, where it is 1×1; for Max-Pooling, the kernel sizes are 2×2, and also using a stride of 2. The intermediate outputs of stages two to five lead into convolutions of depth 16, followed by a Deconvolutional layer (blue) that is parametrized such that it upscales the resolutions back to the original one (i.e. the first transposed convolution has a 4×4 kernel with stride 2, and both values double with each consecutive layer). For the primary output, the four intermediate outputs are concatenated (green) and (linearly) combined into single channel image output.

Source for drawing code: `https://github.com/HarisIqbal88/PlotNeuralNet`

Note that this side output architecture provides another opportunity: **deep supervision**. While this will be covered mainly in the training section, enabling deep supervision requires the side output layers to be just one channel, so that a loss function can be applied to them. The OSVOS architecture (which is the same as the one used for the optical disc segmentation task in DRIU) *extends* the single channel side outputs to a volume of 16 channels. Further, it drops the side output after the *first* stage, which seems to be a task specific optimization; e.g. the blood vessel detection task in DRIU drops the *last* side output instead, as the features become too coarse there. For OSVOS, it might as well be the case that the fine details of first stage side output obstruct the final result more than they help, as most object surfaces in the dataset are rather simple compared to a fine net of blood vessels. As the depth of the Convolutional and Deconvolutional layers increases to 16 (and the concatenation of their outputs becomes 64), deep supervision is no longer possible. Note however that since the depth-16 Convolutional layers are no longer immediate outputs, they are now free to do further processing, and get a $3{\times}3$ kernel size. This allows them to do some processing that is specific to their individual output scales. The architecture so far, which is already used 'as is' (in one case), can be seen in Figure 1. But for the final modification, in order to retain deep supervision, the intermediate outputs in the network are split again—this time after the depth-16 Convolutional layers, that now flow through another set of $1{\times}1$ Convolutional and Deconvolutional layers. The complete model, as can be seen in Figure 2, now consists of the main Convolutional path of VGG and a set of four side output volumes that lead into 1. a final convolution combining all side outputs and 2. another set of four 1-channel side outputs that can be used for deep supervision.

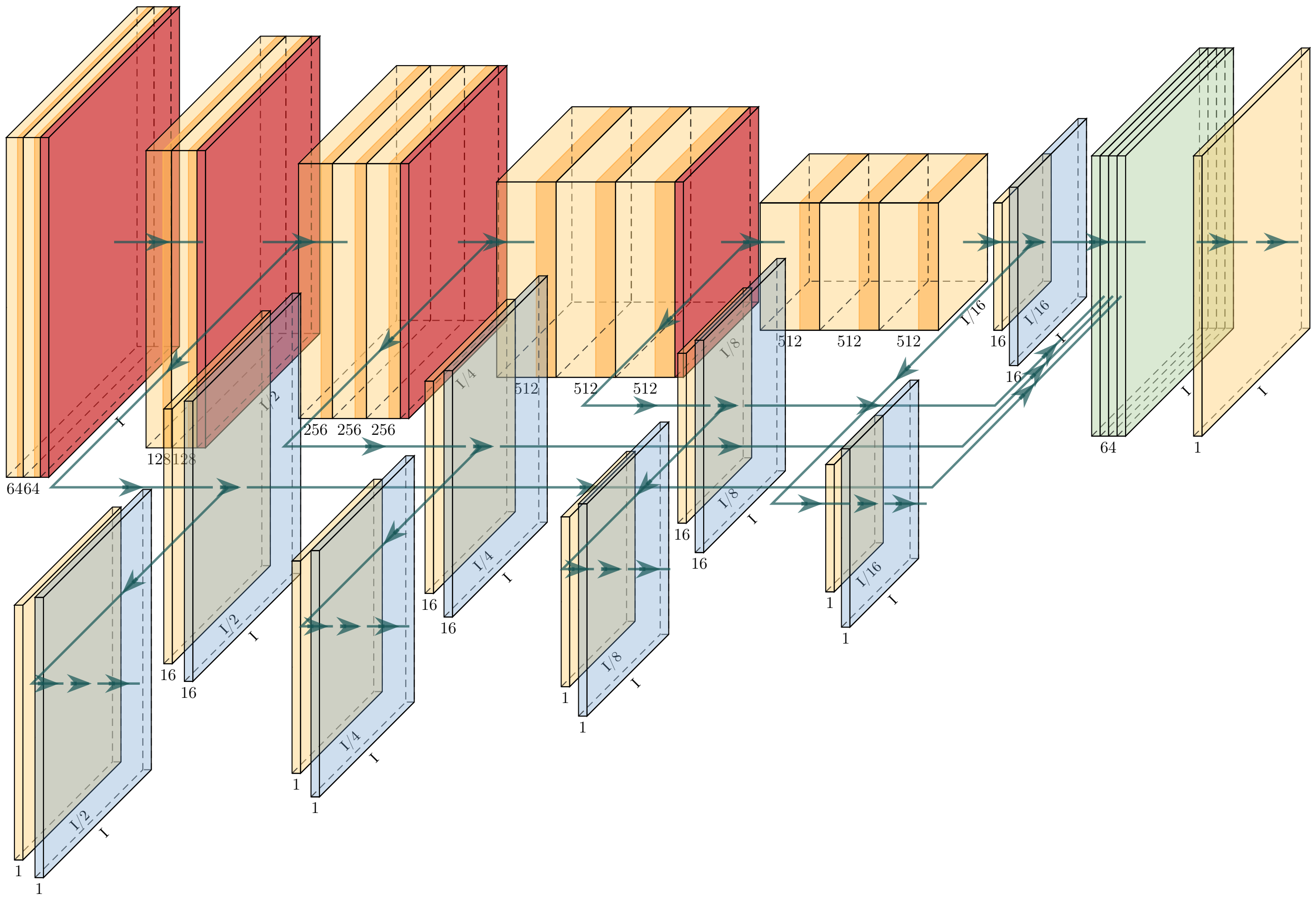


Figure 2. **OSVOS architecture for parent training:** To obtain the complete architecture, the model from Figure 1 is extended by another set of four Convolution/Deconvolution pairs. They respectively reduce the depth of the outputs to a single channel, and (as always) upscale the outputs to the full resolution. These are placed after the Convolutions of the intermediate outputs. In total, this architecture offers five outputs (marked by the terminal arrows) at different feature scales, and can be used for deep supervision, i.e. to guide what each stage learns, by applying a loss to each output.
This graphic is best viewed in the PDF version of this text.

While it is possible to use the model as is to obtain decent segmentation results, and despite having localization accuracy improvements, there is still room for some form of post-processing to (slightly) improve the results. The selected method for this generally does not depend on the main approach, and could be swapped for other methods—just like the pretrained architecture. The relevance for this to be in this section is twofold: Firstly, OSVOS tried a (fast) bilateral solver, which as mentioned in Section 2.1, is a method that detects naive edges (i.e. using only on colors/textures) in the input image. These presumably contain some of the edges from the objects to segment, and similar/close edges from the output mask could then be snapped to those, for some additional accuracy. This approach is relatively fast and also differentiable, meaning it could even be integrated into the network architecture. Still, using a naive edge detection is not as good as it could be, especially after already having created an architecture that effectively is able to recognize object edges. With an already established contour snapping procedure, the natural next step is to use another FCN to obtain such contours. OSVOS does exactly this—using another instance of the exact same architecture, but instead trained for contour detection. Their complete algorithm consists of these two instances, called foreground branch and **contour branch**, followed by a boundary snapping step. The next sections will cover how these instances are trained and what this boundary snapping step looks like; although the main focus of this thesis will remain on the foreground branch.

3.2 Training and Dataset

This section deals only with the foreground branch; training of the contour branch will be covered in the next section. So far, there has been a gap between using a pretrained network for image classification, and achieving an object mask as a dense output. The architecture is set up such that it is possible, and the dense outputs can be seen as also doing (binary) classification, but for each pixel. Further, the pretrained network has already learned how to recognize objects. Yet it has not learned to do a binary foreground/background classification, nor to keep activations in a spatially coherent form. Additionally, there are a few new layers which have not been trained at all yet. Without any segmentation training (one-shot training included), the network results would thus be completely random [6]. While the one-shot training will help adjust the network to the new task, the ideal spatially coherent form would be to just turn off all pixel classifiers that lie outside the given mask and vice versa. Even stopping early before overfitting like that, the network could not adjust to the general segmentation task as well as if it had more segmentation training data available. For this reason, OSVOS introduces another (pre)training stage between the **Base network**, that was trained for classification on ImageNet, and the **Test network** that is trained on a single image. To start with, we will look at what is necessary to (pre)train this intermediate network for general segmentation, which gets the name **Parent network**.

3.2.1 Datasets

With this new goal of another training phase in mind, a new question arises: What are the available datasets? The answer to this also ties into whether it is worth it to even go through the process of transfer learning (twice), which is to repurpose what the pretrained model has learned, for the new task. A range of possible datasets for video segmentation are listed in [37], but none of them were deemed suitable for video object segmentation. Firstly some of those are for different tasks, such as object tracking or semantic segmentation, although some semantic segmentation video sequences could possibly be adapted to (single) object segmentation with little effort. Secondly, some are too specific, e.g. being focused on hand segmentation. For cases where the data and annotation fit, the quality or quantity may not be good enough, e.g. the resolution is very low relative to current technology, or the amount of videos is insufficient to avoid overfitting. Even if a dataset exists that ticks all aforementioned boxes, it would also need to be challenging enough, such that the performance of algorithms on it has not yet saturated. The lack of modern standard datasets for the task presumably inspired the authors of [37] to create the DAVIS dataset. It contains 50 annotated and (presumably) challenging sequences for the video object segmentation task in 480p and 1080p resolutions, which represents an advancement as far as available datasets go; the most widely adopted dataset contained only six (low-resolution) sequences [37]. Shortly after the release of this dataset, OSVOS took the opportunity, making it their primary dataset for parent training. Still, this number of sequences pales in comparison to over a million images in ImageNet, which can be explained by the fact that dense annotation like this is significantly more challenging than for classification, or even object tracking.

The DAVIS dataset was designed to be challenging in several ways. Specifically, their paper [37] lists 15 attributes for videos that impede the object segmentation task, e.g. the object becomes occluded or is moving fast / changing appearance, or the background is cluttered/moving. The sequences in the dataset were then selected to contain multiple of these attributes. I encourage the reader to look at the full list in Table 1 of their paper. Seeing as these attributes may introduce a bias towards algorithms that do not make use of temporal consistency, an unfair advantage may be given to those. One could argue that the standard by which to measure/compare video segmentation algorithms should represent commonly found videos, which could be more favorable towards temporal consistency. There are other possible arguments, but even so, OSVOS still adds some justification for theirs, by also evaluating and comparing algorithms on *Youtube-Objects* as a second dataset. Note that an extended (more challenging) version of DAVIS also exists by now (*DAVIS 2017*), but in order to test reproducibility for the results of OSVOS, this thesis remains focused on the original version. Still, this thesis will at least do some testing on other video sequences, that will challenge reproducibility independently from any preexisting datasets.

The relevance for this dataset for the OSVOS algorithm is threefold. Obviously it provides the opportunity to adapt the VGG (and newly added) parameters into a parent network for general object segmentation (using training sequences). But seeing as it provides (rare) annotated data, it can also be used to take the first frame (from a validation/test sequence) for online training, and take the remaining annotated frames for evaluation of the network. The next three (sub)sections will cover these use cases of the dataset (parent training, online training, evaluation).

3.2.2 Parent Network

To answer the question from earlier on whether transfer learning is worth it: The tremendous gradient between the amount of training data from ImageNet and DAVIS provides at least some justification. What adds to this is the fact that later layers in neural networks become more task specific, and the last few of those were removed anyway from the VGG pretrained architecture. From the three reasons for which to use transfer learning in [38], a higher asymptote, i.e. a the test accuracy that can be achieved, seems to be the most important one for parent training. The increased accuracy is backed up by ablation studies in OSVOS, which will be covered later.

The first step of training for both networks (Parent/Test) is very similar. A loss function is required that compares the guessed mask of the network with the ground truth mask to determine a scalar value for how good the guess was. This creates a height dimension in the parameter-space landscape, which can be explored by computing the gradient of this landscape at a current position. Based on this gradient, an optimizer can then update the parameters of the network, i.e. moving into a (nearby) parameter position with lower cost/loss.

Let $\mathbf{W}$ be the trainable parameters, X the input image, and $y \in \{0, 1\}$ the ground truth label for a pixel ($y=1$ defines the mask and $y=0$ the background). A regular cross-entropy function, for two classes and a single prediction (i.e. assuming the image is only one pixel) is:

$$\mathcal{L} = -y \log P(y{=}1|X; \mathbf{W}) - (1-y) \log (1-P(y{=}1|X; \mathbf{W})) \tag{1}$$

This relates to the network architecture in the following way: The model takes an input X and computes the likelihood for a pixel to be part of the mask $P(y{=}1|X;\mathbf{W})$. This likelihood can be represented by adding a **sigmoid** activation function to the network output, that simply maps it into to a value between 0 and 1. It is functionally equivalent to do this with an additional Sigmoid layer or to compute it directly for the loss function; conceptually, the latter was chosen here. This is extended for multiple predictions y_j by doing this calculation for each pixel j of X and taking the sum. In the notation of Equation (1), the left term is nonzero when $y{=}1$ and the right term when $y{=}0$. Equivalently, we can omit the prefacing terms y and $(1{-}y)$ and make a case distinction instead. We omit $\mathbf{W}$ for readability and check the cases for ground truth pixels $Y_1 = \{\, j \in 1, \ldots, |X| \mid y_j{=}1 \,\}$ and Y_0 (where $y_j{=}0$):

$$\mathcal{L} = -\sum_{j \in Y_1} \log P(y_j{=}1|X) - \sum_{j \in Y_0} \log P(y_j{=}0|X) \tag{2}$$

The loss function that is used in OSVOS (going as far back as HED) is a class-balanced version of this cross-entropy loss. The notation of Equation (2) is useful, because the sizes $|Y_0|$ and $|Y_1|$ are used to determine the ratio β between the two classes, i.e. $\beta = |Y_1| \,/\, |Y|$:

$$\mathcal{L}_\beta = -\beta \sum_{j \in Y_1} \log P(y_j{=}1|X) - (1{-}\beta) \sum_{j \in Y_0} \log P(y_j{=}0|X) \tag{3}$$

Imbalanced classes for dense prediction tasks are common, especially for a task like contour detection (e.g. HED), where most pixels belong to the background. Decreasing a very imbalanced loss by means of learning what the input pixels represent requires lots of trial and error, whereas simply guessing the dominant class does so very quickly. In the worst case, the optimization could become stuck in a local minimum on (or near) such a constant prediction, preventing the network from ever reaching better results. Even if this is not the case, training may still take longer to reach convergence. In machine learning there are two common approaches to address class imbalance: sampling and cost-sensitive methods [39, 40]. Sampling methods are not (immediately) applicable to the segmentation task, where one cannot simply change the distributions of classes that the networks sees, as they are contained within single images—except for maybe feeding carefully cropped inputs into the network, such that the number of foreground/background pixels roughly balances out. Out of the four ways under cost-sensitive methods in [40], modification of the loss function yielded the best results. The second best way, adapting the learning rate based on the class, is again not really applicable for this task. The foreground class representation for the dataset at about 8% does not present an extreme imbalance (e.g. order of 1:100 or more) and is presumably addressed sufficiently by the simple modification by means of the β terms in the described loss function. Still, it may be possible to improve the results of OSVOS by employing a different loss function. Hashemi et al. mention some alternatives that outperform weighted cross-entropy loss in image segmentation tasks [41].

The loss from Equation (3) is used for both parent and online training. One difference for the parent training is that the architecture uses **deep supervision**, which adds an additional term to the base loss. As seen in Figure 2, we now have an additional set D of four side outputs for stages two to five. The same loss and ground truth is applied to each of those, but they are also weighted less as the training progresses, such that deep supervision guides mostly the early training. Let N be the number of training epochs and $n \in \{0, \ldots, N{-}1\}$ be the training progression:

$$\mathcal{L}_D = L_\beta(X) + \left(1 - \frac{n}{N}\right) \sum_{k \in D} L_\beta(D_k) \tag{4}$$

Deep supervision plays two roles in the training process: It acts like a regularizer in the sense that it improves test (but not training) accuracy, and it speeds up convergence [42]. Intuitively, it guides what each supervised stage of the network learns, which should result in a better representation of all feature scales; HED mentions that otherwise their network becomes biased towards large scale structures [12]. This can be explained by viewing the architecture up to each side output as a separate shallower network, where classification (and presumably segmentation) is forced to occur at lower depths already [42, 43]; thus feature discrimination is more likely to occur at all scales. Apart from training, deep supervision also provides a convenient insight to see how the final output is constructed. This is especially important when using an architecture that has multiple paths leading to the output. Ideally all of them contribute, achieving an overall greater accuracy. The issue when only looking at the final output is that it is not obvious if this is the case; e.g. when only one path contributes to a still remotely tolerable result.

With the losses defined for both networks, an **optimizer** can now be used to update the parameters. One important aspect that was not yet mentioned is what those parameters even are, because it is not the ones of all layers. For the Deconvolutional layers, the parameters are not trained, but set manually such that they compute a bilinear interpolation for upscaling. While it could be more common to have trainable parameters for Deconvolutions, HED found that this provided no noticeable improvements for their work [12]. For OSVOS, it plays nicely into the scheme of keeping the number of trainable parameters low, which is even more important there than in HED. The chosen optimizer for the remaining parameters was SGD with momentum 0.9, a base learning rate of 1e-8 and weight decay of 2e-4. Another aspect that was not mentioned yet is the scaling of the loss. Generally, network output and label are tensors of size $batch \times channels \times height \times width$ (in any order), where the loss value is the sum of losses for batch elements divided by the batch size. This is done because the learning rate is tied to the magnitude of the loss. While normalization of the loss is not necessary, as the learning rate can be chosen freely to fit any scale, it does make for easier comparison of optimization hyperparameters between algorithms. The loss here is treated in this fashion, i.e. *per-image* instead of per-batch, but the magnitude now still depends on the image dimensions. These are fixed to a certain size within the dataset, but as the algorithm generally accepts images of any size, it may make more sense to additionally divide the loss by width and height. Doing so would yield per-pixel loss with a corresponding learning rate in the order of 1e-3. The details for the optimizer, such as changes in learning rate and number of iterations, vary depending on the implementation (Caffe/TensorFlow/PyTorch), and will be covered briefly in Chapter 4.

With this setup, we could now train the parent network, using the DAVIS dataset as is. Although as available training data is limited, it is worth it to take a closer look at regularization. Weight decay is already used, which puts some limitation on the complexity of functions the model can represent. But the effectiveness of this is limited, since it encodes prior knowledge about the task that we do not have; we only know that very complex functions tend to overfit, where ‘very’ is somehow encoded by the weight decay parameter. Another regularization method that is widely used is **data augmentation**. At least when working with image data, there are very simple methods to generate more training data, namely: flipping, shifting, scaling, rotating. Among OSVOS and its successors (taken from the DAVIS 2017 benchmark[1]), most of them use at least two of these simple transformations to augment the training data [7, 44, 45, 46]. The choice of augmentations for OSVOS will be covered in Chapter 4 as well. More sophisticated means of data augmentation are also possible, e.g. *lucid data augmentation*, which is what the two remaining entries make use of [47, 48]. While improved performance of algorithms is generally a good thing, one has to be careful when comparing them, as one algorithm performing better than another could be simply due to using a better (i.e. augmented) dataset. Ideally, all comparisons between algorithms are done on the exact same dataset (including augmentations), so that improvements can be attributed to changes in algorithms, rather than having more/better data available. This presents somewhat of a problem in the comparison of the algorithms above, as all of them are using different kinds of augmentations. In particular it is worth mentioning that the best results on DAVIS were at one point held by the first segmentation algorithm to use *LucidTracker* [45], and the current best performing method on the DAVIS 2017 dataset is [48], which is using a variation of that same elaborate data augmentation

[1] `https://davischallenge.org/davis2017/soa_compare`. Last accessed: 2018-12-27

method. However, this does not necessarily invalidate those algorithms; there is not even a clear line between what is considered a pre-processing step or a part of the algorithm. Usually, the distinction is whether an operation is generally applicable, or specific to an application domain [8]; the image augmentations here are fairly general. Lastly, regardless of any augmentations, there is no objectively correct measure of what even constitutes a better algorithm (for the segmentation task), which is a topic that will be covered in Section 3.3.

In summary, we are now able to take our modified VGG architecture, train it with images from the (augmented) DAVIS dataset. We then apply a sigmoid function to main and side outputs to compute probabilities for pixels belonging to the mask; we apply a weighted cross-entropy loss to compute gradients, and use them in an SGD optimizer to update the trainable parameters of the network. This process is repeated multiple times over the whole dataset, and should result in a network that is able to segment general (and not specific) objects in a scene. If the training converges to an acceptable performance, which can be verified by observing that all side outputs are contributing to segmentation work (at least in most test sequences), we have a *Parent* network that is pretrained for the actual online training.

3.2.3 Test Network

The *Test* network, perhaps less ambiguously called *Online* network, has the task of taking a single mask of a sequence and continuing this segmentation for all remaining images. For the most part, the training here is identical to the *Parent* network; but still, some constraints here are very much different. To revisit the question on transfer learning for this task: It is most certainly worth it here (although arguably it is not much of a transfer, since the task is almost the same). The raw training data is only *one* element, but we now also have an additional **time constraint**. The term *online* implies live interaction with a user, e.g. some tool that could be used in a video editing application, where long processing times represent a large inconvenience. Running time is billed on a per-frame basis, and results should not only be qualitative, but also quick. At least without any of the two pretraining steps, the algorithm would have no chance of solving the problem.

As already mentioned, the loss function remains the same, but is not applied to the side outputs anymore. This was already the case in the last epoch of parent training, as the weight of deep supervision gradually decreases there. Deep supervision could be seen as training wheels, that are then no longer needed. It could also be argued that some degree of overfitting is desired here, in the sense that we no longer care about being able to identify/classify various foreground objects, but just one specific object instance. Still, it could be possible that deep supervision would provide some small benefit in online training as well.

The optimizer used here remains identical as well, even down to the specific hyperparameters. Given the additional time constraint and the number of optimizers, there likely exists a different one that gives at least a minor increase in speed of convergence. It may be the case that not many experiments have been done in this avenue, as qualitative results are more important initially. It also generally makes sense to keep both networks as identical as possible. Otherwise, even a simple change in the Test network (which is relatively quick to evaluate) would have to be justified to be different from the Parent network, that has a similar task (which takes significantly longer to train/evaluate).

The data augmentation for the Test network also remains identical. It is worth mentioning that this is a critical aspect of any semi-supervised (e.g. one-shot) algorithm. No amount of pretraining can prepare the algorithm such that this last training phase is not important. For instance, even if it were able to perfectly segment people in a given image/sequence, it may be given a scene with multiple people, then tasked to learn what one specific person looks like, even from unseen angles. Note that this exact task is tested multiple times in the DAVIS dataset. Data augmentation in itself is also very important when working with a single image. At the very least, it prevents the network from learning all the exact pixel locations, so that it still has to look at the color values. OSVOS is using a batch size greater than one, so that no optimization step can even lead into learning the precise pixel locations. Still, there may be some object regions that remain part of the mask through any of the applied transformations, so hard overfitting is always theoretically possible. This is where the elaborate data augmentation methods can really shine, e.g. creating new viewing angles for objects in a scene or rearranging the objects within. The latter is what the LucidTracker method does [49], although the former has, at least in the field of deep learning in general, been done before as well.

As mentioned in the beginning of this chapter, it would also be possible to improve a segmentation result by adding additional annotated data, i.e. another mask for a frame with particularly bad results. The network could then either continue training on this extended dataset, or restart the online training, depending on the time constraints. Experiments in OSVOS suggest that at least adding one additional annotation yields a small improvement (very roughly 5% on the score), and further frames yield only minor improvements (roughly 1% or less). So far, no definition of such a score has been given, which is required in order to discuss and compare results of segmentation algorithms; this will be the purpose of the next section.

Lastly, the time constraint on the online training means that the optimization does not typically run until the test error converges; rather, there is some earlier stopping point that yields a similarly close error, but in much less time. OSVOS gives qualitative and quantitative results for this, but only as a (per-frame) average in terms of overall computation time on their hardware, i.e. online training plus inference for all frames. To give at least some impression for timing in here: They present a segmentation after ten and 60 seconds of total online training (i.e. fine-tuning) time, and the latter yields a uniform and mostly covering mask, whereas the former has a few holes cut away (with one being fairly large).

3.3 Evaluation Methods

After taking the architecture through parent and online training, it is time to take the full test sequence (of which only the first frame was used for online training), and run it through the network. As within training, a sigmoid function can be applied to the output to obtain probabilities for the pixels to belong to the object. This can be saved as a grayscale image, although to get a per-pixel classification, one may want to threshold those probabilities (e.g. taking only $P(X) \geq 50\%$ as foreground). Regardless of whether the contour branch post-processing is used, this should now give a segmentation close to the results from the OSVOS paper. The next step would be to repeat this for all available test sequences in the DAVIS dataset.

Other than recognizing whether anything went critically wrong, we cannot do much with these results alone. For instance, if we were to change one hyperparameter of training (e.g. the number of iterations), how would those results compare? Or more generally, how does this algorithm compare to others? A starting point for a quantitative evaluation might be to use the loss value, averaged within a sequence, then over all sequences; but there are many issues with this. For one, there are many ways to (re)scale this loss value; secondly, not all segmentation algorithms even use the same loss function. In (binary) classification, it is possible to count the number of correct examples and divide it over the size of the dataset; this gives a value (accuracy) that is easy to interpret for humans, regardless of the underlying implementation. The same could be done for segmentation, where a single image already gives an accuracy percentage. Using this measure, one may find that it saturates quickly for images with mostly background (as is the case here); e.g. even if the prediction mismatches the annotation very visibly, the accuracy might be in the 99% range just because the most of the (large) background area is correct. Instead, it is common to remove correct background pixels (true negatives) from the equation. The resulting measure $\mathcal{J}$ is called **region similarity**, Jaccard index, or intersection over union, and represents the common vs. combined overlap of the two masks. The obvious way to get a score for the full test set is to compute this measure over every image and then taking the *mean* $\mathcal{M}$. While this score is somewhat informative by itself, it fails to capture some behaviors of algorithms that can occur. For instance, one algorithm may produce consistent results, while another does really well on some cases and poorly on others, yet they may average to the same score. A second statistic that may yield more insight on this is the *recall* $\mathcal{O}$, that represents the fraction of frames which score greater than a threshold (here 50%) on the given measurement. In addition, the DAVIS dataset/challenge offers a third statistic to apply a measurement over a sequence: The *decay* $\mathcal{D}$ divides a sequence into four parts, then looks at the score difference between the first and last quarter [37, 50]. This statistic can check if an algorithm is able to keep the initial segmentation quality up over time, which is especially relevant to algorithms using temporal consistency, as they may lose (part of) an object after e.g. an occlusion.

While region similarity roughly measures the number of correct pixels, a different question is how well they align to the contours of the mask. As an example, take an object with ambiguous edges, such as a shadow outline that follows an object closely; it would not add much error in terms of area, but could cause most of the outline to be misaligned. The measure in DAVIS for this is called **contour accuracy** $\mathcal{F}$. The contours of prediction and ground truth can be easily determined e.g. by removing all mask pixels that are not adjacent to the background. Intuitively, the measure then views each of the two contours as a set of points and checks the distances between all pairs of corresponding points. Ideally, each point in one set can be assigned to the closest point in the other set. Formally, this can be viewed as a bipartite minimal-cost graph matching problem [51]. Since a one-to-one matching is generally not possible, as one boundary may be larger than the other, one can find a matching first for all points in the *prediction* contours, then repeat the search but with the *ground truth* contours. This gives us two point sets containing the distances to nearest points. To turn this into a score, some *distance threshold* that defines whether points are matched may be chosen, which results in a precision and recall percentage for the two sets respectively; the threshold should depend on the size of the image for scale invariance. A common method to combine both values is the F-measure, which is their (harmonic) mean. Solving the graph problem, even with simplifications (such as making the graph sparse) and optimized algorithms, is still cost expensive. So at least in DAVIS this is approximated (with morphology operators) instead, i.e. by conceding the one-to-one matching and just checking if *any* point in the other set is close enough. This contour accuracy measure is just another way of scoring a single prediction mask, and the same three statistics from earlier can now be applied as well to get three more scores (for a sequence).

The two measures give complementary insights on the segmentation quality of individual frames. If a human were to evaluate a segmentation result intuitively, there is still at least one case where a quality could be perceived as bad despite both measures scoring high: The masks are (just slightly) incorrect in some areas, but the location of those areas changes over time; for instance, this could result in visible flickering or jittery boundaries. So lastly, DAVIS considers a third measure called **temporal stability** $\mathcal{T}$. The idea there is to take contour points again, though this time comparing *subsequent* frames of only the prediction sequence, to measure the distances between corresponding points again. But importantly, before finding the matching, a transformation/deformation is applied to the *whole* set of points, so as to align it most closely to the subsequent set of points. This is able to introduce a translation/scale/rotation invariance to the shape of the contours, such that large scale motion of the objects can be disregarded, while small scale oscillations of points remain unchanged, adding to the distance costs. Both, transformation and matching—which is again a minimal cost assignment problem—are part of the *Shape Context* algorithm, created for shape matching [52]. Briefly, a *Shape Context Descriptor* is a 2D histogram for one point of the shape, created from a polar coordinate overlay that partitions all other points into histogram bins based on angle and *log* distance from the center. The key for matching with a point in another shape (at about the same relative position) is that two histograms look similar if the shapes are similar. If one shape was e.g. rotated, the histograms would have a different x-offset (thus not being directly comparable), hence the transformation step of the algorithm to first align the shapes. Having obtained all matching points, the measure between two frames is then the average distance between all points. The only statistic that is used for this measure is the mean $\mathcal{M}$. Note however, DAVIS excludes sequences with occlusions and very strong deformations for this measure, as they would be falsely interpreted as instability.

With these measures, OSVOS and other algorithms can do a quantitative evaluation on the dataset. It allows us to e.g. tweak the hyperparameters of an algorithm, and see how small changes manifest in the overall performance. One thing to note is that after doing so, the test set should be called validation set instead, as it was used to improve the algorithm, even if only indirectly. For this reason, the extended DAVIS 2017 dataset/challenge adds an actual test set, where the annotations (except for the first frame) are not publicly available. It also introduces an additional primary measure/statistic score, that is the (arithmetic) mean of $\mathcal{J}$ and $\mathcal{F}$, which is then used to rank the algorithms. Having an overall measure for an algorithm is a good starting point, but it might also be worthwhile to try to optimize with a finer granularity. Instead of looking at performance on the whole validation set, one could look at per-sequence scores. For a semantic segmentation task (which is also what the annotations in DAVIS 2017 provide) one could look at per-class scores; for this task, one could also look at the scores per video attribute (as mentioned in Subsection 3.2.1, e.g. sequences with occlusion, appearance change, dynamic background, etc.).

A different perspective of granularity is granted by **ablation studies**, that evaluate performance of different parts of the algorithm (rather than different parts of the dataset). For instance, when an algorithm improves upon the state of the art by a significant margin, it is likely that it introduced multiple changes to the prior state of the art, so it would be interesting to find out how much each of the changes contributed to that result. In the case of OSVOS, performance is tested while leaving out boundary snapping, parent training, and online training in various combinations. Some insights from these studies are that both parent and online training contribute a significant amount to the scores (adding 15% and 27% to $\mathcal{J}_{\mathcal{M}}$), so if only one could be used, online training seems to be more important. The boundary snapping step adds a comparably small improvement to the scores; so another insight may be that the effort of having a second network plus post-processing is not worthwhile in an application with heavy time constraints. Expanding on the results of these studies, this thesis will also take a closer look at the effects of different levels of parent and online training in Chapter 5.

Being able to do many facets of evaluation now, the final question regards comparison on the state of the art; specifically, which works to select. Any works using the same dataset are obvious choices, and there may also be other works under the object segmentation task that use different datasets. But even broader, comparisons to only related tasks may be of interest. As mentioned in the beginning of the chapter, OSVOS also takes a look at unsupervised segmentation algorithms too; further, they even compare to object proposal methods. One would expect that unsupervised segmentation algorithms perform significantly worse, otherwise the labor intensive annotation process would not be worthwhile. The results show that they do perform worse on average, but not by much. This can be explained by looking at the dataset, that seems to largely meet the assumption that these methods usually make, which is that the objects to segment can be found through their motion. Generally, video sequences may either contain objects with clearly detectable motion *besides* the objects of interest, or instead not much of (obvious) movements at all. But in this case, these methods still make for a reasonable comparison. Meanwhile, the object proposal methods by themselves are likely not comparable at all, as they output many possible objects, of which at most a small fraction should align with the object. Still, the fact that (through many attempts) some *are* likely to be aligned with the objects means that at least the recall should be high. This is interesting because it answers the question what state-of-the-art algorithms could achieve (e.g. in terms of detecting boundaries) given perfect decision making, e.g. from a human operator. Results there show that OSVOS was not far behind this so called upper bound.

3.4 Contour Branch and Boundary Snapping

As already mentioned, boundary snapping is a post-processing step that aims to align the predictions of the primary FCN (foreground branch) to more precise edges in the image. These edges are best obtained by a second FCN instance (contour branch), which has the same architecture here. The task of contour detection being a whole new topic, and given its minor role in the OSVOS algorithm, not many details on it are provided (even taking into account the available source code). Briefly, the network is trained on a different dataset (PASCAL-Context), that provides contour annotations for full scenes (as opposed to for just one object) [53]. One reason for going this route, rather than training for just object contours, is to keep the costly online training time low, since this second network then no longer needs online training for a specific sequence. Even so, the added time of the whole post-processing algorithm competes with additional online training time of the foreground branch, and is only worth it once the online training is starting to saturate (e.g. at a per-frame processing time of over .5 seconds in OSVOS). The boundary snapping step requires a set of closed regions, i.e. superpixels, that are obtained from the contour branch output via the *Ultrametric Contour Map* algorithm [54]. It outputs hierarchical closed regions, i.e. larger regions are subdivided further by lines/arcs, where each such subdividing line/arc has a weight corresponding to the likelihood of it being a true (i.e. semantic) boundary. By thresholding this weight, the granularity of the regions can be adjusted. Finally, these regions are overlapped with the foreground branch segmentation, and those that are covered by more than 50% of the mask are added to the mask. Note that the experiments in this thesis will not use the contour snapping algorithm, as it ends up being somewhat outside of the scope of OSVOS, and is therefore also difficult to reproduce.

4 Implementation

4.1 PyTorch

As mentioned in Chapter 1, this work will build upon the OSVOS PyTorch implementation (rather than TensorFlow or the original Caffe implementation). Without prior experience, PyTorch seems to be the most modern (in terms of release date, as per first tag on GitHub) and easy to learn choice. Depending on the framework, the default parameters for training can be fairly different, and thus the final results may vary to some degree as well.

Batch size: For parent training, all implementations use a batch size of ten. Interestingly for online training, only the PyTorch implementation uses a batch size of five, while the others have a batch size of one. Since only a single image is given in online training, this makes only sense when taking into account data augmentation. The difference may be due to the fact that the PyTorch framework has integrated data augmentation, while (if not the frameworks) at least the other implementations are relying on external tools for this. Note however, that the available implementation of batch size here is atypical, by not actually feeding multiple images to the network in a single pass, but rather doing several single image passes and averaging the gradient for e.g. five iterations before doing an optimizer step. This allows processing of differently sized images and saves memory, however a classical batch size for better performance was implemented for this thesis. This is done by simply duplicating the one-shot image path in the *Dataset* class to match the desired batch size (for which the *DataLoader* automatically creates a *BatchSampler*), yielding a speedup of 1.7 over the averaged gradient method (for the batch size of five)

Training iterations: Note that the meaning of training iterations may not be obvious when taking into account the available PyTorch implementation. Generally, it could refer to *optimizer* iterations, i.e. how often the network parameters are updated, *image* iterations, i.e. how often images are loaded and modified, or *input* iterations, i.e. how often the network is fed with data. When using the average gradient method in place of the regular batch size, the input iterations are e.g. five times greater than the optimizer iterations. It may seem pointless to even consider input iterations (whereas the others are useful to judge the optimization progress or the computation time, respectively), but this (also known as the number of epochs in online training) is the parameter that the implementation uses. But any number of iterations from here on will refer to optimizer iterations.

Optimizer: The paper mentions a gradual reduction of the learning rate, which exists in form of a 3-step function, except for the PyTorch implementation. The learning rate there is fine-tuned per layer though, where the deep supervision Convolutions have a 10× and the final fusion Convolution has a 100× smaller learning rate. Additionally all the biases have twice the learning rate of the respective weights of the layers. For parent training, the default number of optimizer iterations is 50,000 (the same in Caffe, 95,000 in TensorFlow). For online training, the values obviously depend on the desired time/quality trade-off, but the default value is 2,000 iterations (or 500 in Caffe/TensorFlow).

Data augmentation: There are two fundamental approaches here: Generate a fixed number of augmented images beforehand, or generate them dynamically at runtime. The PyTorch implementation does the latter, with random flips, rotations in the range -30 to 30 degrees, and scaling in the range of 0.75 to 1.25. This does add a small cost at runtime, but has the advantage that most likely no exact same image is ever seen twice by the algorithm. The two other implementations follow the former approach, using only random flips and fixed scales of $\{0.5, 0.8, 1\}$.

Contour Branch: As already mentioned at the very end of Chapter 3, the contour branch will not be used. This mainly stems from the fact that it is not implemented in PyTorch (nor in TensorFlow). Even in the Caffe implementation, the contour branch is only available as a pretrained network instance, i.e. there are no instructions on how to train it. The contour snapping algorithm is also implemented only with a MATLAB interface (that also requires to build MATLAB executables from C++ code), which is otherwise not used in TensorFlow and PyTorch.

Changes: Other than the new batch size implementation, changes to the code were mostly for quality of life, e.g. small fixes, exception handling, support for multiple annotations, automation of multiple runs, saving of side outputs, dynamic device selection.

4.2 Datasets

Following the OSVOS paper, the primary dataset will be DAVIS 2016 as well. Additionally, a small test set of video sequences was created for this thesis, which will be detailed in Chapter 5. The raw footage was recorded with a smartphone in 1080p 60 FPS quality, but is sometimes additionally cropped (to a lower quality). The frames (for all but one sequence) were then extracted at 24 FPS, for simplicity and to match the DAVIS dataset (though this is irrelevant for the non-temporal algorithm here). The lengths of the sequences also matches those from DAVIS, i.e. they are up to three seconds long. Annotation is generally done just for the first frames, by me (i.e. not professionally, nor averaged from multiple people). One sequence is also fully annotated to allow qualitative evaluations.

4.3 Measures

Although the DAVIS dataset provides the source code for their evaluations (in Python), the $\mathcal{J}$ and $\mathcal{F}$ measures, with all three statistics, have been reimplemented here as well (in Java). For the measures, region similarity is very straightforward to implement, but the proposed bipartite graph matching for contour accuracy is also only approximated here. This approximation should be functionally equivalent to the one in DAVIS, where they dilate every mask pixel in a large disc shape (the size of the search threshold in pixels), then multiply this dilated mask with the other mask to check against; this leaves only pixels that are matched within the search radii with nonzero values. The implementation here simply takes foreground pixel locations of one mask and searches for matches within the other, in concentric rings in an order that finds the minimal distance. The distance threshold (search radius) here is set to 0.8% of the image diagonal, to remain consistent with DAVIS again. For the statistics, the implementation has been chosen to largely match the one from DAVIS as well. The mean takes into account all images, the recall removes the first and last frame from the sequences (the first because the ground truth is available for it, the last because some of the algorithms tested in DAVIS did not provide an output there). For the decay, there are various ways to divide a sequence into four parts (i.e. ways of handling cases of uneven quartile sizes); here, the first and last frames are discarded again, then for a sequence of length n, the first and last $\lfloor n/4 \rfloor$ frames are taken. This makes for an uneven distribution of the two inner 'quartiles', but those are not included in the calculation anyway; note that this differs from the DAVIS implementation (which seems to neither partition, nor have even length sequences). The temporal stability measure $\mathcal{T}$ is not implemented here, due to time constraints.

4.4 Sequence Viewer

During this thesis, a large amount of image data was accrued (around 500,000 images). Where applicable, quantitative evaluation is the most useful way to address this; however, some qualitative comparison is also necessary, especially for the deep supervision data. To deal with this case, a custom tool was developed to allow for direct comparison of sequences, i.e. the ability to switch back and forth between individual frames (without obstruction), which enables spotting even minute differences. Most conventional tools are able to do this only along one axis, i.e. all files in a folder, which corresponds to frames within a single sequence here. This *Sequence Viewer* however allows (mouse/keyboard) navigation along any number of axes, so long as all files share a common folder structure and filenames. In this case, navigation is possible along frames within a sequence, across different sequences (of the validation set), and across different result folders (e.g. comparing two full 20 sequence result sets with a different number of training iterations). Elements within an axis can be reordered and skipped, e.g. to ignore deep supervision outputs. The interface uses JavaFX, so the tool should run on all common operating systems.

5 Experiments

This chapter contains all the numerical experiments done with the OSVOS algorithm. Since the online network has to be trained for each sequence anew, it becomes a time expensive operation on the available hardware (that is bottlenecked by the CPU, most likely due to the implementation of the data augmentation and data loading in general). So usually experiments are done *initially* either with a smaller number of iterations, or on just a single validation sequence. But unless stated otherwise, the experiments mentioned in here are done over the full validation set of DAVIS, consisting of 20 sequences. As explained in Section 4.1, any number of iterations in here will refer to the amount of steps that the optimizer makes.

5.1 Reproducing OSVOS

The obligatory question is whether the results of the original paper can be reproduced. This is especially interesting when using the PyTorch implementation, that differs in a number of ways and has (expectedly) no published results. As mentioned in Chapter 4, the contour branch is not used, so a direct comparison can be done only on the quantitative measures from the ablation study without boundary snapping. Note that comparisons in this section will be done only with quality in mind, as direct timing comparisons would be on entirely different hardware, and numbers for online training iterations in OSVOS are not given. Table 1 shows quantitative evaluations on the DAVIS validation set with two network instances that were online trained for

	Measure comparison		Reproducibility		
Measure	Pre OSVOS[†]	Pre OSVOS[‡]	OSVOS -BS[†]	Online	Parent/online
$\mathcal{J}$ Mean ↑	79.8	79.8 *0.0*	77.4	**78.4** *1.0*	78.0 *0.6*
$\mathcal{J}$ Recall ↑	93.6	93.6 *0.0*	**91.0**	89.7 *1.3*	89.4 *1.6*
$\mathcal{J}$ Decay ↓	14.9	14.9 *0.0*	**17.4**	18.0 *0.6*	18.3 *0.9*
$\mathcal{F}$ Mean ↑	**80.6**	80.2 *0.4*	78.1	78.6 *0.5*	**78.7** *0.6*
$\mathcal{F}$ Recall ↑	92.6	**93.2** *0.6*	**92.0**	89.7 *2.3*	89.8 *2.2*
$\mathcal{F}$ Decay ↓	15.0	**14.9** *0.1*	**19.4**	20.9 *1.5*	20.5 *1.1*

†	Values taken from [6]
‡	Values taken from own measurements
Pre	Precomputed masks from [6]
-BS	Without boundary snapping

Table 1. **Comparison to OSVOS:** The big numbers show performance on the DAVIS validation set. The first two columns compare differences in the implementation of the measures/statistics here. The smaller italic numbers show the relative differences, where blue means a change for the worse, and red indicates a favorable change. Region similarity $\mathcal{J}$ itself is straightforward to implement—although there are some faint differences in the way the statistics are calculated, they are beyond single decimal place precision. Contour accuracy $\mathcal{F}$ on the other hand is implemented somewhat differently, which results in subtle differences. The second group of columns starts with the OSVOS scores without boundary snapping, and then shows the best attempts at reproducing those. Firstly, using the available pretrained *Parent network* (where only online training was done here), then with an instance that was (self)trained straight from the VGG16 base network. The role of the first column group is to give a sense of how the values of *OSVOS -BS* compare to the following columns, as they do not use the exact same measurements.

20,000 iterations. An entirely different perspective is whether similar results can be reproduced for different datasets as well; this will be covered in Section 5.5.

Online training only: Even though the measures differ by up to 0.6 percentage points, the comparison to the values from OSVOS is still valid, assuming the tendency for the $\mathcal{F}$ mean to score equal or worse and recall/decay to score equal or better holds for all network instances. The pretrained parent network happens to score exactly against the trends of the measure, meaning (under the assumption) it actually performs better in mean region similarity by one percentage point, and better by *some* amount in mean contour accuracy. For recall and decay of both measures it performs worse by up to 2.3 percentage points, which might be due to overfitting, i.e. training too long. It may be possible that some number of training iterations exists for which the overall statistics would improve. Still, when also taking into account random variance for network training results, it can be said that the results could be reproduced, despite the differences between the primary Caffe and secondary PyTorch implementation. The latter fact could even be responsible for potential improvements, due to e.g. better data augmentation and using mini-batches for online training.

Parent and online training: For complete reproducibility, it should also be possible to create the *Parent network* from only the well established VGG16 base network. As a first step, parent training was done over a range of iterations between roughly 37,000 and 92,000 (or 160 to 400 epochs with a batch size of 9, in increments of 40 epochs). For most of the statistics/measures, the instance with about 46,000 iterations performed best, which is also the closest number of iterations to the 50,000 given in OSVOS. Using this best performing parent network for online training, the scores in Table 1 are very close to that of the downloaded parent network, beating it slightly in contour score while trading off region similarity.

5.2 Time vs. Quality

One of the key features of the FCN approach is the relatively fast processing time for video segmentation. Especially when approaching the realm of real-time applications, an algorithm must not only yield good results, but also do so fast. This is obviously covered in OSVOS as well, however no direct numbers of online training iterations are given, as this is abstracted away to real time, and also only averaged per-frame. The availability of scores at known numbers of iterations would be generally useful for further analysis, e.g. to decouple performance gains of some methods from their specific (possibly not well optimized) implementations.

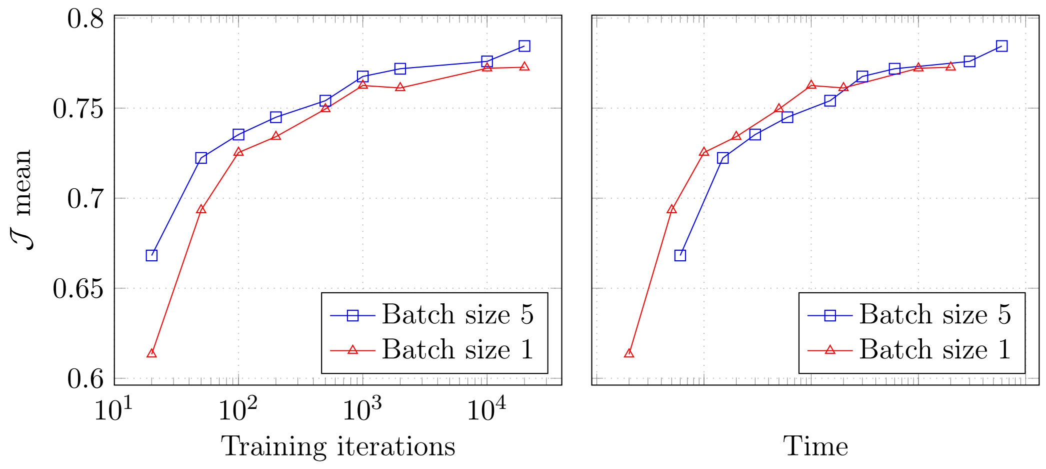


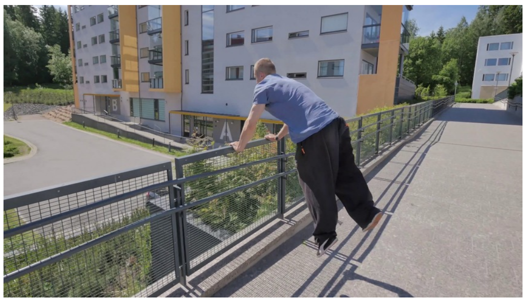
Figure 3. **Time vs. quality:** The left plot shows the region similarity score for various amounts of online training iterations, for two batch sizes. The number of optimizer steps is the same in both cases, but for a greater batch size, the number of images that are processed by the network is greater. As expected, a higher batch size yields equal or better results; but this does not factor in the increased computation time. The second plot takes the same data, but scales the iterations for the batch size of five by a factor of three, which has been very roughly experimentally determined. This factor depends on the hardware, especially the CPU in this case, so the results will vary for other systems. Still, there is generally some upper limit on the number of iterations / training time, below which a smaller batch size yields a better score, due to fitting a greater number of optimization steps into a time period.

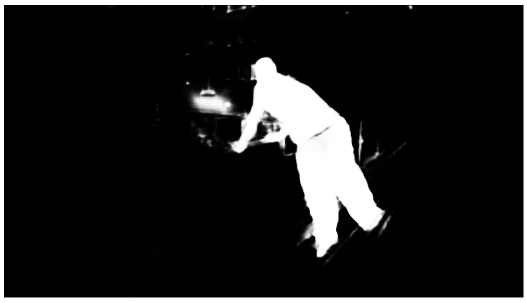
One such example is batch size, whose viability changes based on whether the default average gradient, or the faster classical batch size implementation is used. Figure 3 first compares the effect on performance of the default batch size of five against that of one, then (on the right) relates it to the actual computation time. The left plot shows that using a batch size of five for one-shot training is marginally better across all of the evaluated training durations. But when taking into account computation time, the greater batch size is at first not worthwhile to use, until somewhere between 1,000 and 10,000 training iterations, where a batch size of one no longer makes significant performance gains. Note that saturation for the batch size of five has not been observed yet, as training for 20,000 iterations over the validation set takes already roughly five days on the available hardware.

5.3 Deep Supervision

In Section 3.1 the role of the four side output stages was mentioned, which is for the earlier and higher resolution layers of the network to improve localization accuracy of the coarse final stage prediction. Deep supervision is no longer functionally used for the Test network, but may still be useful to gain some understanding about the algorithm. HED and *Convolutional Oriented Boundaries* [10] both use a multi-scale architecture with side outputs as well, and show what hierarchical outputs look like for the contour detection task. There, the early outputs contain many general contours (akin to simple edge detectors) while the later ones have fewer but more object-aligned contours. For the experiments, it would be interesting to see what this looks like for the segmentation task. It could also for instance be used to gain some understanding about the strengths and weaknesses of the algorithm. More concretely, it can also be used to verify that all parts of the network are successfully trained, which is non-obvious given only the combined end result; this will be used to validate some of the design choices of OSVOS, in Section 5.6.

Figure 4 shows the side outputs for a (late) frame of the *parkour* sequence from DAVIS, after 10,000 training iterations. All of the test sequences from DAVIS follow a similar pattern as seen there, given enough training. This reaffirms that hierarchical features can be learned for the segmentation task as well, which is an important step in approaching human abilities for edge/boundary recognition [12]. It may be interesting to note that the internal representation, as can be seen through the side outputs, changes at a faster rate than the main output. In other words, even when the main output remains relatively constant, the side outputs can still shift around visibly. As online training begins to converge, especially the last side output typically still goes through non-distinct gray level activations and converges towards a more binary segmentation result. This property could potentially be used to dynamically determine a good stopping point for online training, i.e. one that may be tailored to the optimization difficulty of a particular sequence. Looking at the side outputs, we


(a) Input image


(b) Main output

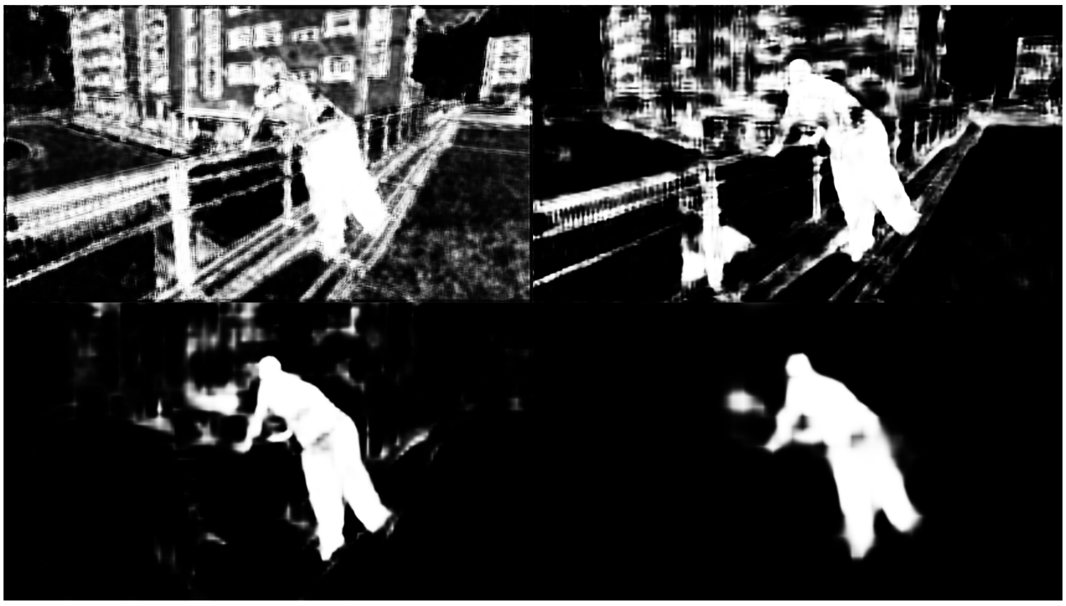

(c) Deep supervision outputs

Figure 4. **Deep supervision:** Given input image (a), the network produces (b) as an output, which is a combination of four intermediate outputs (c) (stages 2 to 5, in order top left, top right, . . .) of the network. Note that the actual intermediate outputs that are combined to (b) are 16 channel 'images' each, and shown are only compressed single channel representations. Throughout the network increasingly more features are discarded, e.g. trees and asphalt early on, then building structures, while the resolution is simultaneously reduced (seen as increasingly blurry images). As a result of being able to incorporate information from higher resolution stages, the final output contains much finer details than the last stage (c) bottom right provides. In this particular instance, each stage plays a significant part in the segmentation process, i.e. each one removes increasingly more background clutter.

can also see what an optimal representation (in terms of our loss function) looks like: Background clutter is segmented out gradually—too early and information would be discarded before higher level features can be recognized, too late and the final result would still contain background clutter. From this, it can be reasoned that adding more stages to the architecture may give more time for the network to make these segmentation decisions and thus yield better results; this can be seen as further justification or understanding for doing so, since the benefits of increasing network depth are widely known already.

5.4 Parent vs. Online Training

The overlap of parent and online training leads to several questions: Is online training complementary to parent training, or does it rather override much of the effects of the prior parent training? Which of the two plays the major role in achieving good segmentation results? Answering these questions is an important component in understand the algorithm, and gives a good starting point towards achieving improvements. OSVOS lists the mean region similarity scores for their (foreground branch) model with only parent and only online training respectively, but this only goes as far as proving that both contribute in some way towards the final result.

It is not clear how long these two networks were trained for, and regardless, it may be interesting to see what kind of features are developed over time.

So Figure 5 shows the performance of various amounts of training iterations, which validates the findings of OSVOS: Without parent training, the maximum $\mathcal{J}$ mean score is slightly above 60% here (compared to 64.6% in OSVOS); without online training it is at about 50% (compared to 52.5% in OSVOS). It also shows that parent training is likely more difficult to optimize, as the score difference between the measurement steps has more variance than for online training, and it also takes a while until an upwards trend is even noticeable.

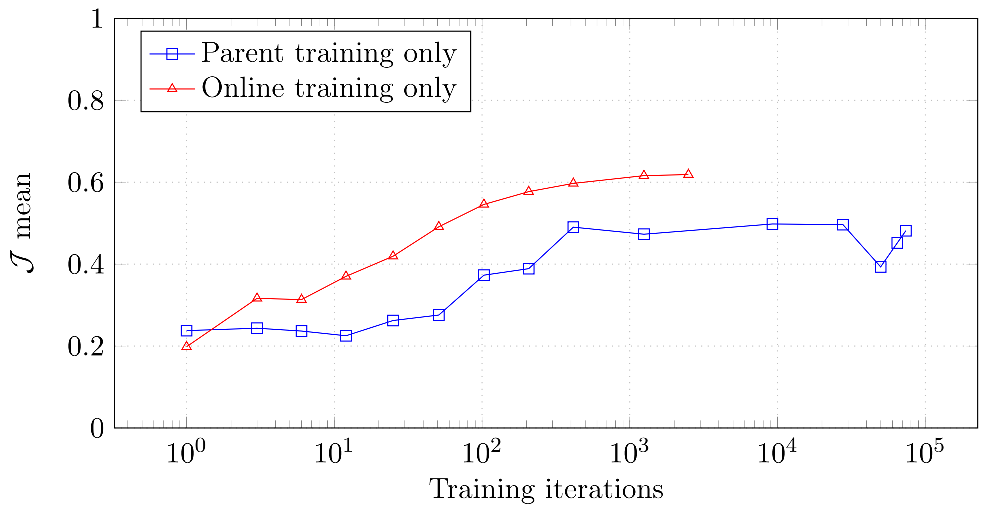


Figure 5. **Quantitative results of parent vs. online training:** Shown is the contour accuracy mean score for the DAVIS validation set. The red curve (triangle markers) was obtained by online training the OSVOS architecture initialized from only the VGG weights, i.e. parent training was skipped. The blue curve (square markers) was obtained by doing only parent training on top of the VGG initialization, i.e. online training was skipped. If only one of parent or online training could be chosen, the latter would be the better choice to maximize the score; it converges faster, more consistently, and to a higher maximum. Note that all online training results were obtained using the pretrained *Parent network* instance (from the OSVOS PyTorch repository), whereas all but one of the parent training results were obtained with a self-trained parent network. The one exception is the third to last blue measurement (at 50,000 iterations), which also used the pretrained instance.

Perhaps more interesting is a look at the segmentation results of the two series of network instances. Figure 6 gives some insight into the learned features there: Only parent training returns multiple object activations, as expected. One additional issue (that is not shown in the figure) is that some objects have blurry outlines. This means that the parent networks are able to identify the presence of foreground objects, but not their extent. More surprising are the outputs from only online training. The scores from OSVOS and Figure 5 suggest that online training plays the more important role leading to good object segmentation. But in contrast to that, a human subjective evaluation may actually come to the opposite conclusion: After parent training, most objects are part of the foreground mask and mostly intact, while after

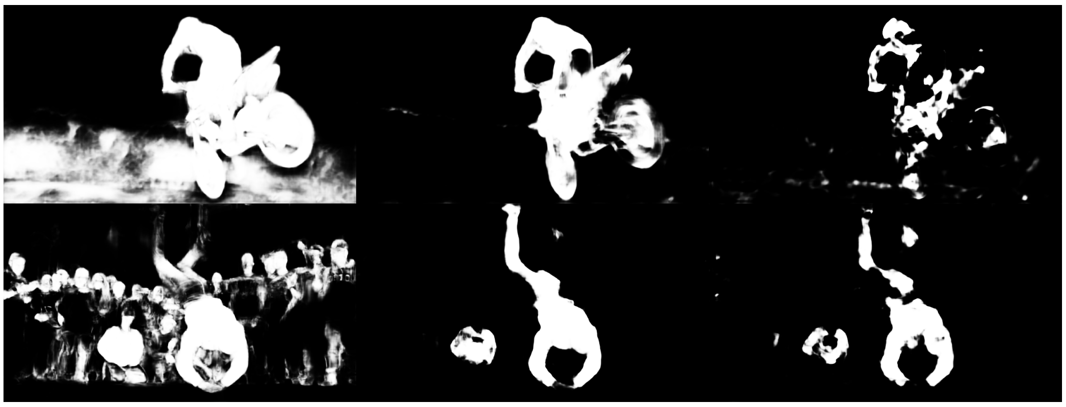

(a) Parent training (b) Both (c) Online training

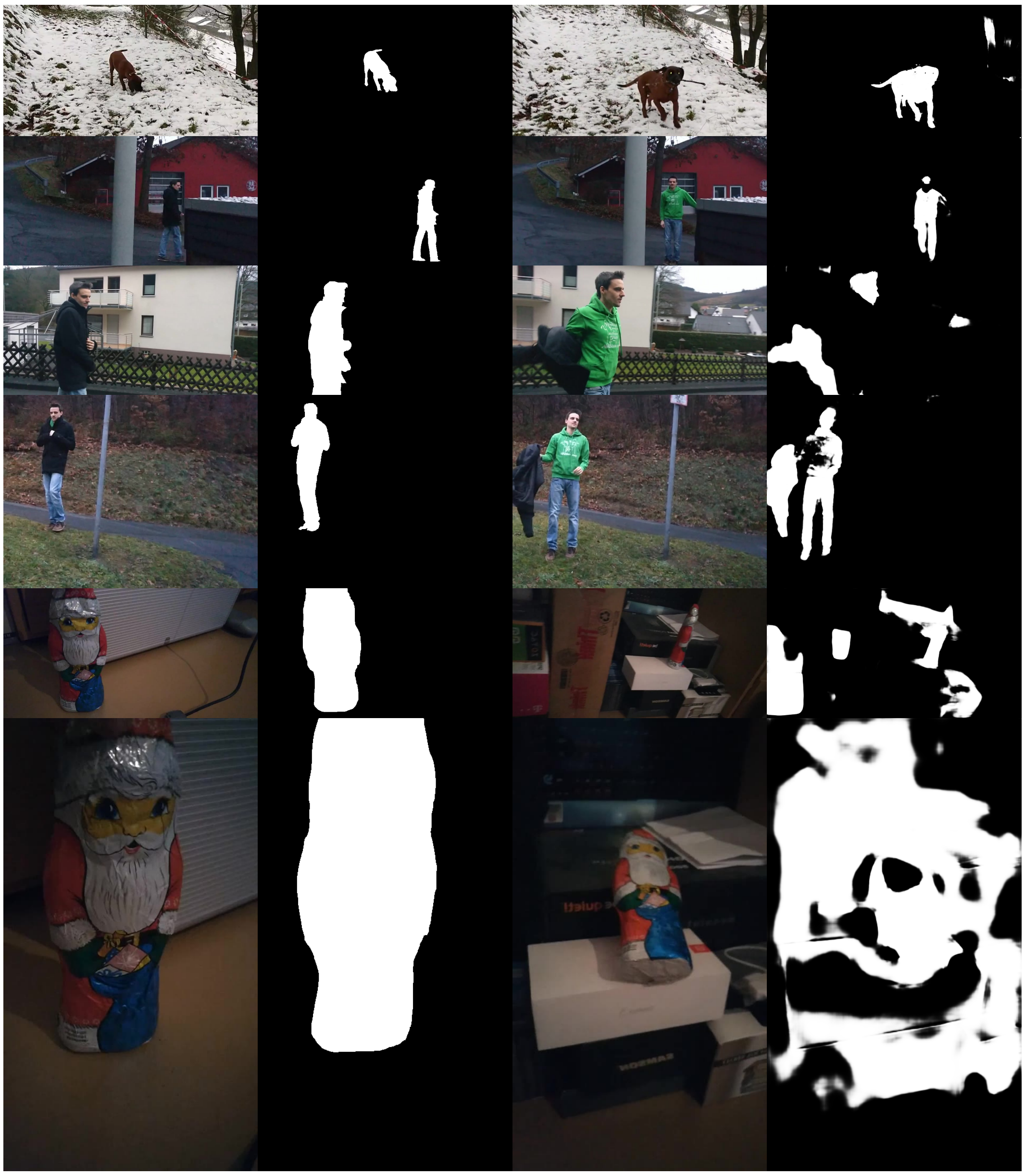
Figure 6. **Comparison of different types of training:** Shown are segmentation results for three differently trained networks. The two rows show a frame from the *motocross-jump* and *breakdance* validation sequences respectively. Column (a) was obtained using a network that only used parent training, column (c) was obtained using only online training, and column (b) shows a regularly trained network that uses both. Without online training, the network can only do general object segmentation, which is why the mask contains multiple objects, such the people and the forest background seen here. Without parent training, the network may not learn general segmentation at all. So when the relative viewing angle to the object changes during the sequence, it fails to recognize previously unseen parts of the object, i.e. the mask may contain many holes. Note that this is not simply due to overfitting, as (c) was obtained from a network that was trained just long enough to yield a relatively binary segmentation result (about 1000 iterations); training any less only yields a more blurry/grayish result. Combining both types of training, most of the unrelated background objects are successfully removed, but simultaneously some of the hole errors from online training are introduced along with it. Note how (a) was able to correctly assign the entire motorbike to the foreground, and after additional online training (b), this ability is lost; the same effect can be observed on the left leg of the dancer.

only online training, over half of the sequences show major gaps in the foreground prediction—with some objects becoming completely unrecognizable after only a few frames. In other words, such a human subjective score may give additional weight to smoother and more connected results—but this would of course be difficult to quantify. However, for some applications, the results of online training may still be preferable, e.g. for fully automated processing. But either way, this shows that online training may not have to do a lot of work, particularly on low level features. It also shows that just online training is not enough for the network to learn general object segmentation, which is an important ability for sequences where the object changes appearance, e.g. by facing a different direction relative to the camera. This can be further supported by looking at the side outputs after only online training, that also lack the typical structure of (human-like) hierarchical segmentations, with mostly binary activations; here, they have exclusively gray-level activations throughout all stages, with no discernible hierarchy.

To lastly answer the question on whether online training overrides parent training or is complementary to it, it seems like the latter case holds most of the weight. The visual similarity after both trainings in the two examples in Figure 6 is closer to that of only online training, but for most sequences it is actually closer to the result of only parent training. In particular, the specific blurriness around objects that is only introduced by parent training, can still be commonly found, and also the very degenerate cases of only online training have very little similarity to the respective cases with both trainings. Also for at least some sequences, convergence towards a mostly binary segmentation result is much faster when parent training precedes, which would not be the case if online training were to override most of parent training.

5.5 Custom Sequences

As already mentioned in Section 5.1, a different perspective on reproducibility of OSVOS is not just getting the expected results on the same dataset, but also testing how well it generalizes to arbitrary video sequences. It is also essential to be able to take in common video formats and process them so that they can be segmented by the network, including the additional (one-shot) annotation. As such, multiple different video sequences have been created from scratch, encompassing the entire workflow from filming to editing, annotating and evaluating. Evaluation is done mostly qualitatively, as computing the scores requires full annotation of all frames, which is prohibitively time consuming. The scenes of the sequences have been chosen to probe various strengths and weaknesses of the algorithm, especially with expected failure cases in mind, so as to test the limitations. Although innately practical, the quality of the videos has also been chosen to be purposely low; at times, they feature motion blur (due to fast object movement), low resolution, compression artifacts, bad lighting (i.e. noise), and shaky camera movement. This should reflect the type of videos that are widely available through the use of smartphone cameras. Further, the format of the sequences varies in resolution and aspect ratio, which is not the case with the uniformly sized DAVIS sequences. Although the FCN architecture should be able to process these, the segmentation quality could possibly still be negatively affected by it, given that all parent training is done in one particular size / aspect ratio.



(a) First frame (b) Ground truth (c) Later frame (d) Network output

Figure 7. **Custom validation sequences:** Each row summarizes a sequence that was created for this thesis and its segmentation result. Columns (a) and (b) are the first frame of a sequence, and its manually annotated mask. Columns (c) and (d) show a later frame from the same sequence, and the segmentation mask that the network produced; these have been selected such that they are representative, i.e. (c) shows a keyframe of the scene, while (d) shows typical segmentation quality. Note the appearance change of the foreground object between the two time steps, in rows two to four, and the background changes in rows three, five and six.

An overview over the created sequences shown in Figure 7. Note that the definition of an object has been chosen to presumably match that of most human annotators. In particular, where one object is an actor that holds or wears an item, this is considered to be part of that object, even if this was not positively specified in the ground truth frame (the exception being if the item was visible and marked as background in the ground truth). However, this will always be somewhat subjective, e.g. for what constitutes *holding* an object, or at what point something is considered *visible*.

Buddy: This sequence represents a simple initial test run, and is expected to work fairly well. It features a brown dog, over a mostly white background (with some brown/green spots). The segmentation works well, with a few false positives on parts of some trees (for a minority of the frames) and a few grass spots, as well as minor false negatives on the pupils (due to eyeshine). The stick that was picked up is not detected, although it was also not segmented in the ground truth, where it is barely visible.

Behind-wall: Here, the foreground mask is an actor that undergoes a strong appearance change, by replacing a black jacket with a green hoodie; this change is also hidden behind a wall. Additionally the background displays more color/texture variation, although it remains mostly fixed throughout the scene. The segmentation result works surprisingly well, with most of the body still being correctly segmented as it becomes visible again. Being visible from the front rather than the side, the feet are missing, as well as the neck (which was also concealed in the first frame). Note however, that half of the jacket is still visible behind the wall, which by the earlier definition is still considered foreground as well.

Jacket-walk: This sequence features the same appearance change, which is this time visible in the scene as well. Increasing the difficulty further, the background changes significantly and reveals multiple new surfaces that are unknown to the algorithm. Additionally, the background is more similar to the foreground here: The black jacket overlaps with a dark fence, the green hoodie overlaps with green lawn, and brown hair overlaps with a brown tree/forest. Figure 7 shows a frame where the jacket is being removed, where the underlying hoodie is then no longer predicted as being part of the object. The jacket itself and the jeans are correctly detected for the most part, except for the areas in close proximity to the hoodie, also including the neck again. Even more surprising is the amount of false positives that occur towards the end of the sequence, but this will be elaborated on later. All the of following sequences are at least partially aimed at figuring out the conditions that lead to this issue. More results are shown in Figure 8.

Jacket-easy: To isolate the item holding and dynamic background challenges, this sequence focuses on the former. As *behind-wall* showed, it is possible for the network to overcome the appearance change, but it is not clear under which conditions this holds. Here, the scene background is intentionally simple, and also a hint of the green hoodie is visible initially. As a result, the segmentation works even better than in *behind-wall*. The jacket is correctly segmented all throughout the sequence, almost all of the body is correct, and no false positives are found. Only when the outline of the jacket stops overlapping with the rest of the body, the upper body is no longer classified entirely correct. Still, the foreground detection from the known body parts, namely face, hands and legs, appears to carry along some form of intuition that there should be a surface that connects them in between. This result suggests that knowledge about the background is a factor of correct classification decisions, e.g. when all of it is known, then the class of unknown surfaces can possibly be inferred from known foreground object parts.

Santa: Complementary to the prior experiment, this one focuses on dynamic background, while the foreground object remains fixed in color and shape. At the same time, the second idea here is to test a foreground object that is much different from the ones of the DAVIS dataset, that contains both dogs and humans already. This way, the parent training of the network should be less meaningful, which may amplify some issues. The concrete question is how much of a problem the unknown background is, when the object does not undergo strong appearance changes like in *jacket-walk*. The results here give no clear answer to this; while the network also falsely predicts multiple background objects, they are often visually similar to the red and white Santa. As seen in Figure 7, the box on the left is reddish (pink) and white, the box on the right is mostly white, and the text on the tall brown box is red. Further, the green box on the left matches Santa's green gloves, while the blue box in the far back somewhat matches his blue sack. With the absence of any of these colors in the background of the first frame, this seems like a plausible explanation. This is unlike *jacket-walk*, where e.g. roofs, road, shrubs and a car are false positives, with no apparent color similarity to the foreground mask. Note how the (falsely) predicted red text has its outline corrected to be a simple—almost rectangular—shape, suggesting that the network expects certain shapes (that are similar to the learned mask), which would be a human-like quality.

Santa-vertical: This final sequence is very similar to the one above; the difference is that the object is now much closer to the camera initially and it moves as well. While it obviously tests vertical video, the sequence also increases the difficulty on the unknown background challenge, as even less of the background is taught. The results reflect this increased difficulty very well; much like *jacket-walk*, we now get false positives on boxes that are not so similar in color anymore. The outlines of the detections are also no longer aligned well to the actual objects in the scene. Also new is that Santa itself is no longer detected entirely as foreground. The interpretation of the results is difficult due to the many alterations in the sequence. The detection of Santa is worse, likely because the shape changes more than before (i.e. a stronger rotation and the size in the image plane becomes much smaller). A link between the false positives and false negatives cannot be established from this; the misclassification of the black boxes now could just as well be due to the fact that there is no longer a black background object (e.g. the black cable) in the first frame. The vertical aspect ratio does not seem to be the cause, as processing the same video rotated by 90 degrees yields fairly similar results.

In summary, the network deals very well with low quality video and different formats, and can also handle even strong appearance changes to some degree. A significant issue seems to be background changes; e.g. for a video that is much longer than the three second clips tested here, where the camera moves a greater distance, the algorithm would likely not have a high recall $\mathcal{O}$ (i.e. many frames with an acceptable score). Another observation is that changes in the hue of a color seem to have more weight than changes in brightness, e.g. a dark red object would be less likely classified as a black object, than a light gray object would. This makes sense, as changes in illumination of the scene (including reflections) are likely to cause the latter, but not the former. For best results, the first frame should probably be selected such that it contains a good variety of background objects, especially if they are similar in color to the foreground object. It may seem odd that unknown objects in the background are classified seemingly random as foreground (rather than being defaulted to background); but if this was not the case, new angles or general changes of the foreground object would also not be recognized. A better strategy would be to focus the search for object parts either near the previously predicted location (i.e. using temporal consistency), or to enforce some kind of consistency regarding the foreground composition, e.g. preferring a single connected object in the mask (i.e. using a more sophisticated loss).

5.5.1 Further Annotation

Next to these two general approaches to improve the results, a more direct approach is to add more data to online training, as already suggested in OSVOS. In order to effectively test this and other potential improvements on one of the custom sequences, a full annotation would be very much desired. Being the most versatile sequence, *jacket-walk* has been chosen; all 64 frames of the sequence have been manually annotated. Note however, that this has been done with some assistance of the algorithm; all of the outlines have been manually thresholded and hand-corrected, but

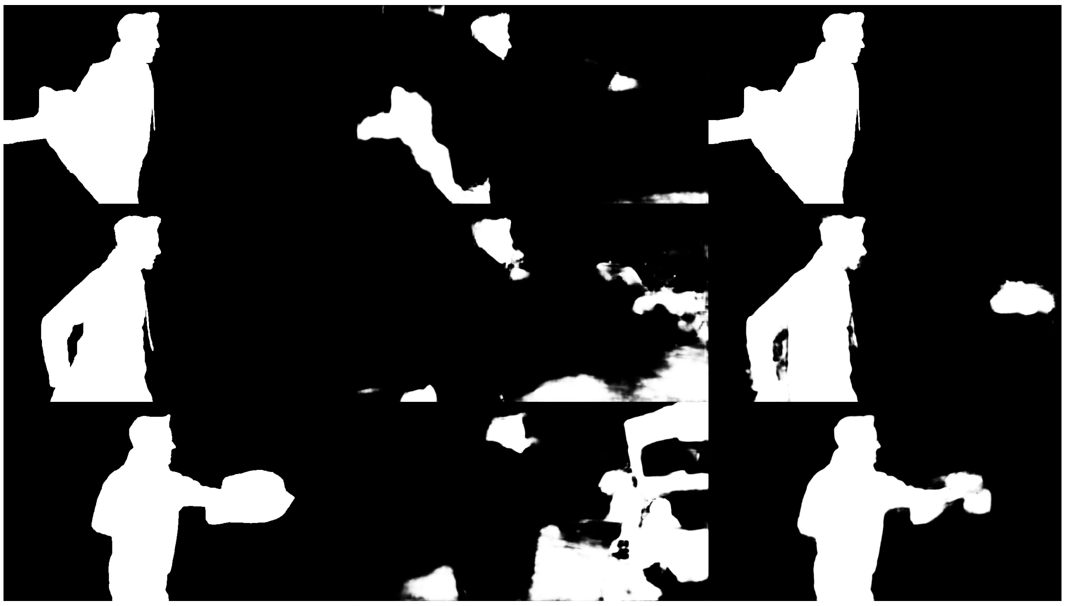


Figure 8. **Additional annotation:** Column (a) shows the ground truth for the custom *jacket-walk* sequence, for which regular one-shot training (b) (as seen in Figure 7) gives $\mathcal{J}$ **mean**/recall/decay scores of **36.5**/27.0/81.9, due to object appearance and background changes. By adding a second ground truth frame from the very end of the sequence to online training (c), the scores improve significantly to **89.5**/100/4.5. Note that both ground truth images together contain most of the background of the scene, as well as the two distinct foreground mask appearances.

a slight bias for this particular algorithm still remains. Three of these ground truth images can be seen in Figure 8. Adding the last frame as ground truth addresses the two core problems of the scene and improves the $\mathcal{J}$ mean from 36.5 to 89.5. Using the first 15 frames instead only improves the score to about 48.0, which shows that a good selection of frames is critical. Adding the last frame to the 15 frames yields a score of (at least) 95.3, which is consistent with the score ceiling for the progressive refinement test in OSVOS as well. A possible compromise to solve only the unknown background problem, without a significantly increased annotation effort, might be to crop out only background sections such that the entire mask is zeros. An initial experiment using two small background crops yields a score of only 46.4, where there are still false positives (even for those entirely negative images) during testing. This can likely be improved though, as at least one problem is that the loss was not scaled to the image size (i.e. the loss on the smaller images had less weight), and a sensible choice for ‘batch size’ would be important as well.

5.6 Parameter Validation

So far, a few of the parameter choices of the OSVOS approach have been investigated, namely the number of parent/online iterations and batch sizes. It would also be interesting to see how well tuned other parameters are, i.e. if there is any room for small improvements, and how much they contribute to the segmentation quality.

An overview over the results is shown in Table 2. Unlike Table 1, the number of iterations is one order of magnitude lower, which leaves the computation time still at around ten hours for the full set. So the first column now represents the new baseline for all comparisons. *Sel9* is the same network that was used in Table 1. One interesting thing to note is the other **self-trained network** *Sel10*, that has a better contour accuracy score, and thus might have overall outperformed *Sel9* in the 20,000 online training iteration evaluation. Then next two columns show the result of changing the online training **batch size** to much higher and lower amounts. The increase yields no significant performance gains, the highest being a lowered decay by 0.5. The decrease was already covered earlier, however the batch size of one here is the baseline for the following evaluation without data augmentation, as an increased batch size there would have no effect. The performance loss without **data augmentation** is the most significant among all evaluations. But surprisingly, the decay scores are also the lowest among all evaluations. This could be explained by the fact that the number of foreground activations is generally lower, including false positives, which is likely the main driver behind an increased decay. This is because false positives are much more likely towards the end of a sequence, where the background has had time to change. Another observation is that the outlines are

Measure	Pre	Sel10	Sel9	Bat9	Bat1	-Aug	-Bal	-WD	Adam	Lim
$\mathcal{J}$ Mean $\uparrow$	**77.2**	76.4	76.6	**77.2**	76.1	73.0	75.4	76.8	77.1	77.0
$\mathcal{J}$ Recall $\uparrow$	88.3	87.3	88.2	88.4	87.1	80.0	85.9	88.2	86.5	**89.0**
$\mathcal{J}$ Decay $\downarrow$	17.5	17.7	17.2	17.0	19.0	**15.0**	23.9	16.9	21.7	**15.0**
$\mathcal{F}$ Mean $\uparrow$	78.5	78.0	77.3	78.5	77.1	74.8	75.0	77.9	**79.0**	77.5
$\mathcal{F}$ Recall $\uparrow$	90.7	89.4	88.7	90.9	89.0	84.9	82.9	**91.2**	89.0	91.0
$\mathcal{F}$ Decay $\downarrow$	19.1	18.5	19.5	18.6	20.6	**16.9**	24.6	18.5	22.9	18.1

Pre	*Pretrained* parent network (PN) with the default online batch size of 5
Sel10	*Self-trained* parent network with a parent batch size of 10
Sel9	*Self-trained* parent network with a parent batch size of 9
Bat9	PN with online *batch* size of 9
Bat1	PN with online *batch* size of 1
-Aug	PN without data *augmentation* for online training (with a batch size of 1)
-Bal	PN without a class *balancing* loss for online training
-WD	PN without *weight decay* for online training
Adam	PN using *Adam* optimizer with a learning rate of 1e-5 for online training
Lim	PN with *limited* learning rates for the first three stages (divided by 50/50/5)

Table 2. **Parameter validation:** Each column shows the scores obtained from an evaluation of a network with one changed parameter, after 2,000 online training iterations over the DAVIS validation set. Unless stated otherwise, parent/online training were done with a batch size of 10/5 respectively. The best scores in each row are highlighted with bold font.

sharper, which could be tied to the higher (false) confidence for predictions (due to the simpler training without regularization), and partly tied to the necessary resampling after data augmentation. Another question is whether the **class balancing** aspect of the loss function is needed at all. A heavy imbalance is known to sometimes cause the network to settle on making constant predictions in classification. Even though the imbalance in this dataset does not seem to be in this range (as covered in Subsection 3.2.2), training the parent network with an imbalanced loss actually did lead to essentially constant predictions, i.e. single large rectangles as the foreground predictions. But interestingly, the results in Table 2 show that even if the network does converge properly, it does so at lower scores, e.g. 3.5 less in $\mathcal{F}$ mean. To get back to the end of Section 5.3, a network without sufficient parent training and an imbalanced loss function for online training leads to a mostly correct *main* output, where only the first *side* output shows a segmentation and the rest is just constant predictions. The final omission that was tested is the **weight decay** parameter for the optimizer. It was at a low value to begin with, so no strong changes are to be expected. Indeed, the scores are generally lower by a small amount, e.g. 0.6 in $\mathcal{F}$ mean. The $\mathcal{F}$ recall however, scores the highest among all evaluations, even if only by a small margin. A qualitative analysis shows that for some sequences the foreground predictions are generally wider, with more blurriness. This is overall bad for the mean score, but does end up helping the scores in later parts of the sequences, where foreground predictions tend to lack. The choice of SGD (with momentum) as the optimizer is perhaps slightly unusual, with **Adam** seeming to be the most popular choice at this time. In any case, it would be interesting to see how they compare in this instance. Using a learning rate of 1e-5, the mean scores are close to SGD, with $\mathcal{F}$ mean actually being the highest among all evaluations; however, this comes at a cost of an increase in decay of up to 4.2 points. Looking at directly at the segmentation masks again, the trend is that the object outlines tend to be a lot sharper, even more so than in the case of no data augmentation. Knowing that more blurry object outlines tend to reduce decay, as seen in the case with no weight decay, it makes sense that the sharper object outlines then cause an increase here. Unlike the increased sharpness in the case of no data augmentation, this does not just dial back foreground activations—instead, it goes both ways, even blowing up small false positives in the background to large and confident predictions. As the number of online training iterations increases, outlines tend to become sharper anyway, so presumably this advantage of Adam would fade, letting it fall behind SGD even further. On the flip side, if less online training iterations were available (i.e. given a time constraint), then Adam actually outperforms SGD, at least for these particular values for the learning rates. An evaluation for 200 online training iterations leaves Adam ahead by 2.0 and 4.2 points in $\mathcal{J}/\mathcal{F}$ mean (76.5/78.1 Adam vs. 74.5/73.9 SGD). The final test, a **limited learning rate** for the early stages of the network, is motivated by the results of Figure 6. What was indicated there is that online training overrides some of the effects of parent training, by introducing some of the hole type errors that can be seen in the case of only online training. A possible solution to this might be to limit changes to the network parameters that correspond to low level (i.e. small) features. An initial test on the *motocross-jump* sequence showed a promising $\mathcal{J}$ mean score improvement of 9.8 points (63.7 to 73.5), with the desired effects. However, this limitation also prevents the network from removing small background clutter, so for the full evaluation, both mean scores end up falling behind the baseline. Still, the achieved $\mathcal{J}$ recall and decay scores are among the best values over all evaluations, with $\mathcal{F}$ recall and decay also taking a close second place. For longer sequences, this late-stage improvement might actually carry the mean score towards an overall lead as well, and the additional background clutter could perhaps also be cleaned up by a post-processing method.

In summary, apart from some situational improvements, the parameters of the algorithm seem to be fairly well tuned. Data augmentation is unsurprisingly important, given the size of the 'dataset' for online training. The batch size, even though consisting of only simple transformations of the same image, proves to be useful. In addition, the class balancing loss, weight decay, and the choice of optimizer all contribute small amounts to the overall performance.

6 Conclusion

At this point, we should have a general overview of the challenge of object segmentation, and a good understanding of one simple, yet effective modern solution to it. Namely, the OSVOS algorithm, can be used to segment even video data. However, especially after doing segmentation by hand, the importance of using the temporal information that is inherent to video data becomes apparent. Another avenue for improvements could be to find loss functions that align more closely with a human approach, e.g. assigning a greater cost to disconnected and smaller shapes. Simple tuning of hyperparameters does not seem likely to be a worthwhile effort. While the focus in this work was mostly on quality, an optimization of computation time, with e.g. live segmentation in mind, remains to be explored further. Knowing some of the failure cases and causes of errors, we could look towards finding new architectures or post-processing methods. Beyond the approach covered here, we should now also be able to better understand entirely different existing approaches for (video) object segmentation, semantic segmentation, and hopefully even other image-based deep learning methods.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. 1, 2.2.1

[2] K. Grace, J. Salvatier, A. Dafoe, B. Zhang, and O. Evans, "When will AI exceed human performance? evidence from AI experts," *CoRR*, vol. abs/1705.08807, 2017. 1

[3] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II." `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019. Last accessed: 2019-02-24. 1

[4] OpenAI, "Openai five." `https://blog.openai.com/openai-five/`, 2018. Last accessed: 2019-02-21. 1

[5] DeepMind, “Alphago.” https://deepmind.com/research/alphago/, Oct 2017. Last accessed: 2019-02-21. 1

[6] S. Caelles, K.-K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool, “One-shot video object segmentation,” in *Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 2.2, 2.2.1, 3.2, ??, ??

[7] K.-K. Maninis, S. Caelles, Y. Chen, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool, “Video object segmentation without temporal information,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018. 1, 3.2.2

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 1, 2.2, 3.2.2

[9] “Semantic image segmentation with deeplab in tensorflow.” https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html, Mar 2018. Last accessed: 2019-02-21. 1

[10] K. Maninis, J. Pont-Tuset, P. A. Arbeláez, and L. V. Gool, “Convolutional oriented boundaries: From image segmentation to high-level tasks,” *CoRR*, vol. abs/1701.04658, 2017. 1, 2.2.1, 5.3

[11] I. Kokkinos, “Surpassing humans in boundary detection using deep learning,” *CoRR*, vol. abs/1511.07386, 2015. 1, 2.2.1

[12] S. Xie and Z. Tu, “Holistically-nested edge detection,” *CoRR*, vol. abs/1504.06375, 2015. 1, 2.2.1, 3.1, 3.2.2, 5.3

[13] G. Bertasius, J. Shi, and L. Torresani, “High-for-low and low-for-high: Efficient boundary detection from deep object features and its applications to high-level vision,” *CoRR*, vol. abs/1504.06201, 2015. 1, 2.2.1

[14] G. Bertasius, J. Shi, and L. Torresani, "Semantic segmentation with boundary neural fields," *CoRR*, vol. abs/1511.02674, 2015. 1, 2.2.1

[15] X. Ren and J. Malik, "Tracking as repeated figure/ground segmentation," *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007. 2.1

[16] N. Marki, F. Perazzi, O. Wang, and A. Sorkine-Hornung, "Bilateral space video segmentation," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 743–751, 2016. 2.1

[17] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph.*, vol. 26, p. 103, 2007. 2.1

[18] F. Perazzi, O. Wang, M. H. Gross, and A. Sorkine-Hornung, "Fully connected object proposals for video segmentation," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 3227–3234, 2015. 2.1, 3

[19] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient hierarchical graph based video segmentation," *IEEE CVPR*, 2010. 2.1

[20] J. Chang, D. Wei, and J. W. Fisher, "A video representation using temporal superpixels," *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2051–2058, 2013. 2.1

[21] S. A. Ramakanth and R. V. Babu, "Seamseg: Video object segmentation using patch seams," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 376–383, 2014. 2.1

[22] Q. Fan, F. Zhong, D. Lischinski, D. Cohen-Or, and B. Chen, "Jumpcut: Non-successive mask transfer and interpolation for video cutout," *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA 2015)*, vol. 34, no. 6, 2015. 2.1

[23] A. Khoreva, F. Perazzi, R. Benenson, B. Schiele, and A. Sorkine-Hornung, "Learning video object segmentation from static images," *CoRR*, vol. abs/1612.02646, 2016. 2.1, 2.2

[24] V. Jampani, R. Gadde, and P. V. Gehler, "Video propagation networks," *CoRR*, vol. abs/1612.05478, 2016. 2.2

[25] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," *CoRR*, vol. abs/1510.07945, 2015. 2.2

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. 2.2.1, 3.1

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 2.2.1

[28] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. 2.2.1

[29] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *CoRR*, vol. abs/1412.7062, 2014. 2.2.1

[30] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. 2.2.1, 3.1

[31] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015. 2.2.1

[32] J. Dai, K. He, Y. Li, S. Ren, and J. Sun, "Instance-sensitive fully convolutional networks," *CoRR*, vol. abs/1603.08678, 2016. 2.2.1

[33] J. Yang, B. L. Price, S. Cohen, H. Lee, and M. Yang, "Object contour detection with a fully convolutional encoder-decoder network," *CoRR*, vol. abs/1603.04530, 2016. 2.2.1

[34] P. H. O. Pinheiro, T. Lin, R. Collobert, and P. Dollár, “Learning to refine object segments,” *CoRR*, vol. abs/1603.08695, 2016. 2.2.1

[35] J. Pont-Tuset, P. Arbelaez, J. T. Barron, F. Marqués, and J. Malik, “Multiscale combinatorial grouping for image segmentation and object proposal generation,” *CoRR*, vol. abs/1503.00848, 2015. 2.2.1

[36] K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. V. Gool, “Deep retinal image understanding,” in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2016. 3.1

[37] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool, M. H. Gross, and A. Sorkine-Hornung, “A benchmark dataset and evaluation methodology for video object segmentation,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 724–732, 2016. 3.2.1, 3.3

[38] L. Torrey and J. Shavlik, “Transfer learning,” in *Handbook of Research on Machine Learning Applications*, 2009. 3.2.2

[39] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 1263–1284, Sep. 2009. 3.2.2

[40] M. Kukar and I. Kononenko, “Cost-sensitive learning with neural networks,” in *ECAI*, 1998. 3.2.2

[41] S. R. Hashemi, S. S. M. Salehi, D. Erdogmus, S. P. Prabhu, S. K. Warfield, and A. Gholipour, “Tversky as a loss function for highly unbalanced image segmentation using 3d fully convolutional deep networks,” *CoRR*, vol. abs/1803.11078, 2018. 3.2.2

[42] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, “Deeply-supervised nets,” in *AISTATS 2015*, vol. 38, 09 2014. 3.2.2

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. 3.2.2

[44] L. Yang, Y. Wang, X. Xiong, J. Yang, and A. K. Katsaggelos, "Efficient video object segmentation via network modulation," *CoRR*, vol. abs/1802.01218, 2018. 3.2.2

[45] P. Voigtlaender and B. Leibe, "Online adaptation of convolutional neural networks for video object segmentation," *CoRR*, vol. abs/1706.09364, 2017. 3.2.2

[46] J. Cheng, Y. Tsai, W. Hung, S. Wang, and M. Yang, "Fast and accurate online video object segmentation via tracking parts," *CoRR*, vol. abs/1806.02323, 2018. 3.2.2

[47] L. Bao, B. Wu, and W. Liu, "CNN in MRF: video object segmentation via inference in A cnn-based higher-order spatio-temporal MRF," *CoRR*, vol. abs/1803.09453, 2018. 3.2.2

[48] J. Luiten, P. Voigtlaender, and B. Leibe, "Premvos: Proposal-generation, refinement and merging for video object segmentation," in *Asian Conference on Computer Vision*, 2018. 3.2.2

[49] A. Khoreva, R. Benenson, E. Ilg, T. Brox, and B. Schiele, "Lucid data dreaming for object tracking," *CoRR*, vol. abs/1703.09554, 2017. 3.2.3

[50] W.-D. Jang and C.-S. Kim, "Online video object segmentation via convolutional trident network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7474–7483, 2017. 3.3

[51] D. R Martin, C. C Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, pp. 530–49, 06 2004. 3.3

[52] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 509–522, April 2002. 3.3

[53] R. Mottaghi, X. Chen, X. Liu, N. Cho, S. Lee, S. Fidler, R. Urtasun, and A. Yuille, "The role of context for object detection and semantic segmentation in the wild," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 891–898, June 2014. 3.4

[54] P. A. Arbeláez, M. Maire, C. C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 898–916, 2011. 3.4