Self-Supervised Video Object Segmentation using Generative Adversarial Networks

A Master Thesis

Submitted to the Faculty of the

Escola Tècnica d’Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

and Northeastern University

by

Ponç Palau Puigdevall

In partial fulfilment of the requirements for the

Master in Telecommunication Engineering

Advisors

Prof. Octavia Camps

Prof. Xavier Giró

Boston, June 2020
Abstract

Video Object Segmentation is arguably one of the most challenging tasks in computer vision. Training a model in a supervised manner in this task requires a high number of manually labelled data, which is extremely time-consuming and expensive to generate. In this thesis, we propose a self-supervised method that leverages the spatiotemporal nature of video to perform Video Object Segmentation using Generative Adversarial Networks. In this context, we design a novel framework composed of two generators and two discriminators that aim to reach an equilibrium to fulfill the task. Both at training and testing time, the model needs only the first mask of the video to be trained end to end, which is possible because it exploits the temporal consistency of videos to self-supervise its training. In addition, we refine the masks predicted by the model with the Sum of Squares polynomial, a tool adopted from the convex optimization community. Although our approach is considerably ambitious, our model achieves promising results on the DAVIS2016 dataset, which are reported both in a qualitative and quantitative manner.
Acknowledgements

I would like to thank Professor Octavia Camps and Professor Mario Sznaier for inviting me to join their lab and providing me with all the freedom of research I could possibly ask for, but at the same time always guiding me towards the right direction. Being able to perform research in the amazing city of Boston has been a dream come true, I most sincerely thank you.

In addition, I would like to thank Professor Xavier Giró-i-Nieto for his guidance and for all his effort in providing state of the art deep-learning courses at UPC.

I would also like to truly thank Marina Alonso Poal for helping me develop the idea of this project and for her valuable constant technical advice. Besides, I would like to acknowledge that I am extremely grateful for being able to share my time with her, for in this experience, she has showed once again her magnificent resilience and ability to make the most out of all situations.

Finally, I would also like to remark the invaluable support of my parents, who always taught me the value of effort with their own example and provided me with an excellent education that brings me where I am today.
# Contents

1 Introduction ........................................ 8
   1.1 Related Work ..................................... 9
   1.2 Our approach ..................................... 9

2 Background ........................................... 12
   2.1 Comparing Probability Distributions .......... 12
      2.1.1 Kullback-Leibler (KL) divergence ......... 12
      2.1.2 Jensen-Shannon (JS) divergence .......... 13
      2.1.3 Wasserstein distance ...................... 14
   2.2 Generative Adversarial Networks ............. 16
   2.3 Inverse Moment Matrix Sum of Squares Polynomial .......... 18

3 Methodology .......................................... 20
   3.1 Problem formulation ............................... 20
   3.2 Loss functions .................................... 21
      3.2.1 Reconstruction Loss ......................... 22
      3.2.2 Foreground adversarial loss .............. 22
      3.2.3 Background adversarial loss .............. 22
   3.3 Algorithm structure ............................... 23
   3.4 Refining masks with SoS polynomial .......... 25
4 Experiments

4.1 Baseline ................................................................. 26
   4.1.1 Architecture .................................................. 26
   4.1.2 Results .......................................................... 31

4.2 Dmask ................................................................. 36
   4.2.1 Results .......................................................... 36

4.3 Dmask-prop ......................................................... 39
   4.3.1 Results .......................................................... 39

5 Conclusions and Future Work 42
List of Figures

1.2.1 Model scheme. ................................. 10

2.1.1 KL/JS comparison. In this picture we study the KL and JS divergences of Gaussian distributions of different means. We can see that as the mean of $q$ increases, the KL and JS reach a point in which no increment is achieved. ................................. 13

2.1.2 Two disjoint probability distributions. ........................................ 15

2.3.1 SoS polynomial example for a Gaussian scalar random variable with $\mu = 0$, $\sigma = 1$. We can see a better approximation of the inverse distribution when using higher order moments. ........................................ 19

2.3.2 SoS polynomial example for a Gaussian mixture. We can see that using moments up to second order is not enough to obtain a good representation of the inverse of the distribution because it is multi modal. Contrarily, increasing the moment order leads to a better representation of the inverse of the distribution. We display the value of $\log Q$ to provide a better visualization. ........................................ 19

3.1.1 Example of an image decomposed in foreground/background by a binary mask ........................................ 20

4.1.1 Ganimation generator’s architecture. This network generates an attention mask $A$ and a color mask $C$, which modify the original image $I_{go}$. [19] ........................................ 27

4.1.2 $G_F$ architecture. Thick arrows represent convolutional layers (blue) or recurrent layers (yellow). Thin arrows denote adding values. ........................................ 28

4.1.3 $G_b$ architecture. Thick arrows represent convolutional layers (blue) or recurrent layers (yellow). Thin arrows denote adding values. ........................................ 29

4.1.4 Baseline model scheme ........................................ 31

4.1.5 (Left) Ground truth background. (Right) Input background masked with noise ........................................ 32
4.1.6 Training results of $G_B$ when input background masked with noise. First column denotes the input to $G_B$ (t=0). Second column shows the output of $G_B$ at time step t=1, $\hat{I}_B^1$. Third column represents $G_B$’s output at timestep (t=5). Notice how the model does not produce a perfect impainted background at t=1 because it is autoregressive, but the results at t=5 are fairly good.

4.1.7 Training results of $G_B$ when input background masked with lines. First row denotes the input to $G_B$ (t=0). Second row shows the input to $G_B$, which is $I_B^0$ with mirroring lines. Third row shows $G_B$’s output at timestep (t=7). Note the frame difference between time steps and how $G_B$ is able to approximate it not only producing similar textures, but also approximating the motion on the sequence as a consequence of the $L_1$ loss.

4.1.8 Training/Validation baseline results of $G_F$. The first three rows show training segmentation results at time steps 1,4,7, proving that our baseline approach works. Fourth and fifth row show a validation result, evidencing that our model is able to generalize fairly good.

4.1.9 Results of refining the mask with SoS polynomial. Left image represents the mask shown in 4.1.8 (middle image of bmx seq.), containing spurious elements in the back of the boy as well as above the helmet. The middle image shows in purple filtered outliers by SoS polynomial, inliers are shown painted in blue. Right image shows the final filtered mask.

4.1.10 Training results where our model underperforms for t=1,4,7. The first row shows a case where our assumption that foreground has a different appearance than background is broken because it is translucent, so $G_F$ does not include it as part of its mask. The second example shows a big spurious spot in the right hand side of the middle image. Third and fourth row show an example where $G_F$ reconstructs part of the background due to the poor performance of $G_B$.

4.2.1 Training/Validation results where our model $D_{mask}$ performs fairly good. The first row shows a case in which despite having spurious elements, the segmented results are acceptable given that are in t=1,5,11. In second row we show a challenging video where $G_F$ produces part of the moving sticks because $G_B$’s impainting was poor. However, $G_F$ produces a good segmentation mask. Third row shows the mallard example previously examined with baseline model, now we can see that the mask embraces more area of the mallard, which is good. However, the spurious elements are not removed. Finally, we show a good validation result.

4.2.2 Jaccard index as a function of time. A shape of a flipped J can be appreciated due to the trade-off between forward and backward propagation.
4.3.1 Qualitative training/validation results where \textit{Dmask-prop} performs reasonably better than previous models. All examples show timesteps $t=1,5,10$. First three rows show training results while the last two show validation examples, achieving the best results so far in both cases. In first and second row, we can appreciate that although the segmentation is not the best possible, masks do not contain the aforementioned spurious elements present in \textit{baseline} and \textit{Dmask}. In validation examples of rows four and five, we can appreciate that our model is capable of generalizing fairly well.

4.3.2 Results of $G_F$ and $G_B$ cooperation. First row exhibits segmentation masks at times $t=1,5,10$. We also show the impainted background for this case to prove that when background is static and simple enough, $G_B$ can estimate it properly despite having entered a "corrupted" version of it. In second row, image on the left corresponds to original background, middle to input background and right to impainted background at time $t=10$.

4.3.3 Qualitative training/validation results where our model \textit{Dmask-prop} underperforms. All examples show timesteps $t=1,5,10$. First row shows the already studied training swing example, which our model does not properly segment, in spite of all improvements. Second row shows a validation case where our model fails, possibly because of the resemblance of foreground with background and poor optical flow.

5.0.1 Proposed new $G_F$ architecture. The reasoning behind this architecture is that $H^t$ estimates the new position of the foreground in the next frame by a perspective transform \textit{H}, applied to the features, while we still modify the appearance of the new foreground through the combination with $A$ and $C$. Moreover, dynamic constraints could be applied to coefficients of $H$, requiring them to have low dynamics.

5.0.2 Example of 2 discrete distributions.

5.0.3 Example of 2 transport plans $\gamma_1$ and $\gamma_2$ for 2 discrete probability distributions. Although the Wasserstein distance results in the same value for both plans, the transport plans differ. We notice how both constraints mentioned in §2.1.3 are fulfilled, ending with $p$ if we marginalize accross the $y$ coordinate of $\gamma(x,y)$ and viceversa. We can appreciate this fact observing the "boxes" at the sides of the grid.
Chapter 1

Introduction

In this thesis, we explore the task of self-supervised Video Object Segmentation (VOS) using a novel Generative Adversarial approach. This idea was developed at Robust Systems Lab (RSL), at Northeastern University, under the supervision of Prof. Octavia Camps, whose research is focused in Computer Vision. This laboratory is also managed by Prof. Mario Sznaier, whose expertise relies in Control and Optimization theory, a field from which we adopt some concepts to enhance the results generated by our baseline model.

The idea of self-supervision is not new in computer vision. A task is considered self-supervised if a model does not have the explicit ground truth labels at training time, but through exploiting some consistency, it can be trained end to end. Nowadays, with the massive amount of media content generated every minute, the fraction of unlabeled data compared to labeled data is huge. Not surprisingly, as of 2020, self-supervision is gaining popularity among the Artificial Intelligence community and it is believed that it will continue expanding in the future.

Recent publications such as [13], [32] explore in a similar way the idea of self-supervising the Video Object Segmentation task. However, we believe that this concept still has a lot to offer, for we try to exploit its potential by tackling it using a Generative Adversarial approach. Our approach is inspired by some papers in the recent literature. Concretely, by GANimation [19], proposed by Pumarola et al., which received the best paper award at ECCV 2018. In this paper, they proposed an unsupervised Generative Adversarial architecture to condition expressions in human faces. Similarly, another work that inspired this idea was the paper presented by Wang et al. at CVPR 2019 [25], where a tracker is trained in an unsupervised manner by exploiting the fact that the location of an object tracked backward and then forward must match. In addition, the idea of leveraging the reconstruction of video sequences to self-supervise the training of an algorithm using a Generative Adversarial approach appeared in [16]. Joining the fundamental ideas of these three papers, we considered that applying Generative Adversarial Networks to solve the problem of self-supervised Video Object Segmentation would be worth investigating.
Video Object Segmentation is a binary labeling problem aiming to separate foreground object(s) from the background region of a video. Training an algorithm on such a challenging task requires a considerable amount of labelled data, not only in the spatial dimension (HxW labels) but in the temporal dimension (TxHxW). Thus, the number of research publications related to VOS increased remarkably after the release of the DAVIS 2016 dataset [18], where only single objects are manually annotated. Two years later, Youtube-VOS dataset was released by Xu et al. [30], increasing even more the research on this field. Youtube-VOS is more challenging than DAVIS2016 because it contains more difficult and longer video sequences. Since the purpose of this thesis is to study and analyze the proposed approach, we worked on the DAVIS2016 dataset, where only a single object per frame is annotated. For this reason, from now on in this thesis we use the term object/foreground interchangeably.

1.1 Related Work

Solving the problem of VOS can be addressed using a wide range of different techniques.

On one hand, there exist supervised techniques. Some of these approaches rely on tracking while segmenting [23], [24], [12], trying to find the most similar object in the next frame given a reference object. Usually this is done at the price of not updating the features of the reference object, therefore not adapting well to changes in the object across time. Other approaches rely precisely on capturing the features of the reference object and updating them across time, such as [23], [30] and [2]. In [23], they propose a recurrent neural network architecture on both space and time domains, while in [30] they capture temporal features using a convolutional LSTM [29]. In [2], they propose a spatio-temporal Generative Adversarial Network similar to our idea, but trained in a supervised manner.

On the other hand, unsupervised methods rely on the assumption that the object to segment has uncorrelated motion with respect to the background. However, it needs to be remarked that often there is a misconception regarding the meaning of the term unsupervised in VOS. Some work in the literature is posed as unsupervised although they train their algorithm in a supervised way. They do so because they do not use any reference mask at testing time, so the reference object must be detected at test time. Several works exploiting this idea include [15] and [31]. Nonetheless, very recent publications like [14], [32] and [13] similarly to us, use only one mask at training time, achieving notable results.

1.2 Our approach

In this subsection we briefly describe the idea of this thesis in a high level manner, without going into mathematical details. Our approach consists of a set of networks that aim to solve the task of video
object/foreground segmentation by generating the foreground and background of the next frame. Our model is trained in an adversarial manner having only as available data the video sequence (from which a flow prediction is computed) and the foreground mask of the first frame. We emphasize the restrictive constraint that using only one mask both at training and testing time implies. We assume that the object to segment has singular appearance and motion features that allow us to distinguish it from the background or the objects in the background, although they might look similar to the object of interest.

The model is composed of two pairs of Generator and Discriminator modules, one for the foreground $(G_F, D_F)$ and another for the background $(G_B, D_B)$. On one hand, the goal is to despite having only the mask in the first frame, the first frame and the optical flow, for the foreground generator $G_F$ to be capable of generating an approximation of the next $T$ masks, jointly with an approximation of the foregrounds in the corresponding frames.

On the other hand, we expect the background generator $G_B$ to produce plausible backgrounds that also fill/impaint the foreground area, that together with the foregrounds generated by $G_F$ will compose the next video frames.

The proposed architecture is depicted in figure 1.2.1. This architecture leverages the nature of video to train itself in a self-supervised way. We believe that while training the generative model on the quasi video prediction task (we compute the optical flows between frames) we can force the network to learn a simpler task such as object/foreground segmentation.

Figure 1.2.1: Model scheme.
In this context, three adversarial losses or "fights" are posed to achieve the desired result:

1. **Reconstruction term and video consistency**: $G_F$ and $G_B$ against $L_1$ loss. With this adversarial process we are imposing that the foreground generated by $G_F$ plus the background generated by $G_B$ must be equal to the next frame. This loss helps the model not only to self-supervise the training, but to generate realistic images.

2. **Foreground Adversarial Loss**: $G_F$ against $D_F$. With this adversarial process we are forcing that the fake foregrounds of the next frames generated by $G_F$ have the same distribution as the real foreground (the one of the first frame), that is, a black background with an object on it.

3. **Background Adversarial Loss**: $G_B$ against $D_B$. With this loss we require $G_B$ to produce an inpainted background that looks consistent with the background of the first frame. Although we are not looking for the perfect background inpainter, this is important for our approach because without this loss we would not be able to exploit the reconstruction loss, thus not being able to self-supervise the training. $G_B$ is trained against $D_B$ in a patch-wise manner, not with the entire image. This is due to the fact that the distribution of the first frame’s background is composed of a background plus a black area and the distribution of the fake background is composed by an entire background image.

Both $G_F$ and $G_B$ are auto-regressive, in the sense that they generate next frame samples from their own previous generations. In order to ease this process, we design them in a non-causal fashion so they have both information about the past (a state that is updated each time step and the previous frame) and information about the future (optical flow from $t$ to $t+1$).

In the course of this thesis, we review some fundamental background theory applied in this work in chapter §2. We formulate our problem in chapter §3 and we show the implementation details and experiments in chapter §4. Finally, we conclude the work with a conclusions and future work chapter.
Chapter 2

Background

In this chapter we introduce fundamental theoretical concepts behind this work. When training a Generative Model we need to compare different probability distributions. Therefore, we begin explaining different metrics to compare them. We continue explaining the concept of Generative Adversarial Network (GAN), the type of Generative Model used in this work. We end the chapter introducing the notion of Sum of Squares polynomial (SoS), a tool that allows us to approximate a probability distribution with only its samples.

2.1 Comparing Probability Distributions

Given two separate probability distributions $p(x)$ and $q(x)$, there exist several metrics to quantify the similarity between them. In this subsection, we begin explaining the KL divergence, studying its strengths, weaknesses and noticing the need for other metrics, such as the Jensen-Shannon (JS) divergence, which in turn has also its flaws. Finally, we explain the Wasserstein or Earth Mover’s distance, a robust distance for probability distributions. We conclude the chapter with a simple yet illustrative example.

2.1.1 Kullback-Leibler (KL) divergence

The Kullback-Leibler (KL) divergence is defined as:

$$KL(p \parallel q) = \mathbb{E}_{x \sim p} \left[ \log \frac{p(x)}{q(x)} \right] = \begin{cases} \int x p(x) \log \frac{p(x)}{q(x)} dx & \text{if continuous} \\ \sum x p(x) \log \frac{p(x)}{q(x)} & \text{if discrete} \end{cases}$$

It measures how $p$ differs from $q$. When $p(x) = q(x)$, the KL divergence becomes 0 meaning that the two distributions are equal. However, KL divergence exhibits some problems that may appear when comparing two distributions.
• If $p(x)$ is close to 0 but $q(x)$ is significantly non-zero, the denominator has almost no effect in the KL although the distributions diverge.

• KL divergence is not symmetric, $D_{KL}(p \parallel q) \neq D_{KL}(q \parallel p)$.

• KL saturates when comparing two very different distributions. We can appreciate this fact in Fig. 2.1.1.

![Figure 2.1.1: KL/JS comparison. In this picture we study the KL and JS divergences of Gaussian distributions of different means. We can see that as the mean of $q$ increases, the KL and JS reach a point in which no increment is achieved.](image)

2.1.2 Jensen-Shannon (JS) divergence

The Jensen-Shannon (JS) divergence is defined as:

$$JS(p \parallel q) = \frac{1}{2} KL\left(p \parallel \frac{p + q}{2}\right) + \frac{1}{2} KL\left(q \parallel \frac{p + q}{2}\right)$$  \hspace{1cm} (2.2)

Although JS divergence is symmetric, it manifests the same problem as the KL, saturating when $p$ and $q$ differ in a significant way, as illustrated in figure 2.1.1.
2.1.3 Wasserstein distance

The last metric we present in this section is the Wasserstein or Earth Mover’s (EM) distance. This distance solves the aforementioned problems of both the KL and JS divergences of failing when comparing disjoint distributions.

For continuous probability distributions, the Wasserstein Distance is defined as

\[ W(p, q) = \inf_{\gamma \sim \Pi(p, q)} \mathbb{E}_{(x, y) \sim \gamma} ||x - y|| \]  

(2.3)

where:

- \( \Pi(p, q) \) is the set of all possible joint probability distributions between \( p \) and \( q \). One joint distribution \( \gamma \) represents one mass transport plan (see Appendix B for more details).

- \( \gamma(x, y) \) represents the percentage of mass that should be moved from point \( x \) to \( y \), with the constraints of:

\[ \sum_x \gamma(x, y) = p(y) \]
\[ \sum_y \gamma(x, y) = q(x) \]  

(2.4)

Which are interpreted as when we finish moving all the planned amount of mass (\( \gamma(x, y) \)) from every possible \( x \) to \( y \), we must end up with exactly with what \( y \) had according to \( q \) and vice versa.

- If \( \gamma(x, y) \) is the amount of mass moved and \( ||x - y|| \) is the travelling distance, we define the cost as \( \gamma(x, y) \cdot ||x - y|| \).

Instead of relying on a ratio of the values of the distributions, the Wasserstein distance can be interpreted as the minimum energy cost of moving and transforming a pile of mass in the shape of one probability distribution to the shape of the other distribution. This cost is quantified by the amount of mass moved times the moving distance. In Appendix B we provide a simple example to show insights on this metric in which we compare two discrete probability distributions.

The interpretation of equation 2.3 is reasonably clear. We are trying to find the smallest cost (inf) among all the expected costs:

\[ \sum_{x, y} \gamma(x, y) \cdot ||x - y|| = \mathbb{E}_{(x, y) \sim \gamma} ||x - y|| \]  

(2.5)

This optimal cost can be found solving a Linear Programming problem. The details of how this is carried out are out of the scope of this thesis, but can be found in [6].

We end the section with a simple yet insightful example proposed in the original Wasserstein GAN paper [1], ratifying the robustness of this metric compared to the KL and JS.
Suppose we have two disjoint probability distributions $P_0$ and $P_\theta$ as shown in Fig. 2.1.2 such that:

\[
\forall (x, y) \in P_0, \ x = 0 \text{ and } y \sim U(0, 1)
\]
\[
\forall (x, y) \in P_\theta, \ x = \theta, \ 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1)
\] (2.6)

Figure 2.1.2: Two disjoint probability distributions.

We now discuss how the different metrics perform when comparing these two probability distributions. On one hand, we have the KL divergence, since the two probability distributions are disjoint, we have:

\[
KL(P_0 || P_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}
\] (2.7)

Then, we have the JS divergence, which is computed as:

\[
JS(P_0 || P_\theta) = \frac{1}{2} KL\left( P_0 || \frac{P_0 + P_\theta}{2} \right) + \frac{1}{2} KL\left( P_\theta || \frac{P_0 + P_\theta}{2} \right)
\] (2.8)

We show how this calculation is carried out for the first term in the above equation. Let us first name $P_m = \frac{P_0 + P_\theta}{2}$ for simplicity. Then:

\[
KL(P_0 || P_m) = \sum_x \int_{-\infty}^{\infty} P_0(x, y) \log \frac{P_0(x, y)}{P_m(x, y)} dy
\] (2.9)

\[
= \int_0^1 P_0(0, y) \log \frac{P_0(0, y)}{P_m(0, y) + P_m(\theta, y)} dy + \int_0^1 P_0(\theta, y) \log \frac{P_0(\theta, y)}{P_m(\theta, y) + P_m(\theta, y)} dy
\] (2.10)

\[
= \int_0^1 P_0(0, y) \log 2 \ dy = \log 2
\] (2.11)

Following the same reasoning, we obtain $KL(P_\theta || P_m) = \log 2$. Therefore, $JS(P_0 || P_\theta)$ becomes:

\[
JS(P_0 || P_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}
\] (2.12)
Finally, the Wasserstein distance is

$$W(P_0\|P_\theta) = |\theta|$$  \hfill (2.13)

Summarizing all the results, we can see the true power of the Wasserstein distance. KL divergence results in $\infty$ when the two distributions are disjoint, JS suddenly jumps from 0 to log $2$, not being differentiable at $x = 0$. The Wasserstein distance provides a smooth and intuitive measure of how much both distributions differ, thus making it a more desirable metric for training a network.

### 2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a type of Generative Models (GM). In a deep learning context, we are given a dataset containing $N$ samples coming from an $M$ - dimensional probability distribution $p_{\text{data}}(x)$. GM model this data distribution by learning it, resulting in a new probability distribution $p_{\text{model}}(x)$ from which we can derive new samples.

GANs have shown great results since their first appearance in 2014 [4]. The concept of GAN is inspired by game theory, where a generator and discriminator compete with each other while making each other stronger at the same time until hopefully they reach a Nash equilibrium. A GAN scheme consists of two models:

- A **discriminator $D$** that estimates the probability of a given sample coming from the real dataset (with distribution $p_r$). It is optimized to differentiate the real samples from the fake ones, that is, not being cheated by the fake samples created by $G$.

- A **generator $G$** that outputs synthetic samples given a random variable $z$ (with distribution $p_z$). It is trained to capture the real data distribution $p_r$ so its generated samples look as real as possible. Its goal is to trick the discriminator $D$ so that it cannot tell if the sample is coming from the real distribution $p_r$ or from the generator’s distribution $p_g$.

If we define $x$ as a sample and a binary label such that $[0 : \text{fake}, 1 : \text{real}]$, we can formalize the GAN problem as follows:

- On one hand, the discriminator is trained to make sure that it can distinguish real data and fake data:
  - Maximizing $E_{x \sim p_r}(x)[\log D(x)]$, which happens when $D$ knows that a sample is real, $D(x) = 1$.
  - Maximizing $E_{z \sim p_z}(z)[\log(1 - D(G(z)))$, which happens when $D$ knows that a sample generated by $G$ is fake, $D(G(z)) = 0$.

- On the other hand, the generator is trained to increase the chances of $D$ producing a high probability for a fake example, which is achieved by
- Minimizing $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))])$, which happens when $D(G(z)) = 1$.

Combining both points stated before, $D$ and $G$ play what is known as **minimax game**, a zero-sum game that optimizes the following loss function:

$$
\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
$$

Equation 2.14 defines the loss function for the first GAN proposed in [4], however, this setup shows many flaws that can easily jeopardize its training. Among others, we find the following to be the most relevant:

- If we have a good discriminator but a poor generator, that is, $p_g$ differs significantly from $p_r$ then the gradient for the generator diminishes and the generator learns barely anything, as can be seen in the following equation:

$$
- \nabla_{\theta_g} \log(1 - D(G(z))) \to 0
$$

- The other problem resides in the distance used to compare $p_r$ and $p_g$ in the original GAN paper. Before extracting conclusions, let us review some details about the original GAN loss.

Rewriting Eq. 2.14 as

$$
\int_x \left( p_r(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \right) dx
$$

We find (Appendix A) that the discriminator’s optimum value $D^*(x)$ for Eq. 2.16 is

$$
D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)} \in [0, 1]
$$

When the generator is trained to its optimal ($p_g = p_r$), then $D^*(x)$ becomes 0.5. When both $G$ and $D$ reach their optimal values, the loss function’s global optimal becomes

$$
L(G, D^*) = \int_x \left( p_r(x) \log(D^*(x)) + p_g(x) \log(1 - D^*(x)) \right) dx
$$

$$
= \log \frac{1}{2} \int_x p_r(x) dx + \log \frac{1}{2} \int_x p_g(x) dx
$$

$$
= -2 \log 2
$$

Moreover, it can be proven that

$$
L(G, D^*) = 2JS(p_r || p_g) - 2 \log 2
$$

When $p_r = p_g$ we reach the optimal value of the loss function stated in 2.18. Essentially, grouping the above result with the one obtained in 2.18, we can conclude that the GAN loss function measures the similarity between $p_r$ and $p_g$ using the JS divergence. As a consequence, following the reasoning stated in the previous section §2.1, in [1] they propose to train a GAN using Wasserstein distance, showing far better results than previous approaches, which was later improved by the work proposed in WGAN-GP [5]. Thus, in this work we train a GAN using the methodology stated in [5].
2.3 Inverse Moment Matrix Sum of Squares Polynomial

In this section we introduce the inverse moment matrix Sum of Squares (SoS) polynomial, also referred to as SoS polynomial in this work. Grounded by polynomial and statistical moments theory, this mathematical tool introduced by Lasserre and Pauwels in [17] is able to **capture the inverse shape of a density function given its samples**. This tool has been successfully applied in the outlier detection task in [22]. In this thesis, we use it as a means to refine the masks produced by our model. Before defining the concept of SoS, we provide the mathematical definition of statistical moment. A random variable $X$ with density $\mu$ can be characterized by its statistical moments. The $n$-th statistical moment of a random variable is defined as:

$$ m_n = \mathbb{E}[X^n] = \begin{cases} \int_{-\infty}^{\infty} x^n f_X(x) dx & \text{if } X \text{ is continuous} \\ \sum_k x_k^n P_X(x_k) & \text{if } X \text{ is discrete} \end{cases} $$

Having said that, we introduce the definition of SoS polynomial:

$$ Q_{\mu,n}(x) = v_n(x)^T M_d(\mu)^{-1} v_n(x) $$

Equation 2.21

$Q_{\mu,n}(x)$ represents the inverse moment-matrix SoS polynomial of degree $2d$ associated to $\mu$. Let us define every term in equation 2.21:

- $x \in \mathbb{R}^d$ is a $p$-dimensional sample of a set of inliers in our distribution $\mu$.
- $v_n(x)$ represents the Veronese map up to order $n$. A Veronese map is composed by all $s_{n,d} = \binom{n+d}{n}$ possible monomials of order $n$ in $d$ variables, grouped in lexicographic order:

$$ v_n([x_1 ... x_d]) = \begin{bmatrix} 1 & x_1^n & x_2^n & \ldots & x_d^n \end{bmatrix}^T $$

Equation 2.22

- $M_d(\mu)^{-1} \in s_{n,d} \times s_{n,d}$ symbolizes is the inverse of the matrix containing all the moments up to $2n$ of the distribution $\mu$. This matrix is defined as follows:

$$ M_n = \frac{1}{S} \sum_{i=1}^{S} v_n(x_i)v_n(x_i)^T $$

Equation 2.23

As an example, we obtain the following moment matrix if we are given a distribution of $S$ samples of dimension 1 and we use moments up to $n = 2$, therefore having a moment matrix with moments up to $2n$.

$$ M_4 = \frac{1}{S} \sum_{i=1}^{S} \begin{bmatrix} 1 & x_i & x_i^2 \end{bmatrix} \begin{bmatrix} 1 & x_i & x_i^2 \\ x_i & x_i^2 & x_i^3 \\ x_i^2 & x_i^3 & x_i^4 \end{bmatrix} $$

Equation 2.24

The value of $Q_{\mu,n}(x)$ is **low if we evaluate an inlier and high if we evaluate an outlier**. Moreover, in [17] they define a threshold $m$ above which the sample being evaluated is an inlier or an outlier. Therefore, we can exploit this fact to classify new samples with $Q_{\mu,n}$:

$$ x := \begin{cases} \text{outlier} & \text{if } Q_{\mu,n}(x) > m \\ \text{inlier} & \text{if } Q_{\mu,n}(x) \leq m \end{cases} $$

Equation 2.25
The value of this threshold is $s_{n,d}$, as stated in [17], which is the expected value of $Q_{\mu,n}(x)$.

Concluding this section, we provide two examples where we apply the concept of SoS polynomial, one with a Normal distribution and another composed by a mixture of Gaussian distributions. In both cases the number of examples $S > s_{n,d}$, to properly create the empirical moment matrix.

![SoS polynomial example for a Gaussian scalar random variable with $\mu = 0$, $\sigma = 1$. We can see a better approximation of the inverse distribution when using higher order moments.](image1.png)

**Figure 2.3.1:** SoS polynomial example for a Gaussian scalar random variable with $\mu = 0$, $\sigma = 1$. We can see a better approximation of the inverse distribution when using higher order moments.

![SoS polynomial example for a Gaussian mixture. We can see that using moments up to second order is not enough to obtain a good representation of the inverse of the distribution because it is multi modal. Contrarily, increasing the moment order leads to a better representation of the inverse of the distribution. We display the value of log $Q$ to provide a better visualization.](image2.png)

**Figure 2.3.2:** SoS polynomial example for a Gaussian mixture. We can see that using moments up to second order is not enough to obtain a good representation of the inverse of the distribution because it is multi modal. Contrarily, increasing the moment order leads to a better representation of the inverse of the distribution. We display the value of log $Q$ to provide a better visualization.
Chapter 3

Methodology

In this chapter we formalize the problem, define the inputs and outputs of each network and present the formal loss functions optimized by our model. We do not describe the network architectures and implementation details until chapter §4.

3.1 Problem formulation

We propose an approach to the problem introduced in section §1.2 by dividing the image content into Object/Foreground and Background. An image can be partitioned by a binary mask with ones in the foreground part as follows:

\[ I = M \odot I_F + (1 - M) \odot I_B \]  \hspace{1cm} (3.1)

Where \( \odot \) denotes element-wise multiplication, \( I \) symbolizes the image, \( M \) is a binary mask, \( I_F/I_B \) represent the foreground/background part of the image, respectively.

Figure 3.1.1: Example of an image decomposed in foreground/background by a binary mask.
Since we are dealing with video sequences of length $T$, that is, a set of images $(I^0, I^1, ..., I^{T-1})$, we can define a video frame at time step $t$ as:

$$I^t = M^t \otimes I^t_F + (1 - M^t) \otimes I^t_B$$  \hspace{1cm} (3.2)

The aim of this work is to predict the masks $(M^1, M^2, ..., M^{T-1})$ if we are only given the first frame mask $M^0$. That means that we have the ground truth partition of the frame 0 but we do not know how the frames ahead are divided. The idea of this work is to exploit the aforementioned decomposition by estimating the three unknown elements $M^t$, $I^t_F$ and $I^t_B$, when $t > 0$.

Please note that in order to achieve self-supervision we could avoid estimating $I^t_F$ and $I^t_B$, since we only need $M$ to divide the image into foreground and background. However, we do believe that by also estimating $I^t_F$ and $I^t_B$ the network should do it better, because the knowledge needed to generate $I^t_F$ and $I^t_B$ is higher than just estimating $M$. Therefore, the more knowledge the network has, the better the masks should be.

Having said that, we propose an architecture composed by 2 pairs of Generator and Discriminator, one for the foreground ($G_F, D_F$) and another for the background ($G_B, D_B$). Both $G_F$ and $G_B$ are auto-regressive, recurrent networks. This detail is necessary to model the temporal dimension of video sequences. We could not rely just on a CNN to do a task like this because the network would forget past information at each time step. Since we also want to include some information about the future, we use Optical Flow (OF). Finally, we formalize the generators as follows:

$$\hat{I}^t_F, \hat{M}^t = \begin{cases} 
\hat{I}^1_F, \hat{M}^1 = G_F(I^0_F, OF^0_F, h^0) & \text{if } t = 1 \\
\hat{I}^t_F, \hat{M}^t = G_F(\hat{I}^{t-1}_F, OF^{t-1}_F, \hat{h}_{t-1}) & \text{if } t > 1 
\end{cases}  \hspace{1cm} (3.3)$$

$$\hat{I}^t_B = \begin{cases} 
\hat{I}^1_B = G_B(I^0_B, h^0) & \text{if } t = 1 \\
\hat{I}^t_B = G_B(\hat{I}^{t-1}_B, \hat{h}_{t-1}) & \text{if } t > 1 
\end{cases}  \hspace{1cm} (3.4)$$

Note in equation (3.4) that $G_B$ has no Optical Flow as input. Our experiments showed that removing it from its input produced better results.

Having generators to predict the foreground, background and masks is useful, but how do we ensure that the generators generate foregrounds, backgrounds and masks that look real? With the help of discriminators $D_F$ and $D_B$. By introducing these networks we create adversarial processes that will hopefully reach an equilibrium when the desired behaviour is accomplished. Further details on the discriminator functions are described in the following section.

### 3.2 Loss functions

In this section we explain in detail the three loss functions introduced in section §1.2 and provide details about $D_F$ and $D_B$. Each one of these loss functions has a specific purpose, which is described below.
3.2.1 Reconstruction Loss

The consistency we use in this work to self-supervise the training is that \( \hat{I}^{t+1} \) must be equal to \( I^{t+1} \). In order to attain this objective, we minimize the Absolute Error or \( L_1 \) loss. Concretely:

\[
L_1 = \sum_{t=1}^{T-1} ||I^t - \hat{I}^t||_1
\]

Note that \( \hat{I}^t \) depends on both \( G_F \) and \( G_B \), so they cooperate against the \( L_1 \) loss to ensure the following:

- Video reconstruction (consistency).
- Realistic generated images.

3.2.2 Foreground adversarial loss

In order to ensure that \( \hat{I}_F^t \) belongs to the distribution of real foregrounds \( p_{fg} \), which intuitively can be understood as an image composed by a black background with an object/foreground on it, we set a discriminator \( D_F \) to discriminate between real/fake foreground. However, in our context there exists only one real foreground per \( T-1 \) fake foregrounds, therefore having just one sample of the real foreground distribution for each video, far less than the amount of samples needed to characterize a distribution. In order to solve that problem, we adopt the idea proposed by Khoreva et al. in Lucid Data Dreaming \[12\], where they perform a wide range of affine and perspective transformations to train a VOS algorithm. Thus, our discriminator \( D_F \) is fed with several transformations of \( I_0^F \), let us name them \( f(I_0^F) \), to augment its distribution. Finally, the foreground adversarial Wasserstein loss with gradient penalty becomes:

\[
L_F = \sum_{t=1}^{T} D_F(\hat{I}_F^t) - D_F(f(I_0^F)) + \lambda_{gp} \left( \left\| \nabla D_F(\alpha f(I_0^F) + (1 - \alpha)\hat{I}_F^t) \right\|_2 - 1 \right)^2
\]

3.2.3 Background adversarial loss

Estimating the background in video sequences can be challenging because the background distribution is far more complex than the foreground distribution. The background of a video sequence usually contains either a set of rather constant textures or a diversity of multiple objects that can differ or not from the foreground. In this work, we focus on having a good enough background estimator in order to achieve self-supervision. Note that we are not looking for the best background inpainter, but for one that produces a good enough background (similar to the rest of the image) when the foreground changes its location across time.

We propose a patch based Discriminator \( D_B \) inspired by the work of Isola et al. \[10\] and \[20\]. At training time, \( D_B \) will randomly sample \( N_p \) fake patches \( P_f \) coming from \( \hat{I}_B^t \) for each time step, that is, evaluating \( T \times N_p \) patches for each sample in the batch. These patches are taken over the whole image \( \hat{I}_B^t \) because all the image area should belong to the background distribution. However, when
evaluating real patches of $I_B^0$, we take into account the mask to only take $N_p$ real patches where $M^0 = 0$, that is, $N_p$ real patches $P_r$ coming from $I_B^0$. Back in section §1.2 figure 1.2.1 depicts this idea.

Finally, the loss function becomes

$$L_B = \sum_{t=1}^{T} \sum_{i=1}^{N_p} D_B(P^t_{f_i}) - D_B(P^t_{r_i}) + \lambda_{gp} \left( \| \nabla D_B(\alpha P_{r_i} + (1 - \alpha) P_{f_i}) \|_2 - 1 \right)^2 \quad (3.7)$$

We finally combine linearly the three losses with their corresponding weights, defining the total loss $\mathcal{L}$

$$\mathcal{L} = \lambda_{rec} \mathcal{L}_1 + \lambda_F \mathcal{L}_F + \lambda_B \mathcal{L}_B \quad (3.8)$$

Which is posed as a minimax optimization problem as

$$G^*_F, G^*_B = \min_{G_F, G_B} \max_{D_F, D_B} \mathcal{L}(D_F, D_B, G_F, G_B) \quad (3.9)$$

### 3.3 Algorithm structure

In this subsection we summarize the procedure in order to train our model in form of an algorithm or pseudo-code. For simplicity, we assume a batch size of 1 in order to avoid adding another for loop in the algorithm. We assume that $G_F, G_B, D_F, D_B$ have weights $\theta_F, \theta_B, \phi_F, \phi_B$ respectively, which are optimized by Adam optimizers with parameters $\alpha, \beta_1$ and $\beta_2$. We also make use of all the notation introduced before in this section. Although the algorithm may seem a little bit overwhelming at first sight, it represents the typical procedure to train a GAN, first training the discriminators $n_d$ times and then the generators.
Algorithm 1: Project pipeline. $n_d$ represents how many times we train the discriminators per one weight update in the generators.

while $\theta_F$ and $\theta_B$ have not converged do
    for $i = 0, 1, \ldots, n_d$ do
        Train discriminators $n_d$ times
        for $t = 0, 1, \ldots, T - 2$ do
            if $t = 0$ then
                $\hat{I}_F^1, \hat{M}^1 = G_F(I_F^0, OF_1^0, h_0)$
                $\hat{I}_B^1 = G_B(I_B^0, h_0)$
            else
                $\hat{I}_F^{t+1}, \hat{M}^{t+1} = G_F(\hat{I}_F^t, OF_1^t, h_t)$
                $\hat{I}_B^{t+1} = G_B(\hat{I}_B^t, h_t)$
            end
            Detach $\hat{I}_F^{t+1}, \hat{M}^{t+1}, \hat{I}_B^{t+1}$ (to not calculate gradients w.r.t $G_F, G_B$)
            $L_F = L_F + D_F(\hat{I}_F^{t+1}) - D_F(f(I_F^0)) + \lambda_{gp}(\|\nabla D_F(\alpha f(I_F^0) + (1 - \alpha)\hat{I}_F^{t+1})\|_2 - 1)^2$
            Randomly sample $N_p$ patches from real and fake background
            $P_f^{t+1} \leftarrow \hat{I}_F^{t+1}; P_r \leftarrow I_B^0$
            $L_B = L_B + D_B(P_f^{t+1}) - D_B(P_r) + \lambda_{gp}(\|\nabla D_B(\alpha P_r + (1 - \alpha)P_f^{t+1})\|_2 - 1)^2$
        end
        Update parameters of discriminators
        $\phi_F, \phi_B \leftarrow \nabla_{\phi_F, \phi_B}(L_F + L_B)$
    end
    Train generators
    for $t = 0, 1, \ldots, T - 2$ do
        if $t = 0$ then
            $\hat{I}_F^1, \hat{M}^1 = G_F(I_F^0, OF_1^0, h_0)$
            $\hat{I}_B^1 = G_B(I_B^0, h_0)$
        else
            $\hat{I}_F^{t+1}, \hat{M}^{t+1} = G_F(\hat{I}_F^t, OF_1^t, h_t)$
            $\hat{I}_B^{t+1} = G_B(\hat{I}_B^t, h_t)$
        end
        $\hat{L}_F = L_F - D_F(\hat{I}_F^{t+1})$ Note the minus sign to create minimax problem
        Randomly sample $N_p$ patches from fake background
        $P_f^{t+1} \leftarrow \hat{I}_B^{t+1}$
        $L_B = L_B - D_B(P_f^{t+1})$
        $\hat{I}^{t+1} = M^{t+1} \otimes I_F^{t+1} + (1 - M^{t+1}) \otimes I_B^{t+1}$ Create next frame
        $L_1 = L_1 + ||I^{t+1} - \hat{I}^{t+1}||_1$
    end
    Update parameters of generators
    $\theta_F, \theta_B \leftarrow \nabla_{\theta_F, \theta_B}(L_F + L_B + L_1)$
end
3.4 Refining masks with SoS polynomial

As it will be shown later in this work, our baseline and Dmask model generate segmentation masks that are noisy, with some spurious elements on it. To refine the masks at test time we propose to evaluate estimated masks $\hat{M}^t$ with a SoS polynomial built with the RGB values of the first ground truth foreground $I^0_F$, which we have available in test time. In order to apply the polynomial to the estimated masks $\hat{M}^t$, we evaluate the pixels of the original image at time $t$, $I^t$, in the positions where $\hat{M}^t = 1$. The SoS polynomial used to filter estimated masks is built using moments of 6th order.
Chapter 4

Experiments

In this chapter we give implementation details and results of different models explored in this work. By models we refer to combinations of possible Generators and Discriminators. We begin explaining the baseline model, going into further details about the networks used for each architecture in it. We analyze its qualitative and qualitative results on the DAVIS2016 dataset\textsuperscript{1}\textsuperscript{18}. Based on these results, we study those aspects that need improvement, which naturally will lead us to new models. Since these models rely on the main structure of the baseline model, we only explain the changes with respect to it, and also analyze the results on DAVIS2016 and compare them. In addition, for each model we evaluate its off-the-shelf performance and its performance applying post-processing with Sum of Squares polynomial. Finally, we compare results of all our models quantitatively and qualitatively.

4.1 Baseline

4.1.1 Architecture

The baseline model is composed of 2 pairs generator/discriminator, each one described in the sections below.

Foreground Generator

The foreground generator $G_F$ is the function in charge of generating both the next frame’s foreground and the next frame’s mask. To do so, we modify the network used in $GANimation$\textsuperscript{19}, depicted in figure 4.1.1, which is already a modification of the network proposed by Johnson \textit{et al.} in [11].

We take this network as a base for $G_F$ because in [19] it was used to simulate the effect of changing

\textsuperscript{1}Although simpler datasets for working with video sequences like Moving MNIST\textsuperscript{21} exist, they do not contain background, thereby not being a suitable dataset for this work.
a face expression in a continuous manner. This continuous effect is present in the temporal nature of video sequences so we thought of adding this network to our work. However, in [19] they only worked with 2 frames, while we are working with video sequences of length $T$. Hence, we adjust the network with two modifications, summarized in:

- We include a mask branch generating segmentation masks $\hat{M}$.
- We design the network to be recurrent in a fashion similar to [2] and [28], passing transformed convolutional features from one time step to the next in order to capture past information. Although these 2 works concatenate the past features to the current ones, we found more effective to simulate a Recurrent Neural Network with the past features.

Before going into more details, we show the architecture of $G_F$ in Fig. 4.1.2.

In figure 4.1.2 we can see the modified version of the network used in [19]. We do not provide specific details about the network architecture in terms of layers because these can be better appreciated looking at the code, but we provide the characteristics of the network that we consider most relevant and insightful to explain our work. $G_F$ composed of convolutional layers forming an encoder-decoder structure, with residual connections at its deepests features. It generates two outputs:

- **Next frame’s foreground** $\hat{I}_{F}^{t+1}$, generated by combining an attention mask $A \in \mathbb{R}^{1 \times H \times W}$ and a color mask $C \in \mathbb{R}^{3 \times H \times W}$ in the same manner as in figure 4.1.1. This combination is carried out as follows:

$$I_{F}^{t+1} = A \odot I_{F}^{t} + (1 - A) \odot C \quad (4.1)$$

Equation 4.1 can be understood as a transformation to the previous foreground $I_{F}^{t}$ guided by $A$ and changed by $C$. Intuitively, $A$ is a mask that focuses on which content of the previous foreground should remain and $C$ is the new content that should be added to the previous foreground to generate the next one.

- **Next frame’s mask** $\hat{M}^{t+1}$, generated from deep features before applying the recurrent layer. The reason behind this procedure is that we want to generate the mask taking into account more
Figure 4.1.2: $G_F$ architecture. Thick arrows represent convolutional layers (blue) or recurrent layers (yellow). Thin arrows denote adding values.

...information about the future (green features in figure 4.1.2 come from the previous foreground and the optical flow) than the past (yellow features in figure 4.1.2 contain both past and future information). At testing time, we binarize $\hat{M}^t$ with a threshold, which is an hyperparameter of our model.

The last thing to point out in the $G_F$ architecture is its recurrent layer. Although we took inspiration from [19] and [28], we obtained better results with a recurrent layer style. Formally:

- Let $f_t$ be the features coming from the previous foreground and Optical Flow.
- Let $h_{t-1}$ be the state coming from the previous time step through a transformation $r$. Formally: $h_t = r(g_t) = \tanh(\text{conv}(g_t))$.
- Then $g_t = h_{t-1} + f_t = \tanh(\text{conv}(g_{t-1})) + f_t$.

With this operation, $G_F$ process both past, present and future information.

Regarding the optical flows used in $G_F$, we use Flownet 2.0 [8] flows and we enter them to $G_F$ as in the work proposed by DeepMind [3], that is, truncating the values of OFs to $[-20, 20]$ and then normalizing to $[0, 1]$. 
Background Generator

We adopt the same structure used in $G_F$ for the function $G_B$ that approximates the next inpainted background $\hat{I}^{t+1}_B$. Obviously, we remove the convolutional layer generating the binary mask. The reason behind the use of this network architecture as a background inpainter is that in [11] this network was used to perform style transfer, which basically consists of changing the entire image to make it look like a given style. Therefore, we thought it would be a suitable and capable network to do the task of background inpainting. The architecture of $G_B$ is illustrated in figure 4.1.3.

Figure 4.1.3: $G_b$ architecture. Thick arrows represent convolutional layers (blue) or recurrent layers (yellow). Thin arrows denote adding values.

Note that we do not include $OF^{t+1}_t$ as in $G_F$ because at testing time we do not need future information in the background, since we are only interested in having it in the foreground, which is in charge of producing the masks, the elements of interest in this work. Moreover, we empirically observed that adding the Optical Flow as an input to $G_B$ lead to undesirable results because spurious elements of the foreground appeared in the background.

For reasons that will be discussed later in this document (§4.1.2), we do not input the masked background "as is". We explored different variations of the background such as adding uniform noise in the black area and mirroring the pixel values in the black area, reducing considerably the complexity of the inpainting task carried out by $G_B$. 

29
Foreground and Background Discriminators

As critics for both the fake foregrounds and backgrounds, we use PatchGAN, a network proposed by Isola et al. in [10], where the network design is specifically thought to discriminate image patches [9], achieving more realistic images than discriminating just by looking at them as a whole image. In addition, since the background must be compared in a patch-wise manner (as explained in sections §1.2 and §3.2.3), it becomes convenient for our work.

Implementation details

In this subsection we describe some minor implementation details that we consider relevant for our work. The image resolution we are working with is 224x416. The DAVIS2016 dataset is provided in two original resolutions 1080p (1920x1080) and 480p (640x480). We use a downsampled version of the original 480p resolution. Out of the possible 4 downsampled resolutions that kept the aspect ratio of the image, we opt to choose 224x416. This resolution is slightly larger than the one used in [19] (128x128). Nonetheless, we use 224x416 because in [19] they were dealing with faces, while in this work we deal with richer elements in terms of distribution, foreground and background, composed by objects that need to be well enough defined in the image. Also, this resolution is small enough to allow a good performance of the GAN.

All the training procedure was carried out taking into account the official DAVIS2016 split for training/validation sets, containing 30 training videos and 20 validation videos. Although the length of DAVIS2016 videos varies, having a range of videos that go from 27 to >100 frames, in this work we train our model with only the first $T = 8$ frames of each training video, predicting $T - 1$ masks. However, we provide validation results with different temporal horizons, such as $2T$ and $3T$ to validate our idea. However, further work will include results with the entire dataset. We set $T = 8$ as training temporal horizon based on two aspects, the memory we have available at training time and the trade-off between the performance of our auto-regressive networks and the flow of vanishing gradients through time.

In addition, our model is initialized with random weights, without requiring any trained weights for the backbone networks. Also, the four networks that compose our model are trained together, we do not train $G_F$ or $G_B$ first, we train them jointly expecting them to reach the desired behavior.

Main scheme

Finally, before moving on to the results section, we provide a diagram in 4.1.4 that depicts the entire training process of our model. No new information is provided in this diagram but it provides a complete picture of the approach. At testing time, only $G_F$ and optionally $G_B$ are used.

\[140x260, 175x325, 224x416, 280x520\]
4.1.2 Results

NOTE FOR THE READER: Since this work is solving the task of Video Object Segmentation, visualizing the masks in a PDF file is not the most appropriate format because video animation is lost. For this reason, we welcome the reader to visit the links provided to our GitHub repository where qualitative results of all models in form of "gif" files can be explored.

In this section, we provide qualitative and quantitative results for our baseline model. We begin explaining the background results, which were not obtained until we made some modifications to the input background. We only explain background results in the baseline section, because once we reached a good enough inpainting we focused on improving our models to have better segmentation results. After explaining the background results, we continue with foreground/segmentation results.

https://github.com/ppalaupuigdevall/VOS-ForeGAN
Qualitative

When we started working in this project, we entered the background to $G_B$ "as is", with a black area marking the foreground. However, this led to an undesired result, because when dealing with dark foregrounds, the black background area helped $G_F$ to generate the dark foreground. Consequently, $G_F$ did not generate masks properly. For this reason, we decided to mask the black area with uniform noise, as can be seen in 4.1.5.

Figure 4.1.5: (Left) Ground truth background. (Right) Input background masked with noise

With the addition of noise to the input background, we obtained the first expected results with $G_B$, some of them depicted in figure 4.1.6. This is due to the fact that when the background is masked with noise, it differs significantly from the foreground, and $G_F$ cannot use the content generated by $G_B$ to generate the foreground. Furthermore, at the same time $G_B$ does not use part of the foreground to decrease the $L_1$ loss.

The results shown in figure 4.1.6 are good enough to allow self-supervision in cases where the foreground object occupied a small or medium portion of the image. However, $G_B$ failed when inpainting larger objects, producing white inpainted areas. In order to solve this problem, instead of masking the background with noise, we propose our final approach for the input of $G_B$, adopted for all the models proposed in this thesis. We design a simple yet effective horizontal pixel mirroring, the simplest background inpainting we could think of when entering it to $G_B$. The goal of this technique is to reduce the similarity of the filled area with the foreground and at the same time increasing the similarity of this filled area with the background. In addition, we aim to reduce the complexity of the task performed by $G_B$, so it only has to modify the input in order to produce consistent background patches and reconstruct the next images starting from this easier background. Some results of this technique are depicted in 4.1.7. In general, the generated results of $G_B$ are not outstanding, but are good enough to achieve self-supervision, which is the main goal of this work.

Having detailed how $G_B$ works, we continue by showing qualitative results of $G_F$, the most important part of our model. In figure 4.1.8 we can see that the first training/validation results are promising and that our approach works as expected.

Moreover, we provide a training example in which we refine a mask with the SoS polynomial in 4.1.9. Finally, in figure 4.1.10 we analyze some training results where our model underperforms.
Figure 4.1.6: Training results of $G_B$ when input background masked with noise. First column denotes the input to $G_B$ (t=0). Second column shows the output of $G_B$ at time step t=1, $\hat{I}^1_B$. Third column represents $G_B$’s output at timestep (t=5). Notice how the model does not produce a perfect impainted background at t=1 because it is autoregressive, but the results at t=5 are fairly good.

Figure 4.1.7: Training results of $G_B$ when input background masked with lines. First row denotes the input to $G_B$ (t=0). Second row shows the input to $G_B$, which is $I^0_B$ with mirroring lines. Third row shows $G_B$’s output at timestep (t=7). Note the frame difference between time steps and how $G_B$ is able to approximate it not only producing similar textures, but also approximating the motion on the sequence as a consequence of the $\mathcal{L}_1$ loss.

Quantitative

We proceed to report quantitative metrics of our baseline model. To evaluate our model, we report results using the Jaccard index $J$, one of the official scores of the DAVIS2016 challenge. It is defined
Figure 4.1.8: Training/Validation baseline results of $G_F$. The first three rows show training segmentation results at time steps 1, 4, 7, proving that our baseline approach works. Fourth and fifth row show a validation result, evidencing that our model is able to generalize fairly good.

Figure 4.1.9: Results of refining the mask with SoS polynomial. Left image represents the mask shown in 4.1.8 (middle image of bmx seq.), containing spurious elements in the back of the boy as well as above the helmet. The middle image shows in purple filtered outliers by SoS polynomial, inliers are shown painted in blue. Right image shows the final filtered mask.
Figure 4.1.10: Training results where our model underperforms for t=1,4,7. The first row shows a case where our assumption that foreground has a different appearance than background is broken because it is translucent, so $G_F$ does not include it as part of its mask. The second example shows a big spurious spot in the right hand side of the middle image. Third and fourth row show an example where $G_F$ reconstructs part of the background due to the poor performance of $G_B$.

as the intersection over union of predicted mask (M) and ground truth mask (G):

$$J = \frac{|M \cap G|}{|M \cup G|}$$  \hspace{1cm} (4.2)

We provide results for our best models in both training and validation sets. We found our best model by doing a grid search of the mask’s binary threshold hyper-parameter and selecting the best epoch, note that although we do this in this work to report the best results, with the approach we propose we cannot actually do a hyper-parameter search, at least quantitatively, we would have to do it qualitatively because we do not have the ground truth masks.

We also evaluate this model with filtered masks, indicated with the suffix SoS. Table 4.1 summarizes the results of $J$ index evaluated at time horizon $T$ (temporal horizon in which the model was trained), $2T$ and $3T$. We can appreciate how $J$ decreases as we increase the temporal horizon. We can also appreciate how the filtering increases the $J$ index. However, SoS polynomial cannot help when the model does not even generate the entire mask as in the mallard example in figure 4.1.10 or in validation results of table 4.1.
Table 4.1: Baseline model results for binary threshold 75.

<table>
<thead>
<tr>
<th>Model</th>
<th>T train</th>
<th>T val</th>
<th>2T train</th>
<th>2T val</th>
<th>3T train</th>
<th>3T val</th>
<th>epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>42.1</td>
<td>38.5</td>
<td>35.8</td>
<td>30.6</td>
<td>32.3</td>
<td>28.5</td>
<td>4900</td>
</tr>
<tr>
<td>Baseline-SoS</td>
<td>49.8</td>
<td>38.7</td>
<td>39.4</td>
<td>28.8</td>
<td>33.9</td>
<td>24</td>
<td>4900</td>
</tr>
</tbody>
</table>

Analysis

In this section we derive conclusions of our baseline model. We list them below:

- Although not entering a mask neither to $G_F$ nor to $D_F$, our model produces segmentation masks even though it is not explicitly trained to generate them, thanks to the way it is designed and to self-supervision, in spite of being trained only with one mask.

- Masks contain too many spurious elements. In the stroller and mallard examples in figure 4.1.10 we can appreciate this problem. However, SoS polynomial is able to remove some of them, improving the quantitative results as seen in table 4.1.

- In figures 4.1.10 we can see that in the swing example, the wood part of the swing is also included in the foreground when it should not be. This is due to the fact that $G_B$ is not representing good enough the swing’s wood so $G_F$ generates it. To solve this problem, we apply a modification to our model to make $D_F$ discriminate not only estimated foregrounds $\hat{I}_F$, but also estimated masks $\hat{M}$. With this change we expect the masks to look more realistic. This change leads to our second prototype, named Dmask. We follow the same procedure carried out when discriminating real foregrounds $I^0_F$ and apply a geometric transformation to the first mask $M^0$ before feeding it to $D_F$. We implemented this change slightly modifying $D_F$, by adding a last layer in addition to the backbone which already had. Therefore, now $D_F$ has a convolutional backbone followed by two independent convolutional layers, one for discriminating foregrounds and other for discriminating masks.

4.2 Dmask

In this section we display and study some results of Dmask model. We do not provide a list of results as long as the shown with baseline model, but we only show relevant results. Moreover, since we explained the changes with respect to the baseline model in the previous section Analysis, we go directly to the results, qualitatively and quantitatively.

4.2.1 Results

With the changes applied to our baseline model, Dmask obtains better results in training, which can be appreciated in table 4.2 but almost the same in validation than the baseline model. However, it
is worth pointing out some differences in the results, shown in the figure 4.2.1 right below.

Figure 4.2.1: Training/Validation results where our model $Dmask$ performs fairly good. The first row shows a case in which despite having spurious elements, the segmented results are acceptable given that are in $t=1,5,11$. In second row we show a challenging video where $G_F$ produces part of the moving sticks because $G_B$’s inpainting was poor. However, $G_F$ produces a good segmentation mask. Third row shows the mallard example previously examined with baseline model, now we can see that the mask embraces more area of the mallard, which is good. However, the spurious elements are not removed. Finally, we show a good validation result.

**Analysis**

We list a few conclusions derived from the experiments obtained with $Dmask$:

1. We see that the problem of spurious elements in mask still persists in spite of the changes made to the baseline. These spurious elements are the consequence of the poor performance of $G_B$, as we explained before, and due to the fact that even though $G_F$ generated part of the background, $D_F$ cannot discriminate that as part of foreground.

2. In table 4.2 we can see that we improved the baseline model in training with the slight modification we introduced.
Table 4.2: Mean Jaccard index for Dmask and baseline model at binary threshold 75.

<table>
<thead>
<tr>
<th>Model</th>
<th>T</th>
<th>2T</th>
<th>3T</th>
<th>epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train</td>
<td>val</td>
<td>train</td>
<td>val</td>
</tr>
<tr>
<td>Baseline</td>
<td>42.1</td>
<td>38.5</td>
<td>35.8</td>
<td>30.6</td>
</tr>
<tr>
<td>Baseline-SoS</td>
<td>49.8</td>
<td>38.7</td>
<td>39.4</td>
<td>28.8</td>
</tr>
<tr>
<td>Dmask</td>
<td>45.2</td>
<td>38.5</td>
<td>39.2</td>
<td>30.6</td>
</tr>
<tr>
<td>Dmask-SoS</td>
<td>51.0</td>
<td>37.01</td>
<td>44.2</td>
<td>27.28</td>
</tr>
</tbody>
</table>

3. We can clearly observe that even though we are dealing with video sequences, training takes a long time (epochs 4900, 5400) to achieve good performance. This detail will be thoroughly explained in the Conclusions section.

4. We see that although qualitative results look fairly good, quantitative results seem rather low. To analyze what occurs, we display $J$ index as a function of time in relevant videos to conclude what can be improved in our Dmask model. In figure 4.2.2 we can see that the average shape of the $J$ index is similar to a flipped J. This fact is due to the auto-regressive nature of our model. Let us explain this fact precisely. When dealing with auto-regressive models, we need acceptable inputs to obtain accurate outputs. Therefore, a poor generation in the middle of the sequence can lead to a bad result in the output. However, when training a neural network the gradients flow backwards. If we extend too much the temporal horizon $T$, in the middle of the sequence we can end up having very bad results which have low influence from the input and low influence of gradients.

Figure 4.2.2: Jaccard index as a function of time. A shape of a flipped J can be appreciated due to the trade-off between forward and backward propagation.
In order to solve this problem, we propose a modification and design our last model \textit{Dmask-prop} because it auto-regressively propagates the first mask from the input. This model is exactly the same as \textit{Dmask} but we introduce the first frame mask as an input to \( G_F \), which \textbf{auto-regressively creates the next \( T - 1 \) masks}. With this change, we expect to flatten the curve shown in \ref{4.2.2}, expecting the input mask to propagate better until larger time steps than before.

\subsection{4.3 Dmask-prop}

We proceed to explain the results of our last model \textit{Dmask-prop}.

\subsubsection{4.3.1 Results}

Achieving the best results so far, this is the best model that we obtained in the course of this thesis. In table \ref{4.3} we show quantitative results comparing them across models. We provide the link to ".gif" results where we show qualitative comparisons of our three models. In figure \ref{4.3.1} we study some good qualitative results where we can see that \textit{Dmask-prop} model propagates masks guided by the optical flow, which was the main purpose of this work. In addition, to demonstrate the cooperation of \( G_B \) and \( G_F \), we provide an example where our model performs at its best in figure \ref{4.3.2}. However, there are some cases in which our model fails, which we show in figure \ref{4.3.3}.

Table 4.3: Mean Jacard index for \textit{Dmask-prop} in comparison with our other models at binary threshold 75. A clear improvement can be seen with respect to previous models.

<table>
<thead>
<tr>
<th>Model</th>
<th>train</th>
<th>val</th>
<th>train</th>
<th>val</th>
<th>train</th>
<th>val</th>
<th>epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>42.1</td>
<td>38.5</td>
<td>35.8</td>
<td>30.6</td>
<td>32.3</td>
<td>28.5</td>
<td>4900</td>
</tr>
<tr>
<td>Baseline-SoS</td>
<td>49.8</td>
<td>38.7</td>
<td>39.4</td>
<td>28.8</td>
<td>33.9</td>
<td>24</td>
<td>4900</td>
</tr>
<tr>
<td>Dmask</td>
<td>45.2</td>
<td>38.5</td>
<td>39.2</td>
<td>30.6</td>
<td>36.1</td>
<td>28.5</td>
<td>5400</td>
</tr>
<tr>
<td>Dmask-SoS</td>
<td>51.1</td>
<td>37.01</td>
<td>44.2</td>
<td>27.28</td>
<td>41.6</td>
<td>24.5</td>
<td>5400</td>
</tr>
<tr>
<td>\textbf{Dmask-prop}</td>
<td>\textbf{67.4}</td>
<td>\textbf{60.5}</td>
<td>\textbf{59.3}</td>
<td>\textbf{52.8}</td>
<td>\textbf{54.9}</td>
<td>\textbf{48.2}</td>
<td>\textbf{2000}</td>
</tr>
<tr>
<td>\textbf{Dmask-prop}</td>
<td>\textbf{65.4}</td>
<td>\textbf{57.1}</td>
<td>\textbf{57.5}</td>
<td>\textbf{48.7}</td>
<td>\textbf{53.3}</td>
<td>\textbf{45.5}</td>
<td>\textbf{3900}</td>
</tr>
</tbody>
</table>

\url{https://github.com/ppalaupuigdevall/VOS-ForeGAN}
Figure 4.3.1: Qualitative training/validation results where \textit{Dmask-prop} performs reasonably better than previous models. All examples show timesteps $t=1,5,10$. First three rows show training results while the last two show validation examples, achieving the best results so far in both cases. In first and second row, we can appreciate that although the segmentation is not the best possible, masks do not contain the aforementioned spurious elements present in \textit{baseline} and \textit{Dmask}. In validation examples of rows four and five, we can appreciate that our model is capable of generalizing fairly well.
Figure 4.3.2: Results of $G_F$ and $G_B$ cooperation. First row exhibits segmentation masks at times $t=1,5,10$. We also show the impainted background for this case to prove that when background is static and simple enough, $G_B$ can estimate it properly despite having entered a "corrupted" version of it. In second row, image on the left corresponds to original background, middle to input background and right to impainted background at time $t=10$.

Figure 4.3.3: Qualitative training/validation results where our model $Dmask-prop$ underperforms. All examples show timesteps $t=1,5,10$. First row shows the already studied training swing example, which our model does not properly segment, in spite of all improvements. Second row shows a validation case where our model fails, possibly because of the resemblance of foreground with background and poor optical flow.
Chapter 5

Conclusions and Future Work

In this section we proceed to list conclusions and future work for this thesis. We provide a detailed list of them because we truly think this work has potential enough to be continued as a line of research.

- The proof of concept of the proposed model worked. While it did not achieve state of the art performance it shows that our approach is valid. Taking into account that when training generative adversarial networks good results are not always granted, we consider this to be quite a success.

- The modifications applied to baseline and Dmask models worked, showing that the intuition behind each modification was correct. Thus, we obtained a model that obtained best results, Dmask-prop.

- However, we conclude that the approach of generating foreground, background and mask all together is too ambitious, especially the background. As we have seen throughout this work, generating the background is too complex for our generator, sometimes leading to poor mask results. Therefore, we believe that removing the background branch of our approach will lead to better results. However, we still think that generating the foreground jointly with the mask is interesting, but instead of composing the next frame with the generated background, it should be done partitioning the ground truth next frame with the estimated mask.

- Our models take too long to train. We hypothesize that this is due to two facts:
  1. The auto-regressive nature of the models.
  2. The way in which the next foreground is approximated might not be the optimal one for videos where the object changes its position rapidly. The way of modifying the previous image \((I_{F+1}^t = A \otimes I_F^t + (1 - A) \otimes C)\) worked in GANimation [19] because they only had to slightly modify the image content color-wise, but not the position of the face. Although our model’s \(G_F\) eventually generates the next foregrounds correctly, it takes too long. Therefore, we propose a modification of \(G_F\) which we truly think is more suitable for the cases where the objects in the video change rapidly based on a perspective
transformation $H$ applied to the features. We provide a sketch for the new proposed $G_F$ in figure 5.0.1.

Figure 5.0.1: Proposed new $G_F$ architecture. The reasoning behind this architecture is that $H^t$ estimates the new position of the foreground in the next frame by a perspective transform $H$, applied to the features, while we still modify the appearance of the new foreground through the combination with $A$ and $C$. Moreover, dynamic constraints could be applied to coefficients of $H$, requiring them to have low dynamics.

- An extra classification loss could be added to the model to add another type of self-supervision. This loss would consist in, provided we had all the generated masks of one sequence, a network should be capable of estimate from which video this foreground came.

- Finally, taking into consideration the high quality of the masks generated by $D_{mask-prop}$ at $t=1$, our self-supervision technique could be applied in a sliding window fashion to increase temporal horizon $T$.

This is the end of this thesis, for more details, please visit [1]

Bibliography


[17] Edouard Pauwels and Jean B. Lasserre. “Sorting out typicality with the inverse moment matrix SOS polynomial”. In: NIPS. 2016.


Appendix

Appendix A

Mathematical derivation of the optimal value of $D(x)$. Given the GAN loss function:

$$L(G, D) = \int_x \left( p_r(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \right) dx$$

(5.1)

Since we are interested in the optimal value of $D$, we can rename some items in the equation above for readability purposes:

$$\tilde{x} = D(x),\ A = p_r(x),\ B = p_g(x)$$

(5.2)

Then, ignoring the integral since $x$ is samples over all the possible values, we can rewrite $L(G, D)$ as:

$$f(\tilde{x}) = A \log \tilde{x} + B \log(1 - \tilde{x})$$

(5.3)

Then we find the optimal value for $\tilde{x}$:

$$\frac{df(\tilde{x})}{d\tilde{x}} = \frac{A}{\ln 10} \frac{1}{\tilde{x}} - \frac{B}{\ln 10} \frac{1}{1 - \tilde{x}}$$

(5.4)

$$= \frac{1}{\ln 10} \left( \frac{A}{\tilde{x}} - \frac{B}{1 - \tilde{x}} \right)$$

(5.5)

$$= \frac{1}{\ln 10} \frac{A - (A + B)\tilde{x}}{\tilde{x}(1 - \tilde{x})}$$

(5.6)

Then, the best value of $\tilde{x}$ is obtained by $\frac{df(\tilde{x})}{d\tilde{x}} = 0$, which leads us to the optimal value of $D$:

$$D^*(x) = \tilde{x}^* = \frac{A}{A + B} = \frac{p_r(x)}{p_r(x) + p_g(x)} \in [0, 1]$$

(5.7)

Appendix B

Consider 2 discrete probability distributions $p$ and $q$, with 6 samples each, that have the following distributions:

The Wasserstein distance quantifies how similar 2 distributions are by measuring how many mass of one distribution should be moved to make one distribution look like the other.
Suppose we would like to make $p$ look like $q$. Let us imagine that $p$ is composed of 6 boxes and we have to move the blue boxes to match the distribution of the red ones. There exist many combinations of how we should move the boxes in order to satisfy that, which will be referred as transport plans or $\gamma_i$.

In this example 2 transport plans are studied, $\gamma_1$ (g1 in 5.0.3) and $\gamma_2$ (g2 in 5.0.3). In the following figure, we can appreciate how this transport plans are computed, assuming that each "box" weight is equal to 1. Therefore, we compute the cost as the absolute value of the travelling distance between positions $x$ and $y$. 
Figure 5.0.3: Example of 2 transport plans $\gamma_1$ and $\gamma_2$ for 2 discrete probability distributions. Although the Wasserstein distance results in the same value for both plans, the transport plans differ. We notice how both constraints mentioned in §2.1.3 are fulfilled, ending with $p$ if we marginalize across the $y$ coordinate of $\gamma(x, y)$ and viceversa. We can appreciate this fact observing the "boxes" at the sides of the grid.