Improving DeFMO With Learned Losses

Harish Rajagopal
Computer Science MSc
ETH Zürich
20-946-349

Denys Rozumnyi
Co-Supervisor
Department of Computer Science
ETH Zürich

Prof. Dr. Marc Pollefeys
Supervisor
Department of Computer Science
ETH Zürich

Abstract

DeFMO is an algorithm that aims to remove motion blur in images using a neural network to deblur individual frames as well as generate 'sub-frames', as if captured by a high-speed camera (i.e. temporal super-resolution). This project extends DeFMO using GANs. Our approach overcomes limitations within the loss formulation of DeFMO, more accurately captures the 'quality' of the generated outputs, and thus improves results. We also introduce a novel loss function for tackling temporal consistency among generated sub-frames. Further, we optimize the codebase to be more memory-efficient, and improve readability and maintainability.

1 Introduction

Motion blur is a challenging problem in computer vision. It occurs when moving objects are captured with long exposures. This leads to a loss of information in images, and thus algorithms for deblurring are pursued. Deblurring motion blur due to fast moving objects [28] (FMOs) is a challenging subset of this task, and it is necessary for tracking FMOs such as in ball-based sports, free-falling objects, high-speed cars, etc.

Various deep learning approaches have been proposed to tackle FMO blur. Rozumnyi et al. [30] proposed an algorithm known as DeFMO, that uses a convolutional encoder-decoder architecture to deblur individual images by generating sub-frames that depict the motion of the FMO. The model is trained in a supervised learning setting with a wide variety of losses for good performance.

While these losses lead to good performance on various datasets, they might not be optimal for the best performance. Here, we aim to replace these hand-engineered losses with neural networks that can learn a loss function for training the base model. For this, we use Generative Adversarial Networks [6] (GANs).

The **contributions** of this project are:

- Improvement of the DeFMO model's outputs using GANs.
- A novel learned loss function for temporal frame consistency.
- A generalized approach over DeFMO for any task that involves video super-resolution.
- Optimizations for more efficient memory management during training.
- Refactoring and code quality enforcement over the DeFMO codebase.

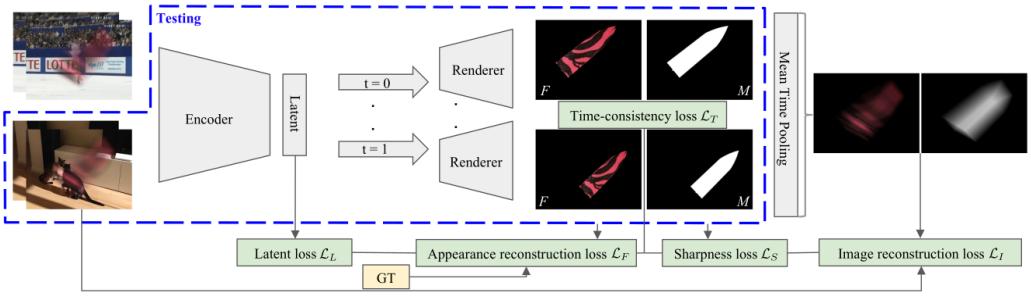

Figure 1: Architecture of DeFMO.

2 Baseline

The baseline model is DeFMO. Here, the input consists of an RGB image $I \in \mathbb{R}^{H \times W \times 3}$ of the foreground blurred FMO with height H and width W, and another RGB Image $B \in \mathbb{R}^{H \times W \times 3}$ of the background. The desired output is a deblurred rendering of the FMO for N sub-frames. The t^{th} sub-frame is an RGBA image $R_t \in \mathbb{R}^{H \times W \times 4}$ consisting of an RGB part $F_t \in \mathbb{R}^{H \times W \times 3}$ for the sharp appearance of the FMO, and an alpha matting mask $M_t \in \mathbb{R}^{H \times W}$ which segments the FMO from the background.

Since the motion blur in the input image is due to movement of the FMO with long exposure time, we should be able to reconstruct the input image using the output sub-frames. This can be achieved by mimicking how a camera would capture the FMO. For this, Kotera and Šroubek [17]; Kotera et al. [13] introduce the blurring and matting (blatting) equation:

$$I = H * F + (1 - H * M)B \tag{1}$$

Here, F is the sharp foreground appearance of the object, M is the segmentation mask, and H is the blur kernel. H identifies the object's trajectory such that $||H|| = 1$.

DeFMO approximates this with the following model:

$$I_{t_0:t_1} = \int_{t_0}^{t_1} F_t M_t dt + \left(1 - \int_{t_0}^{t_1} M_t dt\right) B \tag{2}$$

Here, the trajectory blur kernels H_t aren't disentangled, because they are just Dirac deltas.

Architecture DeFMO uses an encoder to encode I and B into a latent vector $X \in \mathbb{R}^K$. The encoder is a CNN that takes both I and B concatenated along the channel axis. A renderer takes X and time indices $t_0, t_1, \ldots, t_{N-1}$ to output the N sub-frames. This is also a CNN that uses sub-pixel convolutions [31], also known as pixel shuffles, for upsampling. An illustration of the architecture is shown in Figure 1.

2.1 Training losses

DeFMO uses a wide variety of losses to optimize the model. These consist of the output reconstruction loss ($\mathcal{L}_F$), the input reconstruction loss (based on Equation 2) ($\mathcal{L}_I$), and losses to constraint the output to have smooth movements in time ($\mathcal{L}_T$), sharp objects ($\mathcal{L}_S$), and invariance to the background ($\mathcal{L}_L$). Therefore, the total loss is a weighted sum of these losses:

$$\mathcal{L} = \mathcal{L}_F + \alpha_I \mathcal{L}_I + \alpha_T \mathcal{L}_T + \alpha_S \mathcal{L}_S + \alpha_L \mathcal{L}_L \tag{3}$$

Appearance reconstruction loss $\mathcal{L}_F$ is the supervised loss for the output sub-frames. However, since the model has to generate movement based on a single image as the input, the movement can be in either the same direction as the ground-truth or the opposite. Therefore, the supervised loss $\mathcal{L}_R$ is calculated for both directions, and $\mathcal{L}_F$ is the minimum of these:

$$\mathcal{L}_F(\{R_t, \tilde{R}_t\}_1^N) = \frac{1}{N} \min\left(\sum_{t=1}^{N} \mathcal{L}_R(R_t, \tilde{R}_t)dt, \sum_{t=1}^{N} \mathcal{L}_R(R_t, \tilde{R}_{1-t})dt\right) \tag{4}$$

$\mathcal{L}_R$ for the output rendering R_t and the ground-truth $\tilde{R}_t$ at time t is calculated as:

$$\begin{aligned}\mathcal{L}_R(R_t = (F_t, M_t), \tilde{R}_t = (\tilde{F}_t, \tilde{M}_t)) = \mathcal{L}_1(M_t, \tilde{M}_t, \tilde{M}_t > 0) + \mathcal{L}_1(M_t, \tilde{M}_t, \tilde{M}_t = 0) \\ + \mathcal{L}_1(F_t M_t, \tilde{F}_t \tilde{M}_t, \tilde{M}_t > 0)\end{aligned} \tag{5}$$

where $\mathcal{L}_1(A, B, M)$ is the $\mathcal{L}_1$ loss for images A and B conditioned on the mask M:

$$\mathcal{L}_1(A, B, M) = \frac{\sum_{i,j} ||A_{i,j} - B_{i,j}||_1 M_{i,j}}{\sum_{i,j} M_{i,j}} \tag{6}$$

In Equation 5, the first two terms are for comparing the masks M_t and $\tilde{M}_t$ for the foreground and background (as given by the ground truth) separately. The separation is to prevent imbalance due to smaller objects in a large background. The third term is for the object, but evaluated only in the foreground to disentangle the mask and the object channels.

Image reconstruction loss $\mathcal{L}_I$ is just an application of the blatting equation in Equation 2. Thus, this loss is just the $\mathcal{L}_1$ loss between the blatted image and the input image I:

$$\mathcal{L}_I(\{R_t\}_1^N, I) = \left|\left| I, \frac{1}{N}\sum_{t=1}^{N} F_t M_t + \left(1 - \frac{1}{N}\sum_{t=1}^{N} M_t\right) B \right|\right|_1 \tag{7}$$

Time-consistency loss $\mathcal{L}_T$ captures the temporal smoothness of the N output sub-frames. Informally, we want the sub-frames for nearby t to be similar to each other. This similarity is formalized by the maximum value of the normalized cross-correlation over all pixels. Thus, the loss is:

$$\mathcal{L}_T(\{R_t\}_1^N) = 1 - \frac{1}{N}\sum_{t=1}^{N} \text{maxncc}(R_t, R_{t+1}) \tag{8}$$

Sharpness loss Here, $\mathcal{L}_S$ only focuses on the sharpness of the mask, and not the object contents itself. The mask is expected to be binary in most of the image, except at the boundaries. However, we simplify this constraint to just enforcing binary values. This is expressed by the per-pixel binary entropy $\mathcal{H}_2$ averaged over all $H \times W$ pixels:

$$L_S(\{M_t\}_1^N) = \frac{1}{NHW}\sum_{t=1}^{N}\sum_{i=1}^{H}\sum_{j=1}^{W} \mathcal{H}_2([M_t]_{i,j}) \tag{9}$$

Latent learning loss $\mathcal{L}_L$ aims to constrain the model such that blurred images of the same FMO moving along the same trajectory but in front of different backgrounds generate the output. This can be achieved by modelling the latent space such that these images have the same latent representation. Thus, DeFMO trains in pairs of such images, and computes $\mathcal{L}_L$ between their latent vectors $X_1, X_2 \in \mathbb{R}^K$ as:

$$\mathcal{L}_L(X_1, X_2) = \frac{||X_1 - X_2||_1}{K} \tag{10}$$

Note that not all losses used here are unbiased, especially the time-consistency and sharpness losses. The ground-truth renderings won't necessarily be the minimum of these losses individually, since the time-consistency loss prefers static sub-frames, and the sharpness loss prefers binary masks everywhere, even on boundaries. This brings us to our proposed approach, where we want to overcome these limitations and get more general losses.

3 Proposed approach

3.1 GANs

We use GANs to replace the sharpness loss. In the GAN framework, there are two models, a generator and a discriminator, that compete against each other in a zero-sum game. The goal is to train the generator to generate data when given a dataset, in an unsupervised learning setting. The discriminator has to distinguish between the generator's outputs and the ground-truth from the dataset. At convergence, we expect the generator to be able to successfully fool the discriminator by learning to generate data that is indistinguishable from the dataset.

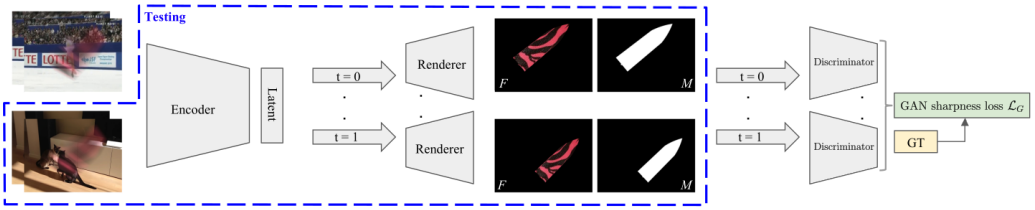

Figure 2: Architecture for the GAN-based sharpness loss. The discriminator takes as input one complete sub-frame and predicts whether each $P \times P$ patch is valid or not.

Patch discriminators Isola et al. [10] extend GANs to also work in supervised learning settings, by introducing the concept of a 'patch discriminator'. As opposed to traditional GANs, where the discriminator outputs predictions for whole images, the patch discriminator outputs predictions for each $P \times P$ patch of the input image. This complements the supervised learning objective to generate more realistic output images. As the authors point out, the supervised loss will capture the low-frequency features in the output data, while the discriminator will capture the high-frequency features.

3.2 Sharpness loss

Since we already have a supervised loss for the outputs ($\mathcal{L}_F$), we use a patch discriminator for the sharpness loss. The main model is now viewed as a generator. The discriminator D_S is a CNN that takes as input an RGBA image $R \in \mathbb{R}^{H \times W \times 4}$ with the same structure and representation as the outputs of the generator. It will output a scalar prediction for each $P \times P$ patch in the image, which are averaged to get the output for the entire image. The new architecture is shown in Figure 2.

We use the Wasserstein loss [1] to train the GAN. This loss requires D_S to be a 1-Lipschitz function. We use spectral normalization [22] over all of its weights to enforce this constraint. Thus, the new GAN-powered sharpness loss $\mathcal{L}_G$ is:

$$\mathcal{L}_G(\{R_t, \tilde{R}_t\}_1^N) = \frac{1}{N} \sum_{t=1}^{N} \left(\mathbb{E}_{\tilde{R}_t}[D_S(\tilde{R}_t)] - \mathbb{E}_{R_t}[D_S(R_t)] \right) \tag{11}$$

Here, the expectation is approximated by a mean over a batch of inputs. The objective for D_S is to maximize this loss, whereas the generator has to minimize this.

Intuitively, D_S will focus on the high-frequency features of each sub-frame R_t — both the foreground F_t and the mask M_t. This means that both the texture of the foreground object and the sharpness of the boundaries must be optimal. Since D_S can learn a highly non-linear function, $\mathcal{L}_G$ improves over $\mathcal{L}_S$ in the following ways:

- It incorporates foreground details.
- It can identify boundaries and thus relax the constraint of binary mask values near them.
- It can enforce spatial continuity of mask values — entropy has no spatial knowledge.
- It is general enough to be used in other tasks involving fine-grained image details.

3.3 Time-consistency loss

Here, we modify the objective for the discriminator to distinguish between consecutive frames and random frames. The input to this discriminator D_T is a pair of RGBA images $R_1, R_2 \in \mathbb{R}^{H \times W \times 4}$ with each RGBA image having the same structure and representation as the outputs of the generator. D_T is a CNN that will be given all possible pairs of output sub-frames, and it has to identify if R_2 immediately follows R_1.

Therefore, if either all sub-frames are identical or lack temporal smoothness, then the discriminator cannot distinguish them. The only optimum is to have proper temporal smoothness. Thus, this loss improves over the previous time-consistency loss since it enforces smooth movement, whereas

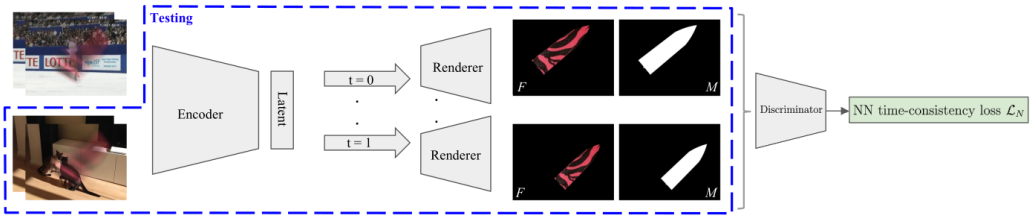

Figure 3: Architecture for the NN-based time-consistency loss. The discriminator takes as input each possible pair of sub-frames separately and predicts whether the second one follows the first.

the previous loss encourages static sub-frames. Further, unlike the GAN sharpness loss, this is a self-supervised setup that doesn't need any ground-truth.

However, unlike the sharpness loss, we cannot use the GAN framework here. This is because we want **both** the discriminator and the generator to optimize this. Since the two networks need to help each other, this is not a GAN. Therefore, we simply model this loss using binary cross-entropy:

$$\mathcal{L}_N = -\frac{1}{N^2}\sum_{t=1}^{N}\sum_{u=1}^{N}\left(\mathbb{1}[u = t+1]\log D_T(R_t, R_u) + \mathbb{1}[u \neq t+1]\log(1 - D_T(R_t, R_u))\right) \quad (12)$$

This is a novel loss function that can be thought of as a neural-network-based learned loss. The architecture for this setup is shown in Figure 3.

4 Experiments

4.1 Dataset

The models are trained on the synthetic dataset introduced by Rozumnyi et al. [30], due to lack of a large and diverse real-world annotated dataset with FMOs. Each synthetic image consists of an *object*, a 6D *trajectory*, and a *background* frame. *Objects* are sampled from 3D models of the 50 largest classes in the ShapeNet [2] dataset. *Trajectories* are sampled uniformly as a combination of 3D linear translations and 3D rotations. *Backgrounds* are sampled from the VOT [18] sequences for training, and from Sports-1M [11] for validation.

For evaluation, the models are evaluated on several datasets using the open source FMO deblurring benchmark introduced by Rozumnyi et al. [30] using ideas taken from Rozumnyi et al. [28]. It includes the Falling Objects [16], TbD-3D [29], and TbD [14] datasets.

4.2 Architecture

The encoder and both discriminators have the ResNet-18 [8] architecture (up to and excluding the global average pooling layer), and their weights are initialized with a model pre-trained on ImageNet [5]. Thus, the latent space is of 512 dimensions (due to ResNet-18). The parameters of the first layer are adapted to the no. of channels in the inputs by duplicating the pre-trained weights.

The discriminators use spectral normalization [22] to stabilize training, and to enforce the 1-Lipschitz constraint for the Wasserstein loss [1] in the GAN sharpness loss. D_S has a convolution layer without any non-linearity after the ResNet-18 backbone, after which global average pooling combines the patch predictions into a single scalar. Here, since ResNet-18 downsamples the input images by 32, therefore the patch size will be $(\lceil H/32 \rceil, \lceil W/32 \rceil)$. D_T keeps the global average pooling layer after the ResNet-18 backbone, and adds a fully-connected layer with sigmoid activation on top of the pooling layer with a scalar output.

The renderer is a standard CNN with batch normalization [9] and ReLU [23]. Additionally, it uses the ResNet bottleneck along with pixel shuffling [31] for upsampling. The exact architecture is shown in Table 1.

Table 1: Architecture of the renderer. Parameter values correspond to those used in PyTorch v1.8 for the corresponding layers.

Layer	Channels	Kernel size	Stride
Conv2d	1024	3	1
BatchNorm2d	1024		
ReLU			
ResNet bottleneck	256		
PixelShuffle			2
ResNet bottleneck	64		
PixelShuffle			2
ResNet bottleneck	16		
PixelShuffle			2
Conv2d	16	3	1
PixelShuffle			2
Conv2d	4	3	1
ReLU			
Conv2d	4	3	1

Table 2: Hyper-parameter values used for all approaches. Any missing values are assumed to have the defaults as in PyTorch v1.8. Hyphen denote that values aren't applicable.

Name	Approach	
	DeFMO	Ours
Epochs	7	4
Batch size	24	9
Input resolution	128×128	128×128
Number of sub-frames N	24	24
Learning rate (main model)	0.001	0.001
Learning rate (D_S)	-	0.00001
Learning rate (D_T)	-	0.00005
Learning rate decay rate (all schedulers)	0.5	0.5
Learning rate decay step size (all schedulers)	10	10
D_S steps per main model step	-	1
D_T steps per main model step	-	2
Image reconstruction loss weight α_I	1.0	1.0
Time-consistency loss weight α_T	1.0	1.0
Sharpness loss weight α_S	1.0	1.0
Latent learning loss weight α_L	1.0	1.0
GAN sharpness loss weight α_G	-	1.0
NN-based time-consistency loss weight α_N	-	0.05

4.3 Setup

The models are trained using the Adam [12] optimizer. There is one optimizer for the main model (encoder and renderer), one for D_S and one for D_T. Optimization steps for D_S, D_T, and the main model are taken alternately. Step-wise learning rate decay is applied for all optimizers.

The code was written in Python using the PyTorch [27] library. It was run on a cluster that uses CUDA [24] on three NVIDIA GeForce® GPUs with 11 GiB memory for accelerating PyTorch.

Hyper-parameters were manually chosen — no hyper-parameter search procedure was used. Their values are shown in Table 2.

Table 3: Evaluation on the FMO deblurring benchmark. The datasets are sorted top-to-bottom by decreasing difficulty: arbitrary shaped and textured [16], mostly spherical but textured [29], and mostly spherical and uniformly coloured objects [14].

Dataset	Metric	Approach	
		DeFMO	Ours
Falling Objects	TIoU ↑	0.000	0.000
	PSNR ↑	**22.794**	22.064
	SSIM ↑	**0.608**	0.604
TbD-3D	TIoU ↑	0.000	0.000
	PSNR ↑	20.156	**21.256**
	SSIM ↑	0.533	0.533
TbD	TIoU ↑	0.000	0.000
	PSNR ↑	20.262	**20.947**
	SSIM ↑	0.471	**0.511**

4.4 Memory optimizations

A disadvantage of our approach is that the addition of discriminators increases compute and memory usage significantly. Therefore, we adopt a few techniques to reduce the GPU memory impact.

Gradient checkpointing This technique trades computation for lower memory usage. Chen et al. [3] introduce gradient checkpointing, an algorithm that can train an n layer network with $\mathcal{O}(\sqrt{(n)})$ memory cost. Simply put, it replaces intermediate activations by 'checkpoints'. When a checkpoint is later needed during back-propagation, the activations are recomputed. Thus, the memory savings come at the cost of extra forward passes per mini-batch.

Mixed precision training This technique trades accuracy for faster computation and lower memory usage. Training networks using 32-bit single-precision floating points is typically more compute- and memory-intensive than 16-bit half-precision floats. However, using half-precision may lead to training instability due to underflow in gradients. Micikevicius et al. [21] introduce mixed-precision training, a technique that uses half-precision for all computations that aren't sensitive to precision, and single-precision otherwise. To avoid underflow in gradients, it also scales losses before calculating gradients, and unscales the gradients when applying them.

5 Results

5.1 Benchmark results

The FMO deblurring benchmark evaluates the models using peak signal-to-noise ratio (PSNR), structural similarity index measure [32] (SSIM), and intersection over union averaged over the trajectory [15] (TIoU). Here, both methods use a median of the previous five frames as the background. The results for this benchmark are shown in Table 3. Image samples on the Falling Objects, TbD-3D and TbD datasets are shown in figures 4, 5 and 6 respectively.

As we can see, our approach performs better than DeFMO in the TbD-3D and TbD datasets. In the Falling Objects dataset, it falls short of DeFMO. This is presumably because of insufficient training for our model (4 epochs for ours vs 7 epochs for DeFMO, as shown in Table 2), although neither model is trained till convergence (due to lack of time). Both models get 0.000 TIoU, further reinforcing the previous remark.

Further, due to memory limitations, the maximum batch size achievable for our approach was 9, lower than that for DeFMO (which is 24). Since we build upon ResNet-18, which uses batch normalization, lower batch sizes impact accuracy. Hence, with more time and more GPU memory, our approach should outperform DeFMO.

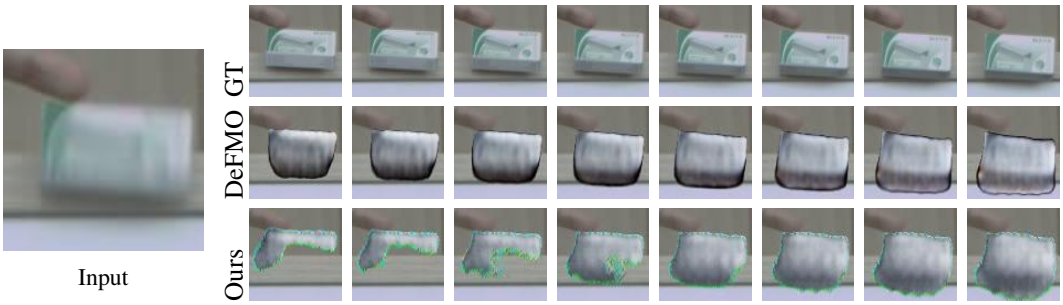

Figure 4: Sample outputs on the Falling Objects dataset.

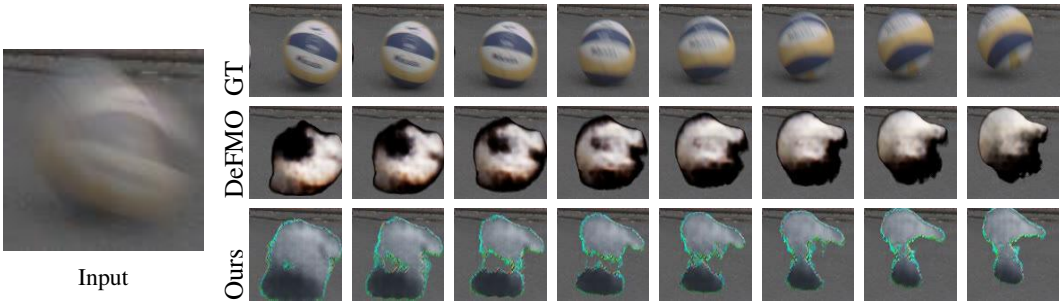

Figure 5: Sample outputs on the TbD-3D dataset.

5.2 Memory optimizations

The memory savings due to techniques in Section 4.4 are shown in Table 4. Memory usage is calculated using a custom wrapper around the NVIDIA System Management Interface [25] (`nvidia-smi`) tool, which is included with NVIDIA's GPU drivers. Here, the training is stopped after two gradient steps. Also, all hyper-parameters are the same as in Table 2, except for batch size, which is 5.

Evidently, our model consumes much more memory than DeFMO. However, the gap can be reduced considerably by both gradient checkpointing and mixed precision. While both significantly reduce memory usage, mixed precision gives a larger improvement.

5.3 Code quality

Code quality is assessed using Cyclomatic Complexity [20] (CC), Halstead [7] difficulty and delivered bugs, and the Maintainability Index [26; 4] (MI). These were calculated using the open source tool Radon [19] on the current source code (as of commit

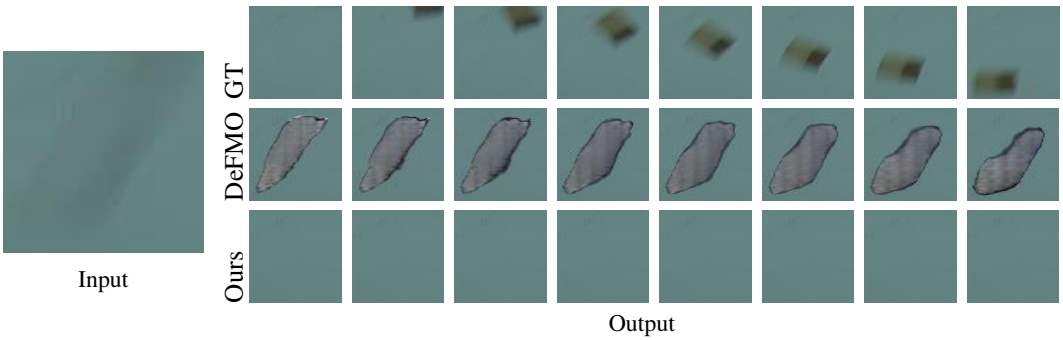

Figure 6: Sample outputs on the TbD dataset.

Table 4: Ablation study for memory usage (in MiB) across both approaches during training on a single GPU. We use gradient checkpointing with 2 segments over the encoder and renderer separately.

Technique	Approach			
	DeFMO	$+D_S$	$+D_T$	$+D_S, D_T$
None	2413	6351	6211	10023
Gradient checkpointing (GC)	2121	5933	5965	9803
Mixed precision (MP)	1807	4003	3997	6073
GC + MP	**1667**	**3909**	**3835**	**5991**

Table 5: Code quality comparison between original DeFMO and this project.

Approach	CC ↓		Difficulty ↓		Bugs ↓		MI ↑	
	Avg	Max	Avg	Max	Avg	Max	Avg	Min
DeFMO	4.647	78	7.030	13.852	0.480	2.228	44.89	8.19
Ours	**3.349**	**15**	**5.149**	**9.960**	**0.209**	**0.589**	**48.86**	**23.24**

`cc4a28106ca0332e8cbca17504ff6691288578a7`) and the original DeFMO repository (at the commit it was forked: `30c5af80e49c96d39090c4b51541cd6801d2934a`), excluding the code for rendering the synthetic dataset. The extrema and average values of these metrics are presented in Table 5.

As we can see, the refactoring done on the code base has significant improvement over the previous state, especially when considering the extrema of these metrics.

6 Conclusion

We proposed an improved version of DeFMO that uses GANs to overcome its problems with biased losses. We also proposed a novel loss function to capture smooth movement across sub-frames. This method can also generalize better to other related applications, like temporal super-resolution, due to more general loss functions. Experimental results show that the proposed model performs better than DeFMO on their benchmark, but at the cost of higher memory usage during training. We also implement popular memory optimization techniques and refactor the codebase to make is easier to use, maintain, and extend the approach. For future work, one can train the model on full resolution images with sufficient compute to possibly achieve state-of-the-art results.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015.

[3] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016.

[4] Don Coleman, Bruce Lowther, and Paul Oman. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29(1):3–16, 1995. ISSN 0164-1212. doi: https://doi.org/10.1016/0164-1212(94)00125-7. URL `https://www.sciencedirect.com/science/article/pii/0164121294001257`. Oregon Metric Workshop.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[7] Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., USA, 1977. ISBN 0444002057.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[10] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[11] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014. doi: 10.1109/CVPR.2014.223.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[13] Jan Kotera, Denys Rozumnyi, Filip Sroubek, and Jiri Matas. Intra-frame object tracking by deblatting. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct 2019. doi: 10.1109/iccvw.2019.00283. URL `http://dx.doi.org/10.1109/ICCVW.2019.00283`.

[14] Jan Kotera, Denys Rozumnyi, Filip Sroubek, and Jiri Matas. Intra-frame object tracking by deblatting. In *Internatioal Conference on Computer Vision Workshop (ICCVW), Visual Object Tracking Challenge Workshop, 2019*, Seoul, South Korea, October 2019.

[15] Jan Kotera, Denys Rozumnyi, Filip Sroubek, and Jiri Matas. Intra-frame object tracking by deblatting. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct 2019. doi: 10.1109/iccvw.2019.00283. URL `http://dx.doi.org/10.1109/ICCVW.2019.00283`.

[16] Jan Kotera, Jiří Matas, and Filip Šroubek. Restoration of fast moving objects. *IEEE Transactions on Image Processing*, 29:8577–8589, 2020. doi: 10.1109/TIP.2020.3016490.

[17] Jan Kotera and Filip Šroubek. Motion estimation and deblurring of fast moving objects. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2860–2864, 2018. doi: 10.1109/ICIP.2018.8451661.

[18] Matej Kristan, Jiri Matas, Ales Leonardis, Tomas Vojir, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka Cehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, Nov 2016. ISSN 2160-9292. doi: 10.1109/tpami.2016.2516982. URL `http://dx.doi.org/10.1109/TPAMI.2016.2516982`.

[19] Michele Lacchia. Radon, 2021. URL `https://github.com/rubik/radon`. Version 4.5.0.

[20] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976. doi: 10.1109/TSE.1976.233837.

[21] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[22] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

[23] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[24] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2): 40–53, March 2008. ISSN 1542-7730. doi: 10.1145/1365490.1365500. URL `https://doi.org/10.1145/1365490.1365500`.

[25] NVIDIA Corporation. NVIDIA System Management Interface. URL `https://developer.nvidia.com/nvidia-system-management-interface`. [accessed 4-June-2021].

[26] P. Oman and J. Hagemeister. Metrics for assessing a software system's maintainability. In *Proceedings Conference on Software Maintenance 1992*, pages 337–344, 1992. doi: 10.1109/ICSM.1992.242525.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[28] Denys Rozumnyi, Jan Kotera, Filip Sroubek, Lukas Novotny, and Jiri Matas. The world of fast moving objects. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.514. URL `http://dx.doi.org/10.1109/CVPR.2017.514`.

[29] Denys Rozumnyi, Jan Kotera, Filip Sroubek, and Jiri Matas. Sub-frame appearance and 6d pose estimation of fast moving objects. In *CVPR*, Seattle, Washington, USA, June 2020.

[30] Denys Rozumnyi, Martin R. Oswald, Vittorio Ferrari, Jiri Matas, and Marc Pollefeys. Defmo: Deblurring and shape recovery of fast moving objects. In *CVPR*, Nashville, Tennessee, USA, June 2021.

[31] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

[32] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, Student Member, Eero P. Simoncelli, and Senior Member. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004.