Fast Spatial-Temporal Transformer Network

Rafael Molossi Escher
Federal University of Rio Grande
Rio Grande, Brazil
Email: rafael-escher@furg.br

Rodrigo Andrade de Bem
Federal University of Rio Grande
Rio Grande, Brazil
Email: rodrigobem@furg.br

Paulo Lilles Jorge Drews Jr.
Federal University of Rio Grande
Rio Grande, Brazil
Email: paulodrews@furg.br

Abstract—**In computer vision, the restoration of missing regions in an image can be tackled with image inpainting techniques. Neural networks that perform inpainting in videos require the extraction of information from neighboring frames to obtain a temporally coherent result. The state-of-the-art methods for video inpainting are mainly based on *Transformer Networks*, which rely on *attention* mechanisms to handle temporal input data. However, such networks are highly costly, requiring considerable computational power for training and testing, which hinders its use on modest computing platforms. In this context, our goal is to reduce the computational complexity of state-of-the-art video inpainting methods, improving performance and facilitating its use in low-end GPUs. Therefore, we introduce the *Fast Spatio-Temporal Transformer Network* (FastSTTN), an extension of the *Spatio-Temporal Transformer Network* (STTN) in which the adoption of *Reversible Layers* reduces memory usage up to 7 times and execution time by approximately 2.2 times, while maintaining state-of-the-art video inpainting accuracy.**

I. Introduction

Image inpainting is the restoration of missing or damaged regions of an image by using information from its surrounding area [1]–[4]. Possible applications for this method also include removing watermarks or undesirable objects [1], [3], [4], as depicted in Fig. 1. Despite the rapid progress of deep neural networks in performing such tasks, it is still challenging to extend these methods to videos, due to the additional time dimension [2] and the need to maintain temporal coherence [3]. Due to sudden movements and non-trivial changes, performing video inpainting through independent frame-by-frame restorations may cause serious temporal inconsistencies [1], [2]. To overcome this issue, the extraction of temporal information from subsequent or preceding frames is required to obtain temporal coherence [1]. However, the use of such temporal information significantly increases the computational complexity of the networks [2], turning the video inpainting process unfeasible in some cases.

Recent state-of-the-art methods for processing temporal information are based on the Transformer Network [5], which makes efficient use of attention mechanisms. Outstanding results have been achieved by such attention-based models in various tasks, such as musical composition [6], natural language processing [7], and the generation of super-resolution images, proving capable of fooling human observers three times more than previous methods [8]. Despite its simplicity [5], training Transformer Networks can be extremely long and costly, and is often realistic only in powerful supercomputers [9].

Regarding video inpainting, still few approaches are found in the literature employing Transformer Networks. Nevertheless, the Spatial-Temporal Transformer Network (STTN) outperforms competing methods, achieving state-of-the-art in such a task [4]. Importantly, this breakthrough is accomplished at a high computational cost. According to the authors, for instance, the training and fine-tuning of the STTN have lasted 3 days and 1 day, respectively, on 8 NVIDIA V100 GPUs. Recent works have proposed modifications to the Transformer architecture, such as the Reformer Network [9], which aims for memory usage and execution time reduction through two different strategies.

In this context, the present work aims for reducing the computational complexity of attention-based state-of-the-art video inpainting, improving performance and facilitating its use in low-end GPUs. To achieve this goal, we introduce the Fast Spatio-Temporal Transformer Network (FastSTTN). Our architecture relies on the Spatial-Temporal Transformer Network (STTN) [4], augmented with *Reversible Layers* [9], which recalculate layers' activations when necessary, eliminating the necessity of storing all them in memory. Such a strategy allows for a considerable reduction in memory usage (up to 7x) and execution time (approximately 2.2x). We perform our experiments on the YouTube-VOS Dataset [10] achieving state-of-the-art video inpainting results. The FastSTTN accuracy is on par with other competing approaches in the literature, but at a significantly lower computational cost, which constitutes a relevant contribution.

II. Related Works

A. Video Inpainting

Existing works for image inpainting [11]–[13] usually adopt diffusion-based or patch-based methods which try to fill certain image regions with information extracted from other areas or from other frames. However, these methods fall short when applied to video inpainting due to the diversity of visual content present in the scenes and blurs caused by sudden camera movements [14].

Many learning-based methods found in the literature use neural networks with convolutions to map spatio-temporal information and predict pixels in missing regions [15]–[17]. Nonetheless, such methods present too many parameters making their training difficult [18]. Moreover, the use of convolutions present additional limitations such as spatial

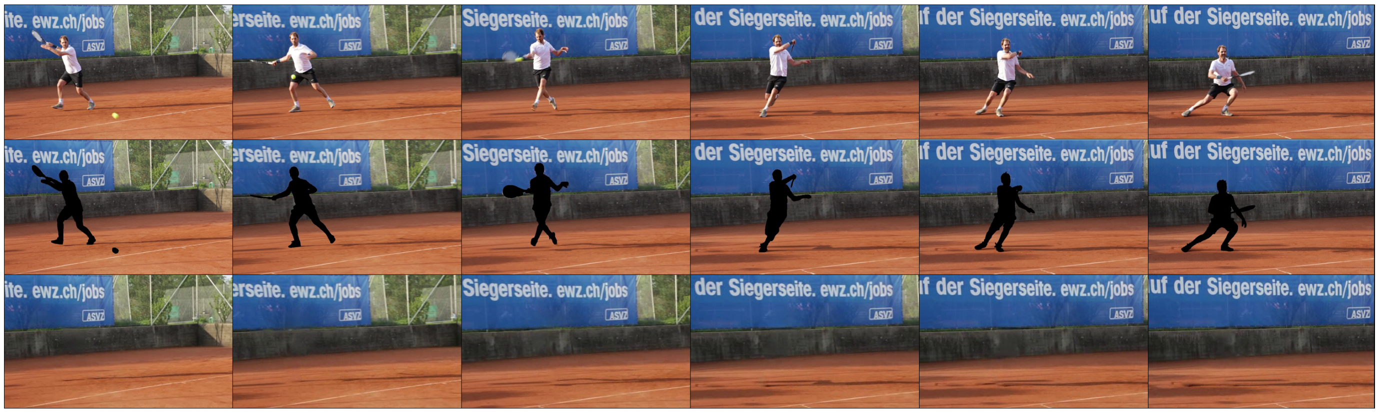


Fig. 1. FastSTTN sample results: our method is capable of inpainting missing regions in video frames *efficiently* taking into account spatio-temporal information. From the top to the bottom row: original frames, frames with undesirable objects suppressed with segmentation masks, and the resulting inpainted frames.

misalignment, lower resolution in the filled area, and blurry image when in complex scenes [1], [19].

On the other hand, distinct methods adopt 3D convolutions to deal with the additional dimension of time due to the need for temporal coherence [1], [2], [14], [20]. In these approaches, different techniques are used in order to capture information from neighboring frames to be applied to a target region. Other works adopt recurrence to improve the temporal stability of the results both in the short and long term, mitigating the problem of visual artifacts, when there is occlusion in a region and movements discontinuities [2].

Optical flow has also been extensively explored for video inpainting [19]–[21]. Xu et al. [20] consider that completing a missing flow is much easier than completing pixels in a missing region, since it would naturally preserves temporal coherence. Thus, the optical flow synthesized by their network is used to guide the propagation of pixels in order to fill in missing regions in the video. Most pixels in such regions can be propagated and deformed from visible regions, and the few regions that do not appear in the video are filled using pixel hallucination [14]. Strobel et al. [21] apply inpainting to the optical flow obtained from the input video to guide the copy-based frame-to-frame inpainting. This method fills target regions by copying and pasting patches from other regions. The connection between the reference frame and the region to be filled is obtained through the filled optical flow of the original scene through a similarity function. Thus, although the inpainting process is performed spatially, one frame at a time, the temporal consistency is maintained through the optical flow used to obtain the patches to be copy-pasted into the missing regions.

Zou et al. [19] take a different path, combining techniques used in 3D convolutional and optical flow methods to solve the problems of artifacts and spatial misalignment that occur in these methods. Their model uses a Temporal Shift Module (TSM), which combines 2D convolutions and channel shifting into temporal information to mimic 3D convolutions. The authors apply improvements to this module in order to correct the misalignment that occurs during the video inpainting. Such a module is used in feature maps at different scales and depths. On the other hand, the optical flow is used to align the temporal features of neighboring frames which are applied together with the spatial features of the target frame.

The methods that use optical flow handle well spatial information and are able to handle a higher resolution compared to convolutional and attention methods. Despite this, they are highly dependent on the accuracy of the optical flow and can generate visual artifacts or be unable to capture refined details in case of errors [19]. Also, they may fail to look up relevant information in the reference frame when there is background movement [22]. Finally, they may also suffer from the lack of texture in the inpainting [23].

B. *Transformer-based Video Inpainting*

The application of Transformer Networks for tackling video inpainting is still incipient in the literature. However, the few existing approaches already have achieved remarkable results in the task. Firstly, we can mention the *STTN* by Zeng et al. [4], which process spatio-temporal information from a set of frames extracted from a video in order to fill a region from a target frame defined by a binary mask. The model has an encoder-decoder architecture, with the encoder being responsible for transforming the input into features and passing them to the Transformer module, and the decoder responsible for processing the output of the Transformer and generate the restored image. This model is closely related to the present work and is revisited in Section III-B.

Liu et al. [22] introduce an architecture similar to the STTN, named Decoupled Spatial-Temporal Transformer (DSTT). This approach separates the spatio-temporal features into two groups: one group with temporal information and the other with spatial information. The group with the temporal information is passed to a temporal block, where continuous movements in small spatial zones are detected. Meanwhile, the group with spatial information is passed to a spatial block, which detects local textures in the image. The two groups are processed and the resulting information is applied to the region to be filled in the video.

Despite the state-of-the-art results achieved by such methods, both of them demand considerable computational power during training and testing processes. Our contribution, the FastSTTN, is an efficient Transformer Network which differs from these previous video inpainting approaches since it allows a substantial reduction in memory usage and execution time.

C. Efficient Transformer-based Methods

The literature shows that the original Transformer Network [5] presents high computational complexity requiring considerable computational power for training and testing. Thus, different methods for reducing memory consumption and the quadratic complexity of Transformer-based models are desirable. Thereby, many modifications emerged, proposing changes both in its architecture and in its attention modules [24].

Choromanski et al. [25] identify that the bottleneck of the Transformer is the operation $SoftMax(Q \cdot K^\top) \cdot V$, where Q (query), $K^\top$ (keys), and V (values) have the dimensions $[L, d]$, $[d, L]$ and $[L, d]$, respectively. The matrix resulting from the multiplication of Q and $K^\top$ has dimensions $[L, L]$. According to the authors, this problem could be solved if there were some way to decompose the *SoftMax* operation, so that first one calculates the multiplication of $K^\top$ and V to then calculates the multiplication of Q with the result of $K^\top \cdot V$, resulting in the approximation $SoftMax(Q \cdot K^\top) \cdot V \approx Q \cdot (K^\top \cdot V)$. This approximation would result in a linear complexity. However, according to the authors, using only trivial mathematical methods, it is not possible to carry out such a decomposition. Although, with the use of kernels, the authors manage to obtain an approximation of this function. Kernels, also called generalized dot products, represent the dot product in a feature space, which is usually high-dimensional [26]. In other words, with the use of kernels it is possible to multiply two matrices ($Q \cdot K^\top$) with a linear complexity, through a linear function that approximates the non-linear function *SoftMax*.

Therefore, a series of other improvements using kernels are proposed by Choromanski et al. [25] in a model called *Performer*. Kernel parameters are changed so that the equivalent function does not return negative numbers, just because the *SoftMax* does not return. The authors claim it is possible to train the network in very large datasets, such as the *ImageNet64*, with $L = 12288$, and the *TrEMBL*, with $L = 8192$, which is not possible with a Transformer within a single GPU.

Also with approximation methods, Wang et al. [27] demonstrate that the attention mechanism of the Transformer can be approximated by a low-rank matrix and proposes a new mechanism, reducing the complexity of the network from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$, both in space and time, making the complexity of their *Linformer* network linear. The authors assume that the attention matrix, obtained by multiplying the matrices Q $[n, d]$ and $K^\top$ $[d, n]$, resulting in a dimension $[n, n]$, can be represented by a smaller dimension matrix. Therefore, the matrix K $[n, d]$ is mapped, in an operation based on the lemma of Johnson–Lindenstraus [28], for an approximated matrix of smaller dimension $[k, d]$, causing the attention matrix $Q \cdot K^\top$ to have the dimensions $[n, k]$. Thus, it remains to also reduce the matrix V $[n, d]$ to $[k, d]$, making the matrix resulting from the operation $SoftMax(Q \cdot K^\top) \cdot V$ to have the dimensions $[n, d]$, thus being a linear operation.

Much like the Performer architecture, Katharopoulos et al. [29] modify the *SoftMax* function of the Transformer to apply kernels internally. According to the authors, the accumulated value $V_i^{'}$ for the *query* Q_i in the position i can be written in the *SoftMax* function as

$$V_i^{'} = \frac{\sum_{j=1}^{N} sim(Q_i, K_j) \cdot V_j}{\sum_{j=1}^{N} sim(Q_i, K_j)},$$

where $sim(Q, K) = \exp(\frac{Q^\top \cdot K}{\sqrt{d}})$ is the similarity function. In the network proposed by the authors, the *Linear Transformer*, the similarity function is expressed as a kernel function, that is, $sim(Q, K) = \phi(Q)^\top \phi(K)$, resulting in the expression

$$V_i^{'} = \frac{\phi(Q_i)^\top \cdot \sum_{j=1}^{N} \phi(K_j) \cdot V_j^\top}{\phi(Q_i)^\top \cdot \sum_{j=1}^{N} \phi(K_j)},$$

where $\phi(\cdot)$ is some function represented in a feature space, which is achieved through the use of Kernels. The terms $\sum_{j=1}^{N} \phi(K_j) \cdot V_j^\top$ and $\sum_{j=1}^{N} \phi(K_j)$ can only be calculated once and reused at each query, thus obtaining the linear time and memory complexity $\mathcal{O}(N)$.

III. Methodology

A. Preliminaries: the Transformer Network Attention Mechanism

The Transformer architecture [5] is well-known for using attention to model long-range dependencies. Doing so, it had a great impact in several areas of deep learning, such as image processing [8], [30], language modelling [31], [32], and question answering [7], [33]. The network is presented as an encoder-decoder architecture. In the encoder, each of its six layers contains two sublayers: a *Multi-Head Attention*, followed by a *Fully-Connected Feed-Forward*. The decoder is also made up of six layers, but each layer has three sublayers: in addition to the two sublayers in the encoder, there is another *Multi-Head Attention* sublayer, which performs attention to the output from the encoder. Its main differential is the efficient use of attention mechanisms, which connect tokens of a temporal sequence using relevance-based operations. Such mechanisms compare a feature with all the other features of a given input sequence, allowing for greater parallelization than other state-of-the-art networks. Instead of using the original features directly, they are used to project linearly the query, key and values matrices, which are essential to the operation of the Transformer. These projections have their own weights and are trainable together with the network. In a simplified way, these matrices go through scalar product operations and exponential normalization, causing the network to learn to focus on certain information from the input sequence. For example, Girdhar et al. [34] claim that the attention mechanisms have learned to emphasize hands and faces, in action recognition tasks, which is often crucial to distinguishing activities like “holding hands” or “looking at a person”.

The Transformer's *Multi-Head Attention* mechanism relates different positions of a single sequence and calculates its representation. In other words, it allows inputs to interact with each other and figure out where to "pay more attention". In this mechanism, the input has three representations: the matrices *query* (Q), *key* (K) and *value* (V), which are obtained by multiplying the input with a set of weights: $Q = X \cdot W_q$, $K = X \cdot W_k$, and $V = X \cdot W_v$, where X is the network's input, and W_q, W_k, W_v are trainable weights initialized by the network.

Since the first step of attention is to obtain the matrices *Q*, *K* and *V* for the input, the second step is to calculate its attention score. The score S determines the relevance of each input in relation to other inputs, and is calculated as: $S = Q \cdot K^\top$. This operation is known as dot product attention. The third and fourth steps are to calculate the softmax of the score S, and then to multiply it with the matrix *V*: $Z = SoftMax(S) \cdot V$. The idea here is to maintain the values of the inputs you want to focus on (pay attention), and eliminate irrelevant values from them. Additionally, the score is divided by $\sqrt{d_k}$ to obtain more stable gradients, where d_k is the dimension of the matrix K. This process, dubbed *Scaled Dot-Product Attention* by its authors, is represented by the formula

$$Z = SoftMax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V. \quad (1)$$

The *Scaled Dot-Product Attention* is also done multiple times in parallel "heads" and the outputs of all heads are concatenated and go through a linear layer, resulting in an output from the attention layer for its input.

B. Revisiting the STTN

The *Spatial-Temporal Transformer Network* (*STTN*) is a model proposed by Zeng et al. [4] for tackling video inpainting. For training, the *STTN* receives a set of five frames, the central frame being the target, and the two precedent and two subsequent frames to the target are the auxiliary frames, or neighbors. The masks are converted to binary and these are also passed as input, which highlights the missing regions to be filled by the network.

For inference, the input is composed of a target frame, the four neighboring frames, frames distant from them and also a binary mask for each frame. The distant frames are extracted from the video with step 10, *i.e.*, only one frame every ten. The amount of frames varies with the video length and with the position of the target frame. For example, a video with eighty frames in total would result in an input with thirteen frames: a set of five frames (reference and neighbors) and a set of $80/10 = 8$ (eight) distant frames, where the reference frames and neighbors are sequential but selected randomly throughout the video.

In the *STTN*, the *Multi-scale Patch-based Attention module* was proposed, which is responsible for extracting relevant information in both temporal and spatial dimensions of the video. In this module, the frames of the video are fragmented into patches (patch-based) of different scales (multi-scale): $[108, 60]$, $[36, 20]$, $[18, 10]$ and $[9, 5]$, which are processed simultaneously by four attention modules, called heads, in order to adapt to visual changes caused by complex motions in the video. Thus, when calculating similarities in frames from different scales, the heads capture the most relevant information and then fill in the missing region of the frames. Within each head, the similarity, or score, between the patches is calculated as:

$$s_{i,j} = \frac{p_i^Q \cdot (p_j^K)^\top}{\sqrt{r_1 \times r_2 \times c}}, \quad (2)$$

where $1 \leq i, j \leq N$; *N* is the number of patches of each *Q* and *K* matrices; p_i^Q is the *i*th patch of the *query* matrix; p_j^K is the *j*th patch of the *key* matrix; r_1, r_2 are the patch dimensions; and *c* is the number of channels of the matrices.

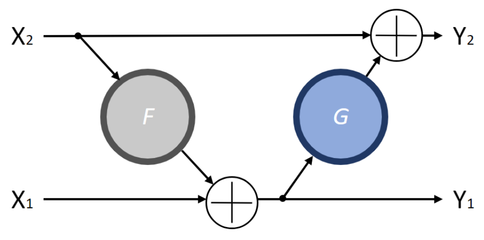
In addition to the *Transformer* layer used within the network, which the authors call *Multi-layer Multi-head Spatial-Temporal Transformer*, the network is composed of an encoder and a decoder. The encoder is composed of four 2D convolutional layers interspersed with four activation layers *LeakyReLU*, and generates features from the pixels of individual frames. The decoder is also composed of four 2D convolutional layers, but its last activation function is a hyperbolic tangent. Furthermore, two *Up Samplings* are performed between them. It is responsible for decoding the features in order to generate an image from them.

The network adopts as discriminator the *T-PatchGAN* [1], inspired by studies that shows that adversarial training helps in the generation of high quality content [1], [18], [35]. The *T-PatchGAN* is composed of six 3D convolutional layers, and it learns to distinguish whether an image, generated by the *STTN*, is real or fake, so that *STTN* learns to "cheat" the discriminator.

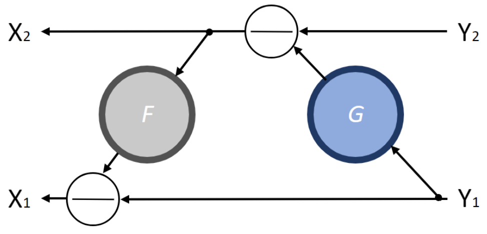
C. Our Approach: the FastSTTN

The general idea of the Transformer [5] is to use attention to calculate similarities between inputs (*e.g.*, words in a text) and know the relevance of each input to another. The inputs are represented by the *Q, K* and *V* matrices, and then the dot product of *Q* and *K* followed by a division by the square root of the *K* matrix dimension are performed. Then, a *SoftMax* of the resulting value is applied, which will ultimately be multiplied by the matrix *V*, and this is the attention process, also known as *Scaled Dot-Product Attention*. The problem of this method is the time complexity of this calculation, which is $\mathcal{O}(L^2)$, where L is the number of inputs to the network, making it computationally expensive, requiring a lot of memory when inputs are numerous.

Reversible Layers [36] constitute an strategy to overcome the previously mentioned issue. The main idea is to eliminate the need of storing the activations (*i.e.*, input and output tensors) of each layer of the network in memory. Normally, during the back-propagation, the calculation of the gradient of a layer is done using such activations, so it is necessary to have them in memory, after their computation during the forward pass. In contrast, a Reversible Layer allows activations to be calculated on-demand. Thus, during the backward pass, activations from



(a) Forward pass.



(b) Backward pass.

Fig. 2. Diagram showing how a Reversible Layer works.

one layer are retrieved only using the activations from the next layer. For this purpose, pairs of inputs x_1, x_2 and outputs y_1, y_2 are needed. In the forward pass, we have

$$y_1 = x_1 + F(x_2) \tag{3}$$

and

$$y_2 = x_2 + G(y_1), \tag{4}$$

while in the backward pass the outputs y_1 and y_2 are used to retrieve the inputs x_1 and x_2 with the equations

$$x_2 = y_2 - G(y_1) \tag{5}$$

and

$$x_1 = y_1 - F(x_2), \tag{6}$$

where $F(\cdot)$ and $G(\cdot)$ are residual layers. Forward and backward passes are illustrated in Fig. 2. For comparison, we have $y = F(x) + x$ in a traditional residual layer, as illustrated in Fig. 3.

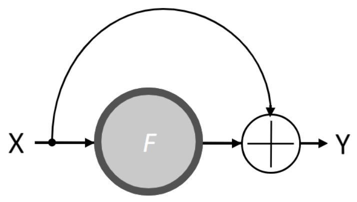


Fig. 3. Diagram showing how a traditional residual layer works.

In our FastSTTN architecture, illustrated in Fig. 4, we employ the Reversible Layers strategy in the Multi-Head Attention layer and in the Feed Forward layer to obtain reduction in memory consumption and faster runtime, without loss of accuracy for filling in frames. These layers are respectively denoted by the terms $F(\cdot)$ and $G(\cdot)$ in Eq. (3), (4), (5), and (6).

Furthermore, in order to reduce the memory occupied by the Feed Forward layer, its processing is done in parts, where its activations are separated into chunks, changing the Eq. (4) to:

$$y_2 = [y_2^1, ..., y_2^{n_c}] = [x_2^1 + G(y_1^1), ..., x_2^{n_c} + G(y_1^{n_c})], \tag{7}$$

where n_c is the number of chunks. Note that there is no difference in memory consumption by processing the *Feed Forward* in a normal fashion, or in chunks in parallel, but rather by dividing into n_c chunks and processing one by one.

IV. Experiments

In this section, we present experiments and results which highlight the computational cost and the quality of our FastSTTN method for video inpainting. We compare our results regarding both training and inference with the closest related method, the *STTN* [4]. The experiments are performed on the YouTube-VOS Dataset [10] and executed in a NVIDIA Tesla T4 16GB GPU. Our model was implemented using the PyTorch library and its code will be publicly available.

A. Dataset

We train the FastSTTN on YouTube-VOS dataset [10], a well-known dataset for video inpainting and the largest dataset for object segmentation in videos. This dataset contains 94 categories of high-resolution videos such as animals, vehicles and accessories, which have been carefully selected to contain a wide variety of objects and movements and to represent everyday scenarios. The dataset has 3,471 training videos, 474 validation videos, and 508 test videos, all taken from YouTube with each video lasting between 3 to 6 seconds. In addition to the diversity of videos, the dataset also includes segmentation of objects in the scenes manually annotated. Fig. 5 shows samples from the dataset.

The *Youtube-VOS* dataset can be used to evaluate the object removal task in videos (video inpainting) because the segmentation of the videos presents can be adopted as binary masks for the neural network, where it highlights the region of the image to be filled. However, for training the model, it is not possible to use them because it is not known what is actually behind the objects removed by the network, making it difficult to assess whether the network filled the region correctly or not. Thus, following [4], the training receives as input the videos from the dataset and masks with random shape and motion are created and applied to the frames of the videos before they enter the network. Samples of randomly generated mask are shown in Fig. 6.

Usually when the region to be filled is too large, the inpainting result is not satisfactory. To avoid this problem, the randomly generated masks have a maximum size. More specifically, the movement of the masks is defined by chance, where half of the generated masks have some random movement, and the other half are fixed.

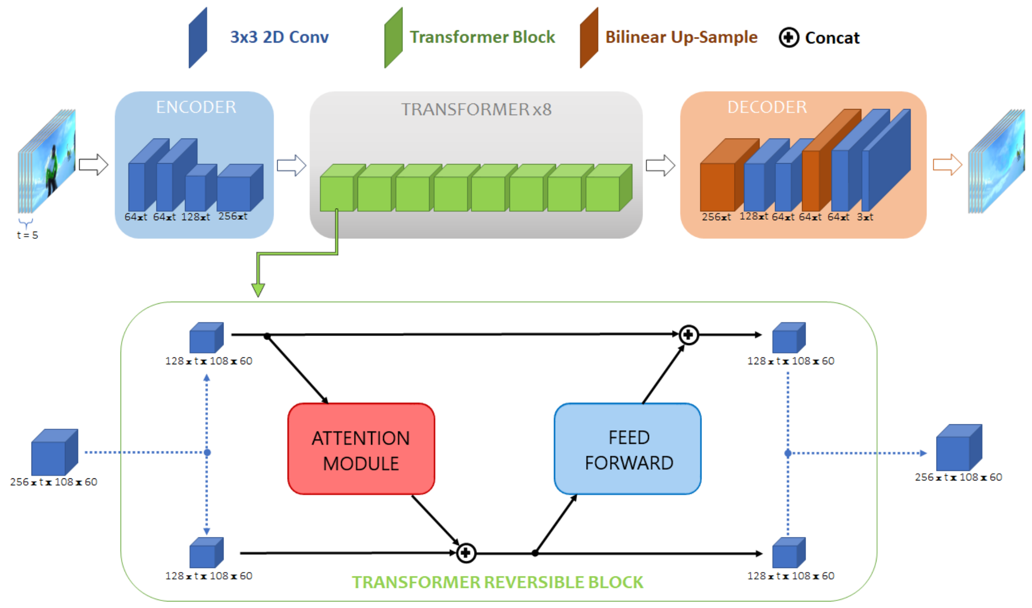


Fig. 4. The *FastSTTN* architecture. It has an encoder, eight *Transformer* reversible blocks, and a decoder. The encoder receives the input frames and generates features from them, and passes these features to the Transformer module. The Transformer process these spatio-temporal features and learns to fill in the holes in the frames. The decoder transforms the features processed by the Transformer back to images. During training, the input is composed of five sequential masked frames. For inference, the input is composed of both sequential masked frames and distant frames, with its size varying accordingly to the video size.

Fig. 5. Samples from the Youtube-VOS dataset. Two frame samples (left) and their respective semantic segmentation masks (right).

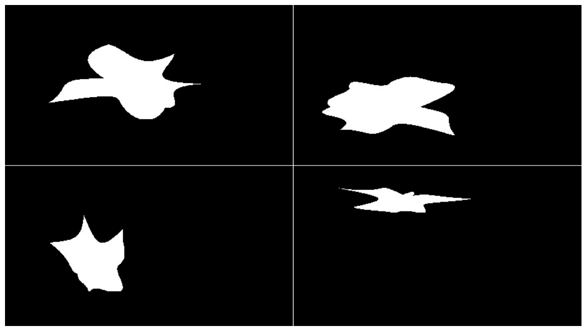


Fig. 6. Examples of randomly generated masks that are applied to the input frames during training.

B. Results

In order to assess the efficiency of our FastSTTN method, we compare it with the closest related state-of-the-art method in the literature, the STTN [4]. In our low-end setup, we firstly measure the memory usage and the average time per iteration of the two methods during training. A batch size of 2 frames was considered initially since the 8 frames batch originally employed in the STTN was not supported by a single NVIDIA Tesla T4 GPU. As shown in Tab. I, our FastSTTN significantly reduces memory usage and the execution time per iteration during training. With batch size equals to 2, the memory needed by the network is reduced to almost half. Furthermore, even with a low-end GPU with 16GB of memory, it is possible to train the FastSTTN with batches of 8 frames, unlike the STTN.

Secondly, we perform experiments regarding inference time.

TABLE I
MEMORY AND AVERAGE TIME CONSUMPTION PER ITERATION AT TRAINING FOR THE STTN AND THE FASTSTTN ACCORDING TO BATCH SIZE.

Batch size	*STTN*		*FastSTTN*	
	Memory	Time	Memory	Time
2	9.7 GB	$\approx$ 2.8s	5.3 GB	$\approx$ 2.0s
4	MemOut	-	12.1 GB	$\approx$ 3.7s
8	MemOut	-	13.7 GB	$\approx$ 7.5s

Tab. II shows the results of experiments performed to identify the maximum number of frames that the networks can process in an inference step. While the *STTN* can process up to 13 frames with 16GB of memory, the FastSTTN can handle up to 123 frames, showing an impressive increase of 9 times the number of frames that can be stored on the GPU, allowing for more complex and diverse experiments. As also happens in training, the FastSTTN was faster than the STTN in inference, processing the same amount of frames approximately 2.2 times faster.

TABLE II
MEMORY OCCUPIED AND FRAMES PER SECOND (FPS) AT THE INFERENCE OF THE STTN AND THE FASTSTTN IN FUNCTION OF THE NUMBER OF INPUT FRAMES (NO. OF FRAMES).

No. of Frames	*STTN*		*FastSTTN*	
	Memory	FPS	Memory	FPS
5	6.7 GB	≈ 14.5FPS	1.3 GB	≈ 34.4FPS
6	7.5 GB	≈ 14.3FPS	1.3 GB	≈ 33.8FPS
7	8.8 GB	≈ 14.9FPS	1.4 GB	≈ 33.7FPS
8	9.9 GB	≈ 14.9FPS	1.4 GB	≈ 33.6FPS
9	11.2 GB	≈ 15.0FPS	1.5 GB	≈ 33.9FPS
10	13.3 GB	≈ 15.7FPS	1.8 GB	≈ 33.8FPS
11	13.3 GB	≈ 15.8FPS	1.8 GB	≈ 35.0FPS
12	14.7 GB	≈ 15.3FPS	1.9 GB	≈ 33.3FPS
13	14.9 GB	≈ 14.7FPS	1.9 GB	≈ 33.0FPS
14	MemOut	-	1.9 GB	≈ 35.3FPS
20	MemOut	-	2.8 GB	≈ 30.8FPS
30	MemOut	-	3.1 GB	≈ 26.7FPS
40	MemOut	-	4.1 GB	≈ 23.4FPS
50	MemOut	-	5.1 GB	≈ 22.0FPS
60	MemOut	-	5.2 GB	≈ 21.5FPS
80	MemOut	-	12.4 GB	≈ 17.3FPS
120	MemOut	-	14.8 GB	≈ 13.9FPS
123	MemOut	-	15.1 GB	≈ 13.0FPS

Following the experiments described previously, regarding the FastSTTN efficiency and performance, we also have evaluated its actual accuracy for video inpainting. To this end, we have trained the FastSTTN using the same training parameters employed in [4], except for the batch size which was equal to 2. The training was finalized with 215k iterations using the early stopping strategy, taking approximately 5 days. We have compared our results with the original pre-trained STTN model (STTN), as well as with a STTN model trained from scratch, which also presented convergence around 215k iterations (STTN 215k), which took approximately 7 days. In Fig. 7 we show qualitative results using fixed random-format masks.

We also perform a quantitative comparison between the same models using the metrics *SSIM*, *PSNR* and flow warping error (E_{warp}). The first two are well-known measures of image quality, while the last evaluates the temporal stability of the results, as proposed by [37]. It can be seen in Tab. III that all methods presents similar results in terms of accuracy, even though the improvement of the *FastSTTN*. In terms of image quality, the metrics *SSIM* and *PSNR* show slightly better values for the *STTN*, while the *FastSTTN* has better values for temporal stability.

TABLE III
QUANTITATIVE EVALUATION IN FUNCTION OF THE METRICS *SSIM*, *PSNR* AND FLOW WARPING ERROR E_{warp}. * BIGGER IS BETTER. † SMALLER IS BETTER.

Models	$PSNR\ (dB)^\star$	$SSIM\ ^\star$	$E_{warp}\ ^\dagger$
STTN [4]	19.16	0.8361	0.000912
STTN (215K it)	19.04	0.8332	0.000937
FastSTTN (215K it)	19.02	0.8297	0.000913

It is worth mentioning that [4] does not mention the number of iterations that they trained the *STTN* on the YouTube-VOS dataset, only that the training took three days on 8 NVIDIA V100 GPUs, and they even perform an one-day *fine-tuning* on the *DAVIS* dataset [38]. In this work, we trained the STTN and FastSTTN models up to 215k iterations for comparative purposes between them, in order to demonstrate that the networks are equivalent.

V. CONCLUSION

We propose the FastSTTN, a deep network that aims to reduce the memory consumption and the running time at training and inference for state-of-the-art Transformer-based video inpainting. We have employed the Reversible Layers method in the Transformer blocks of our architecture. The FastSTTN achieves a significant memory reduction and, consequently, an improvement regarding execution time, as demonstrated in Section IV by our experiments and results. Such a reduction happens without significant performance loss, according to our qualitative and quantitative results regarding the video inpainting quality. The FastSTTN has achieved state-of-the-art results on the YouTube-VOS dataset. Its accuracy is similar to the closest related method in the literature, the STTN, but more efficient in terms of memory consumption and processing speed, which is a relevant contribution. Future works will be focused in improving the network architecture to increase the overall accuracy. Furthermore, we intent to evaluate our method in other video inpainting dataset, such as DAVIS [38]. Finally, other Transformer optimization can be evaluated.

REFERENCES

[1] Y.-L. Chang, Z. Y. Liu, K.-Y. Lee, and W. Hsu, "Free-form video inpainting with 3d gated convolution and temporal patchgan," in *ICCV*, 2019.

[2] D. Kim, S. Woo, J.-Y. Lee, and I. S. Kweon, "Recurrent temporal aggregation framework for deep video inpainting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[3] S. Lee, S. W. Oh, D. Won, and S. J. Kim, "Copy-and-paste networks for deep video inpainting," in *ICCV*, 2019.

[4] Y. Zeng, J. Fu, and H. Chao, "Learning joint spatial-temporal transformations for video inpainting," in *ECCV*, 2020.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[6] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure." in *ICLR*, 2019.

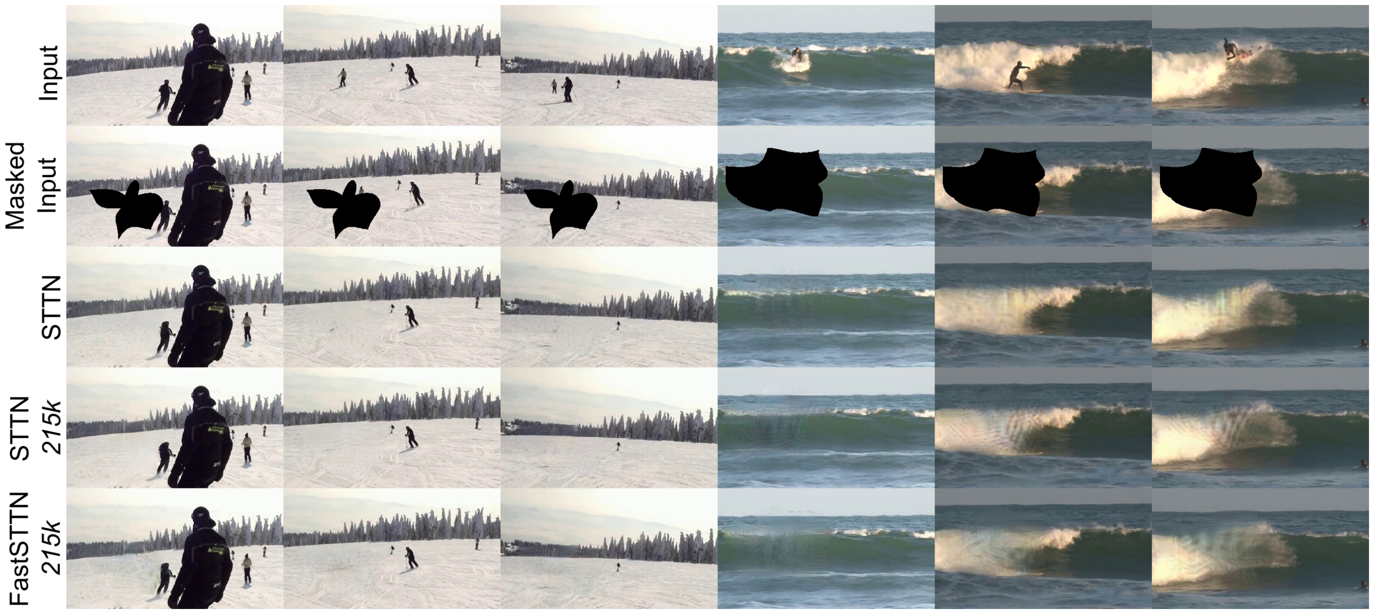


Fig. 7. Comparison of the results of the STTN, the STTN 215k, and the FastSTTN. The first two lines show, respectively, the original video frames and the video frames after the application of fixed random-format mask. While the following lines show the inpainting results from each model. This comparison shows distant frames from two videos of the YouTube-VOS dataset. We selected the frames #46, #108 and #152 for the first video, and the frames #1, #101 and #128 for the second video. More results on youtu.be/XPEa7ccqd40.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[8] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *ICML*, 2018.

[9] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *ICLR*, 2020.

[10] N. Xu, L. Yang, Y. Fan, D. Yue, Y. Liang, J. Yang, and T. Huang, "Youtube-vos: A large-scale video object segmentation benchmark," *arXiv preprint arXiv:1809.03327*, 2018.

[11] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch: A randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics*, 2009.

[12] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani, "Summarizing visual data using bidirectional similarity," in *CVPR*, 2008.

[13] S. Darabi, E. Shechtman, C. Barnes, D. Goldman, and P. Sen, "Image melding: Combining inconsistent images using patch-based synthesis," *ACM Transactions on Graphics*, 2012.

[14] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," *arXiv preprint arXiv:1801.07892*, 2018.

[15] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Trans. Graph.*, Jul. 2017.

[16] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. Efros, "Context encoders: Feature learning by inpainting," in *CVPR*, 2016.

[17] L. Xu, J. Ren, C. Liu, and J. Jia, "Deep convolutional neural network for image deconvolution," *NIPS*, 2014.

[18] Y.-L. Chang, Z. Y. Liu, K.-Y. Lee, and W. Hsu, "Learnable gated temporal shift module for deep video inpainting"," *BMVC*, 2019.

[19] X. Zou, L. Yang, D. Liu, and Y. J. Lee, "Progressive temporal feature alignment network for video inpainting," in *CVPR*, 2021.

[20] R. Xu, X. Li, B. Zhou, and C. C. Loy, "Deep flow-guided video inpainting," in *CVPR*, 2019.

[21] M. Strobel, J. Diebold, and D. Cremers, "Flow and color inpainting for video completion," in *German Conference on Pattern Recognition*, 2014.

[22] R. Liu, H. Deng, Y. Huang, X. Shi, L. Lu, W. Sun, X. Wang, J. Dai, and H. Li, "Decoupled spatial-temporal transformer for video inpainting," *arXiv preprint arXiv:2104.06637*, 2021.

[23] M. Liao, F. Lu, D. Zhou, S. Zhang, W. Li, and R. Yang, "Dvi: Depth guided video inpainting for autonomous driving," in *ECCV*, 2020.

[24] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, "Adaptive attention span in transformers," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, Jul. 2019.

[25] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *International Conference on Learning Representations*, 2021.

[26] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, Jun 2008.

[27] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *ArXiv*, 2020.

[28] W. Johnson and J. Lindenstrauss, "Extensions of lipschitz maps into a hilbert space," *Contemporary Mathematics*, 01 1984.

[29] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *ICML*, 2020.

[30] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *arXiv preprint arXiv:2103.14030*, 2021.

[31] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[32] Y. Tay, M. Dehghani, V. Aribandi, J. Gupta, P. Pham, Z. Qin, D. Bahri, D.-C. Juan, and D. Metzler, "Omninet: Omnidirectional representations from transformers," *arXiv preprint arXiv:2103.01075*, 2021.

[33] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *CoRR*, 2021.

[34] R. Girdhar, J. a. Carreira, C. Doersch, and A. Zisserman, "Video Action Transformer Network," in *CVPR*, 2019.

[35] F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo, "Learning texture transformer network for image super-resolution," in *CVPR*, June 2020.

[36] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *NIPS*, 2017.

[37] W.-S. Lai, J.-B. Huang, O. Wang, E. Shechtman, E. Yumer, and M.-H. Yang, "Learning blind video temporal consistency," in *ECCV*, 2018.

[38] S. Caelles, A. Montes, K.-K. Maninis, Y. Chen, L. Van Gool, F. Perazzi, and J. Pont-Tuset, "The 2018 davis challenge on video object segmentation," *arXiv:1803.00557*, 2018.