ViSt3D: Video Stylization with 3D CNN Supplementary Material

Ayush Pande
IIT Kanpur
ayushp@cse.iitk.ac.in

Gaurav Sharma
TensorTour & IIT Kanpur
gaurav@tensortour.com

1 Dataset

1.1 Generating Motion clips

The algorithm for generating motion clips is given in Algorithm 1. It randomly samples a video as input and generates a random video clip from the input video. We used PySceneDetect library [1] to detect shot boundaries in the video. Algorithm 2 is used to generate a HSV image from the given flow of pair of consecutive frames of the video clip generated by Algorithm 1. Algorithm 3 is the key algorithm which decides whether the generated random video clip has strong motion or not.

1.2 Details

In this Algorithm 1, we select a 16-frame clip from the video that exhibits strong motion. We consider up to $15,999$ frames from each video. We utilize the PySceneDetect library [1] to generate shots from the video. Additionally, we employ Transnet-v2 [3], which is a deep learning-based method for shot detection, to achieve more accurate results. Afterwards, we sort the list of shots obtained from the above detection algorithms. We randomly select up to 9 shots to find a strong motion clip in the video. If no such clip is found, we assume that there is no desired strong motion clip in the considered video. To determine whether a clip has strong motion, we utilize the Farneback's algorithm [2] to generate flow maps for the video clip under consideration.

To interpret this flow map, we use Algorithm 2 to highlight areas in the image with strong and low motion. From the output of the *draw_hsv* algorithm, we are able to easily interpret which areas of the motion pair have strong motion, represented by colors near red, and low motion, represented by colors near blue when visualized. We also utilize Algorithm 3 that employs thresholds to determine whether the motion qualifies as strong motion and whether a clip can be classified as a strong motion clip.

The *Approve* Algorithm 3 takes the output of the *draw_hsv* Algorithm 2 as input. We perform the following steps for each frame in the clip.

First, we calculate the variance in the pixel values of the image. We take this as a proxy indicating motion in the video. If the variance is greater than 100, a threshold determined by initial experiments, it indicates the presence of areas with strong motion and weak motion in the image. This calculation is done to account for the relative strength of motion in pixels compared to other pixels.

Next, we find the unique pixel values in the image along with their frequencies. Then, we calculate the sum of frequencies for all pixels whose values are greater than 80 and store the result in the *sums* array.

After that, we iterate through the *sums* array and increment the *approve* variable if the sum is greater than 20000. Then, we check if the value of *approve* is greater than 9, which indicates the presence of more than 9 frames with strong motion. If this condition is satisfied, we consider the clip to have strong motion and add it to our dataset.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Algorithm 1 Generate motion clip(total_frames, video_path)

$end \leftarrow \min(15999, total\_frames)$ {Number of frames of video to be processed}
if $end = 0$ **then**
  $return$
end if
$scenes \leftarrow$ scenedetect$(video_path, end)$ {scenedetect function output a list of shots and its other parameters. Each entry of scenes list is another list with data like [start_time, end_time, num_frames]}
sort(scenes,key=lambda x:x[2] ,reverse=True) {sort function sorts the shots by shot length in descending order}
$run\_times = 0$
while $run\_times < \min(9, len(scenes))$ **do**
  $start = scenes[run\_times][0]$
  $end = scenes[run\_times][1]$
  $num\_frames = scenes[run\_times][2]$
  if $num\_frames \leq 17$ **then**
    return False
  end if
  $frame\_start =$ randint$(start + 1, end - 16)$ {randint generate random number between the two input numbers}
  $images =$ ARRAY[]
  $imgs =$ ARRAY[]
  $video =$ load_video$(video_path)$
  for $i = 0$ to $frame\_start$ **do**
    $ret, prev =$ read_i$(video)$ {we are reading the i^{th} frame in the video}
  end for
  $ret, prev =$ read$(video)$ {we are reading the frame whose position in video is given by frame_start}
  append$(images,$ bgr_to_rgb$(prev))$ {converting prev from bgr to rgb}
  $prevgray =$ bgr_to_gray$(prev)$ {converting frame to grayscale}
  for $i = 0$ to $frame\_count - 1$ **do**
    $ret, suc =$ read_i$(video)$
    append$(images,$ bgr_to_rgb$(suc))$
    $suc\_gray =$ bgr_to_gray(suc)
    $flow =$ calcOpticalFlowFarneback$(prevgray, suc_gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)$
    {Calling farneback algorithm for evaluating the flow between frames}
    $prevgray = suc\_gray$
    $hsv\_img =$ draw_hsv$(flow)$ {Algorithm mentioned in 2}
    append$(imgs, hsv_img)$
  end for
  if len$(images) < frame_count$ **then**
    return FALSE
  end if
  if approve(imgs, frame_count) **then**
    return (TRUE, outpath, filename, images, frame_start) {Algorithm mentioned in 3}
  else
    $run\_times = run\_times + 1$
  end if
end while

Algorithm 2 draw_hsv (flow)

$h, w = flow.shape[:2]$
$fx, fy = flow[:,:,0], flow[:,:,1]$
$ang = \text{arctan2}(fy, fx) + \pi$
$v = \text{sqrt}(fx * fx + fy * fy)$
$hsv = \text{zeros}((h, w, 3), \text{uint8})$
$hsv[\ldots, 0] = ang * (180/\pi/2)$
$hsv[\ldots, 1] = 255$
$hsv[\ldots, 2] = \min(v * 4, 255)$
$bgr = \text{hsv\_to\_bgr}(hsv)$
return bgr

Algorithm 3 approve (imgs, frame_count)

$sums = \text{ARRAY}[]$
for $i = 0$ to $frame\_count - 1$ **do**
  $img = imgs[i]$
  if $\max(img) - \min(img) \geq 100$ **then**
    $counts = \text{count\_unique\_values\_with\_frequency}(img, return\_counts = True)$
    $sum = 0$
    for $i = 0$ to $\text{len}(counts[0])$ **do**
      if $counts[0][i] \geq 80$ **then**
        $sum = sum + counts[1][i]$
      end if
    end for
    $\text{append}(sums, sum)$
  end if
end for
$approve\_count = 0$
for $i = 0$ to $\text{len}(sums)$ **do**
  if $sums[i] > 2000$ **then**
    $approve\_count = approve\_count + 1$
  end if
end for
return $approve\_count \geq \text{float}(0.5625 * frame\_count)$

1.3 Dataset Examples

We show some clips curated with the help of the above-mentioned algorithms to showcase examples of strong motion clips. These examples are represented by a series of frames from the clips, which can be seen in Figures 1, 2, 3.

2 Supplementary files details

Table 1 details the video results found in the folders along with their corresponding file descriptions.

3 Limitations

Despite our best efforts to control the flashing artifacts with the proposed intra-clip loss, some flashing still occurs in challenging edge cases. The utilization of 3D CNN in all components of the architecture also leads to an increase in both inference time and memory consumption for stylized video generation, when compared to networks based on 2D CNNs.

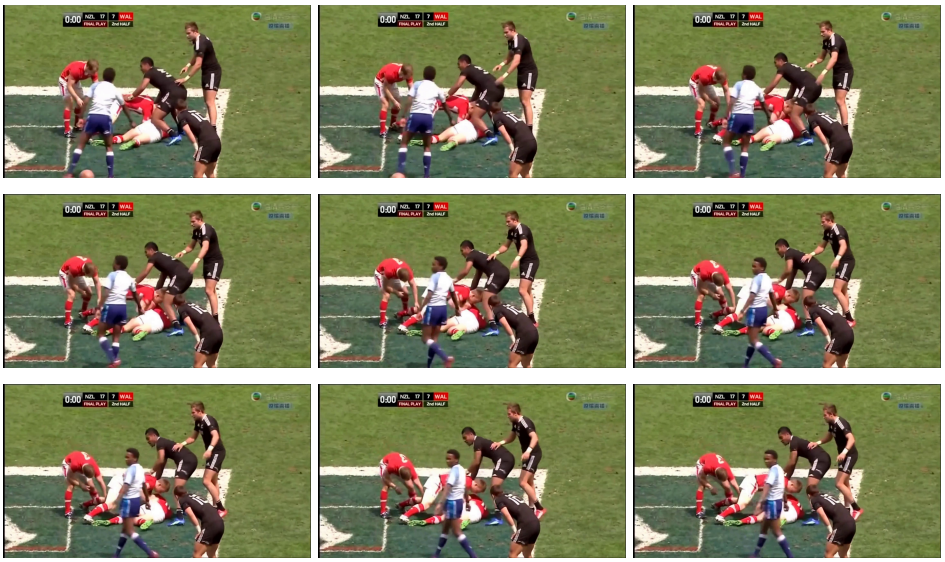

Figure 1: Some examples of our dataset. $1^{st}, 3^{rd}, 5^{th}, 7^{th}, 9^{th}, 11^{th}, 13^{th}, 15^{th}, 16^{th}$ frames of clip

Figure 2: Some examples of our dataset. $1^{st}, 3^{rd}, 5^{th}, 7^{th}, 9^{th}, 11^{th}, 13^{th}, 15^{th}, 16^{th}$ frames of clip

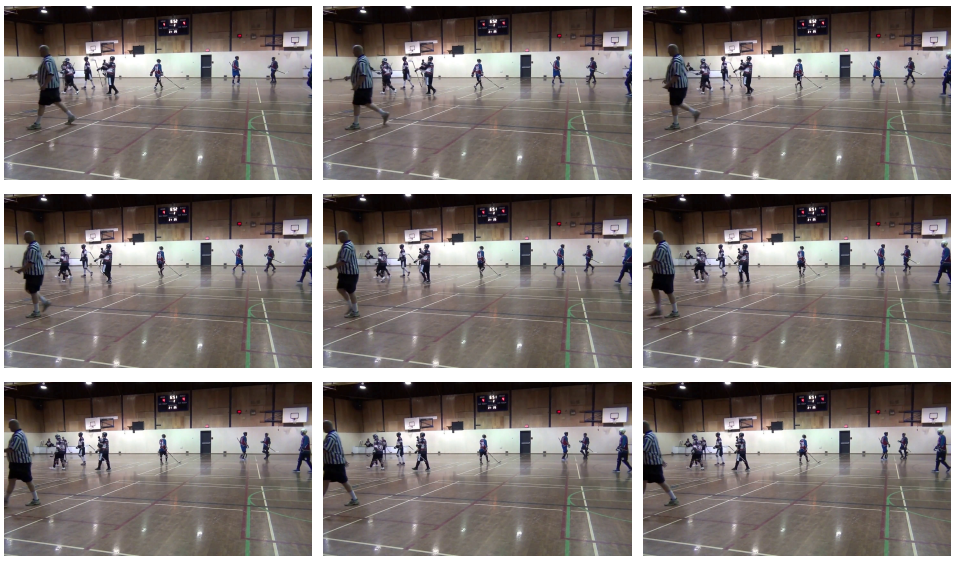

Figure 3: Some examples of our dataset. $1^{st}, 3^{rd}, 5^{th}, 7^{th}, 9^{th}, 11^{th}, 13^{th}, 15^{th}, 16^{th}$ frames of clip

File Name	Description
Ours Results/basic_results1.mp4 Ours Results/basic_results2.mp4 Ours Results/basic_results3.mp4 Ours Results/basic_results4.mp4	Ours results on different styles of a single content video
Ours Results/long_video.mp4 Ours Results/long_video1.mp4	Ours results on different styles of a single long content video
Comparative Analysis/comparative_study_result1.mp4 Comparative Analysis/comparative_study_result2.mp4 Comparative Analysis/comparative_study_result3.mp4 Comparative Analysis/comparative_study_result4.mp4	Qualitative comparison between other methods and ours
Videos_without_intra/basic_results1.mp4 Videos_without_intra/basic_results2.mp4 Videos_without_intra/basic_results3.mp4 Videos_without_intra/basic_results4.mp4	Ours without Intra-clip loss results on different styles of a single content video
Videos_without_temporal/basic_results1.mp4 Videos_without_temporal/basic_results2.mp4 Videos_without_temporal/basic_results3.mp4 Videos_without_temporal/basic_results4.mp4	Ours without temporal loss results on different styles of a single content video
Videos_with_naive_extension_of_2D_stylization/basic_results1.mp4 Videos_with_naive_extension_of_2D_stylization/basic_results2.mp4 Videos_with_naive_extension_of_2D_stylization/basic_results3.mp4 Videos_with_naive_extension_of_2D_stylization/basic_results4.mp4	Results using naive extension of 2D stylization methods to 3D CNN.

Table 1: File paths and descriptions of all the Supplementary video results found in various folders.

4 Network Details

4.1 Encoder

We have used pre-trained C3D network and renamed some of the layers of the network for the simplicity of understanding the architecture in Figure 1 in the main paper. The architecture code of encoder is shown in Table 2.

4.2 Entangle Subnet

Table 3 details the architecture of Entangle Subnet where weights are pre-initialized such that weights of first half of the layer is given value 0.3 and the other half is given the value 0.7.

4.3 Appearance Subnets

We designed the architecture layers based on the number of channels in the output feature maps of the C3D network i.e. the feature map with most number of channels will have the corresponding appearance map with high number of layers. Table 4 represents architecture of Appearance Subnet 1.

Layer	Layer Size	Stride	Output Size	Stage
Input		1	$3 \times 16 \times 128 \times 128$	
Conv	$3 \times 3 \times 3 \times 3$	0	$3 \times 16 \times 128 \times 128$	
Conv+RELU	$64 \times 3 \times 3 \times 3$	1	$64 \times 16 \times 128 \times 128$	F1
MaxPool		$(1, 2, 2)$	$64 \times 16 \times 64 \times 64$	F2
Conv+RELU	$128 \times 3 \times 3 \times 3$	1	$128 \times 16 \times 64 \times 64$	F3
MaxPool		2	$128 \times 8 \times 32 \times 32$	F4
Conv+RELU	$256 \times 3 \times 3 \times 3$	1	$256 \times 8 \times 32 \times 32$	F5
Conv+RELU	$256 \times 3 \times 3 \times 3$	1	$256 \times 8 \times 32 \times 32$	F6
MaxPool		2	$256 \times 4 \times 16 \times 16$	F7
Conv+RELU	$512 \times 3 \times 3 \times 3$	1	$512 \times 4 \times 16 \times 16$	F8

Table 2: Architecture of Encoder where Stage represents different intermediate feature maps.

Layer	Layer Size	Stride	Output Size	Stage
Input Concat F8+F9			$1024 \times 4 \times 16 \times 16$	
Conv+RELU	$512 \times 1 \times 1 \times 1$	1	$512 \times 4 \times 16 \times 16$	F13

Table 3: Architecture of Entangle Subnet where Stage represents different intermediate feature maps.

Layer	Layer Size	Stride	Output Size	Stage
Input			$512 \times 4 \times 16 \times 16$	
(Conv+RELU) $\times 4$	$512 \times 3 \times 3 \times 3$	1	$512 \times 4 \times 16 \times 16$	
Conv+RELU	$512 \times 1 \times 1 \times 1$	1	$512 \times 4 \times 16 \times 16$	F9

Table 4: Architecture of Appearance Subnet 1 where Stage represents different intermediate feature maps.

The table 5 represents architecture of Appearance subnet 2.

Layer	Layer Size	Stride	Output Size	Stage
Input			$256 \times 8 \times 32 \times 32$	
(Conv+RELU) $\times 3$	$256 \times 3 \times 3 \times 3$	1	$256 \times 8 \times 32 \times 32$	
Conv+RELU	$256 \times 1 \times 1 \times 1$	1	$256 \times 8 \times 32 \times 32$	F10

Table 5: Architecture of Appearance Subnet 2 where Stage represents different intermediate feature maps.

The table 6 represents architecture of Appearance subnet 3.

Layer	Layer Size	Stride	Output Size	Stage
Input			$128 \times 16 \times 64 \times 64$	
(Conv+RELU) $\times 2$	$128 \times 3 \times 3 \times 3$	1	$128 \times 16 \times 64 \times 64$	
Conv+RELU	$128 \times 1 \times 1 \times 1$	1	$128 \times 16 \times 64 \times 64$	F11

Table 6: Architecture of Appearance Subnet 3 where Stage represents different intermediate feature maps.

The table 7 represents architecture of Appearance subnet 4.

Layer	Layer Size	Stride	Output Size	Stage
Input			$64 \times 16 \times 128 \times 128$	
Conv+RELU	$64 \times 3 \times 3 \times 3$	1	$64 \times 16 \times 128 \times 128$	
Conv+RELU	$64 \times 1 \times 1 \times 1$	1	$64 \times 16 \times 128 \times 128$	F12

Table 7: Architecture of Appearance Subnet 4 where Stage represents different intermediate feature maps.

4.4 Decoder

The architecture of 3D CNN decoder is inspired from the C3D encoder mentioned above. We break the decoder into four parts where starting of each part takes output of appearance subnet concatenated channel-wise with the output of the previous part of the decoder. The architecture of decoder is detailed in Table 8.

Layer	Layer Size	Stride	Output Size	Stage
Input F13			$512 \times 4 \times 16 \times 16$	
(Conv+RELU) $\times 2$	$256 \times 3 \times 3 \times 3$	1	$256 \times 4 \times 16 \times 16$	
Upsample		$\frac{1}{2}$	$256 \times 8 \times 32 \times 32$	Part0
Concat F6			$512 \times 8 \times 32 \times 32$	
(Conv+RELU) $\times 2$	$256 \times 3 \times 3 \times 3$	1	$256 \times 8 \times 32 \times 32$	
Upsample		$\frac{1}{2}$	$256 \times 16 \times 64 \times 64$	Part1
Concat F3			$384 \times 16 \times 64 \times 64$	
Conv+RELU	$384 \times 3 \times 3 \times 3$	1	$384 \times 16 \times 64 \times 64$	
(Conv+RELU) $\times 2$	$256 \times 3 \times 3 \times 3$	1	$256 \times 16 \times 64 \times 64$	
Conv+RELU	$128 \times 3 \times 3 \times 3$	1	$128 \times 16 \times 64 \times 64$	
Conv+RELU	$128 \times 3 \times 3 \times 3$	1	$128 \times 18 \times 66 \times 66$	
Conv+RELU	$64 \times 3 \times 3 \times 3$	1	$64 \times 16 \times 64 \times 64$	
Upsample		$(1, \frac{1}{2}, \frac{1}{2})$	$64 \times 16 \times 128 \times 128$	Part2
Concat F1			$128 \times 16 \times 128 \times 128$	
Conv+RELU	$128 \times 3 \times 3 \times 3$	1	$128 \times 16 \times 128 \times 128$	
Conv+RELU	$64 \times 3 \times 3 \times 3$	1	$64 \times 16 \times 128 \times 128$	
Conv+RELU	$3 \times 3 \times 3 \times 3$	1	$3 \times 16 \times 128 \times 128$	Part3

Table 8: Architecture of Decoder where Stage represents different components of Decoder.

References

[1] Brandon Castellano. PySceneDetect library for shot boundary detection, 2022.

[2] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian Conference on Image Analysis*, 2003.

[3] Tomáš Souček and Jakub Lokoč. Transnet v2: an effective deep network architecture for fast shot transition detection. *arXiv preprint arXiv:2008.04838*, 2020.