David Kubala

Human Crowd Removal in Non-Stationary Camera Video by Semantic Segmentation and Neural Inpainting

Code Documentation

Code Structure

Table of Contents

main.py

	Function	inputs	outputs
check_init()	Check that the folder structure is correct, and that proper segmentation and inpainting models directories exist	None	None
main()	Begin rendering the video sequence stored at file location ‘input.mp4’	None	None

Renderer.py

	Function	inputs	Data type
render_video	Begin rendering each frame of the video file into its inpainted version in sequence	video_path	<*string*> The path to the input video file to render
		show_frames	<*bool*> Boolean controlling whether each rendered frame is shown at render time. Default True.
		save_frames	<*bool*> Boolean controlling whether frames are saved or discarded after render. Defaut True.
		starting_frame	<*int*> The frame that the renderer begins rendering with. Default 0.
		ending_frame	<*int*> The frame that the renderer ends rendering with. Default 2^{32}
		bilateral_solve	<*bool*> Boolean controlling whether the bilateral solver is applied to the resultant segmentation masks
render_frame	Render a provided individual frame into its inpainted version	frame	<*np.array*> (Shape HxWx3) A 3-channel color image of the frame to modify
		kernel_size	<*int*> The width of the input in pixels to the segmentation model. Default 473.
		frame_no	<*int*> Currently rendered frame's frame number in sequence
		seg_type	<*int*> Segmentation type. There are three segmentation types. 'Standard' for a naieve bucketing approach. 'Linear' for a more robust approach. 'Text_preprocessed' for segmentation masks preprocessed by multimodal transformers.
		show_images	<*bool*> Boolean controlling whether each rendered frame is shown at render time. Default True.

	Function	inputs	Data type
		save	<bool> Boolean controlling whether frames are saved or discarded after render. Default True.
		bilateral_solve	<*bool*> Boolean controlling whether the bilateral solver is applied to the resultant segmentation masks
median_ postprocess	Median postprocessing model. Collects inpainted frames following the full video processing and performs median postprocessing step to reduce flicker in the overall video.	capture_path	<*string*> A string conveying the path to the related video file
		num_steps_into_ past	<*int*> The number of steps the post-processing algorithm looks into the past in order to compute median values over time. Default 5.
load_text_ seg_masks	Loads the preprocessed segmentation masks from the multimodal transformer for text segmentation. Unpacks a large Numpy array into per-frame segmentation images.	None	*N/A*

Segmentation.py

	Function	inputs	Data type
pspnet_50_ADE_20K	Load the pretrained segmentation model to the SegmentationModel class.	None	N/A
average_linear_bucket	Master function for performing average linear bucketing algorithm. Creates a fully structured segmentation map.	frame	*<np.array>* (shape HxWx3) A 3-channel color image of the frame to be rendered.
		kernel_size	*<int>* Input dimension of the segmentation model. Size of square buckets cropped by model.
analyze_linear_bucket_sets	Analyze the set of image sub-regions generated by the linear bucketing collection algorithms using the neural segmentation model	buckets	*List[<np.array> (shape (ksize, ksize, 3))]* Set of 3-channel color images corresponding to image regions
collect_prediction_sets	Collect the set of predictions for the standard bucketing model	frame	*<np.array> (shape (H, W, 3))* 3 channel color image frame
		kernel_size	*<int>* Size of bucketed region areas
analyze_sets	Analyze the set of predictions from the standard bucketing model into a segmentation map	core	*List[<np.array> (shape (ksize, ksize, 3))]* Sets of images from the standard bucketing approach around the core of the image
		vertical	*List[<np.array> (shape (ksize, ksize, 3))]* Sets of images for analysis from the vertical edges of the frame
		horizontal	*List[<np.array> (shape (ksize, ksize, 3))]* Sets of images for analysis from the horizontal edges of the frame
		Corner	*List[<np.array> (shape (ksize, ksize, 3))]* Sets of images for analysis from the corner of the frame
stitch_image	Collect the set of predictions from the	frame	*<np.array> (shape (H, W, 3))* 3 channel color image frame

	Function	inputs	Data type
	standard bucketing system and stitch them back into one complete segmentation map	kernel_size	*<int>* Size of bucketed region areas
		core	*List[<np.array> (shape (ksize, ksize, 1))]* Sets of images from the standard bucketing approach around the core of the image
		vertical	*List[<np.array> (shape (ksize, ksize, 1))]* Sets of images for analysis from the vertical edges of the frame
		horizontal	*List[<np.array> (shape (ksize, ksize, 1))]* Sets of images for analysis from the horizontal edges of the frame
		Corner	*List[<np.array> (shape (ksize, ksize, 1))]* Sets of images for analysis from the corner of the frame
bucket_analyze	Master function for performing standard bucketing algorithm. Creates a fully structured segmentation map.	frame	*<np.array> (shape (H, W, 3))* 3 channel color image frame
		kernel_size	*<int>* Size of bucketed region areas
bilateral_solve_segmentation	Bilaterally solve the segmentation map using the reference image	frame	*<np.array> (shape (H, W, 3))* 3 channel color image frame
		segmented_frame	*<np.array> (shape (H, W, 1))* 3 channel color segmented frame

Inpainting.py

	Function	inputs	Data type
load_inpainting_model	Load the pretrained inpainting model to the InpaintingModel class.	None	N/A
predict_inpaint	Take in an image and an image mask, return a fully inpainted image at the areas described by the inpainting mask.	Image	<*np.array*> (shape HxWx3) A 3-channel color image of the frame to be rendered.
		Mask	<*np.array*> (shape HxWx1) A binary mask image of the areas to inpaint in the frame