ImageEditor

```
gradio.ImageEditor(···)
```

Description

Creates an image component that can be used to upload and edit images (as an input) or display images (as an output).

Behavior

As input: passes the uploaded images as a dictionary with keys: `background`, `layers`, and `composite`. The values corresponding to `background` and `composite` are images, while `layers` is a list of images. The images are of type PIL.Image, np.array, or str filepath, depending on the `type` parameter.

As output: expects a dictionary with keys: `background`, `layers`, and `composite`. The values corresponding to `background` and `composite` should be images or None, while `layers` should be a list of images. Images can be of type PIL.Image, np.array, or str filepath/URL. Or, the value can be simply a single image, in which case it will be used as the background.

Initialization

Parameter	Description
`value` *EditorValue | ImageType | None* **default: None**	Optional initial image(s) to populate the image editor. Should be a dictionary with keys: `background`, `layers`, and `composite`. The values corresponding to `background` and `composite` should be images or None, while `layers` should be a list of images. Images can be of type PIL.Image, np.array, or str filepath/URL. Or, the value can be a callable, in which case the function will be called whenever the app loads to set the initial value of the component.

ımeter	Description
`height` *int | str | None* **default: None**	The height of the displayed images, specified in pixels if a number is passed, or in CSS units if a string is passed.
`width` *int | str | None* **default: None**	The width of the displayed images, specified in pixels if a number is passed, or in CSS units if a string is passed.
`image_mode` *Literal[('1', 'L', 'P', 'RGB', 'RGBA', 'CMYK', 'YCbCr', 'LAB', 'HSV', 'I', 'F')]* **default: "RGBA"**	"RGB" if color, or "L" if black and white. See https://pillow.readthedocs.io/en/stable/handbook/concepts.html for other supported image modes and their meaning.
`sources` *Iterable[Literal[('upload', 'webcam', 'clipboard')]]* **default: ('upload', 'webcam', 'clipboard')**	List of sources that can be used to set the background image. "upload" creates a box where user can drop an image file, "webcam" allows user to take snapshot from their webcam, "clipboard" allows users to paste an image from the clipboard.
`type` *Literal[('numpy', 'pil', 'filepath')]* **default: "numpy"**	The format the images are converted to before being passed into the prediction function. "numpy" converts the images to numpy arrays with shape (height, width, 3) and values from 0 to 255, "pil" converts the images to PIL image objects, "filepath" passes images as str filepaths to temporary copies of the images.
`label` *str | None* **default: None**	The label for this component. Appears above the component and is also used as the header if there are a table of examples for this component. If None and used in a `gr.Interface`, the label will be the name of the parameter this component is assigned to.
`every` *float | None* **default: None**	If `value` is a callable, run the function 'every' number of seconds while the client connection is open. Has no effect otherwise. Queue must be enabled. The event can be accessed (e.g. to cancel it) via this component's .load_event attribute.

ımeter	Description
`show_label` *bool | None* **default: None**	if True, will display label.
`show_download_button` *bool* **default: True**	If True, will display button to download image.
`container` *bool* **default: True**	If True, will place the component in a container - providing some extra padding around the border.
`scale` *int | None* **default: None**	relative width compared to adjacent Components in a Row. For example, if Component A has scale=2, and Component B has scale=1, A will be twice as wide as B. Should be an integer.
`min_width` *int* **default: 160**	minimum pixel width, will wrap if not sufficient screen space to satisfy this value. If a certain scale value results in this Component being narrower than min_width, the min_width parameter will be respected first.
`interactive` *bool | None* **default: None**	if True, will allow users to upload and edit an image; if False, can only be used to display images. If not provided, this is inferred based on whether the component is used as an input or output.
`visible` *bool* **default: True**	If False, component will be hidden.
`elem_id` *str | None* **default: None**	An optional string that is assigned as the id of this component in the HTML DOM. Can be used for targeting CSS styles.

ımeter	Description
`elem_classes` *list[str] | str | None* **default: None**	An optional list of strings that are assigned as the classes of this component in the HTML DOM. Can be used for targeting CSS styles.
`render` *bool* **default: True**	If False, component will not render be rendered in the Blocks context. Should be used if the intention is to assign event listeners now but render the component later.
`mirror_webcam` *bool* **default: True**	If True webcam will be mirrored. Default is True.
`show_share_button` *bool | None* **default: None**	If True, will show a share icon in the corner of the component that allows user to share outputs to Hugging Face Spaces Discussions. If False, icon does not appear. If set to None (default behavior), then the icon appears if this Gradio app is launched on Spaces, but not otherwise.
`crop_size` *tuple[int | float, int | float] | str | None* **default: None**	The size of the crop box in pixels. If a tuple, the first value is the width and the second value is the height. If a string, the value must be a ratio in the form `width:height` (e.g. "16:9").
`transforms` *Iterable[Literal['crop']]* **default: ('crop',)**	The transforms tools to make available to users. "crop" allows the user to crop the image.
`eraser` *Eraser | None | Literal[False]* **default: None**	The options for the eraser tool in the image editor. Should be an instance of the `gr.Eraser` class, or None to use the default settings. Can also be False to hide the eraser tool.
`brush` *Brush | None | Literal[False]* **default: None**	The options for the brush tool in the image editor. Should be an instance of the `gr.Brush` class, or None to use the default settings. Can also be False to hide the brush tool, which will also hide the eraser tool.

tcuts

s	Interface String Shortcut	Initialization
`gradio.ImageEditor`	"imageeditor"	Uses default values
`gradio.Sketchpad`	"sketchpad"	Uses sources=(), brush=Brush(colors= ["#000000"], color_mode="fixed")
`gradio.Paint`	"paint"	Uses sources=()
`gradio.ImageMask`	"imagemask"	Uses brush=Brush(colors= ["#000000"], color_mode="fixed")

Demos

image_editor

```
import gradio as gr
import time


def sleep(im):
    time.sleep(5)
    return [im["background"], im["layers"][0], im["layers"][1], im["composite"]]


with gr.Blocks() as demo:
    im = gr.ImageEditor(
```

Event Listeners

Description

Event listeners allow you to capture and respond to user interactions with the UI components you've defined in a Gradio Blocks app. When a user interacts with an element, such as changing a slider value or uploading an image, a function is called.

Supported Event Listeners

The `ImageEditor` component supports the following event listeners. Each event listener takes the same parameters, which are listed in the Event Arguments table below.

Listener	Description
`gradio.ImageEditor.clear(fn, ···)`	This listener is triggered when the user clears the ImageEditor using the X button for the component.
`gradio.ImageEditor.change(fn, ···)`	Triggered when the value of the ImageEditor changes either because of user input (e.g. a user types in a textbox) OR because of a function update (e.g. an image receives a value from the output of an event trigger). See `.input()` for a listener that is only triggered by user input.
`gradio.ImageEditor.select(fn, ···)`	Event listener for when the user selects or deselects the ImageEditor. Uses event data gradio.SelectData to carry `value` referring to the label of the ImageEditor, and `selected` to refer to state of the ImageEditor. See EventData documentation on how to use this event data
`gradio.ImageEditor.upload(fn, ···)`	This listener is triggered when the user uploads a file into the ImageEditor.

Event Arguments

Parameter	Description
`fn` *Callable | None | Literal['decorator']* **default: "decorator"**	the function to call when this event is triggered. Often a machine learning model's prediction function. Each parameter of the function corresponds to one input component, and the function should return a single value or a tuple of values, with each element in the tuple corresponding to one output component.
`inputs` *Component | list[Component] | set[Component] | None* **default: None**	List of gradio.components to use as inputs. If the function takes no inputs, this should be an empty list.

Parameter	Description
`outputs` *Component | list[Component] | None* **default: None**	List of gradio.components to use as outputs. If the function returns no outputs, this should be an empty list.
`api_name` *str | None | Literal[False]* **default: None**	defines how the endpoint appears in the API docs. Can be a string, None, or False. If set to a string, the endpoint will be exposed in the API docs with the given name. If None (default), the name of the function will be used as the API endpoint. If False, the endpoint will not be exposed in the API docs and downstream apps (including those that `gr.load` this app) will not be able to use this event.
`scroll_to_output` *bool* **default: False**	If True, will scroll to output component on completion
`show_progress` *Literal[('full', 'minimal', 'hidden')]* **default: "full"**	If True, will show progress animation while pending
`queue` *bool | None* **default: None**	If True, will place the request on the queue, if the queue has been enabled. If False, will not put this event on the queue, even if the queue has been enabled. If None, will use the queue setting of the gradio app.
`batch` *bool* **default: False**	If True, then the function should process a batch of inputs, meaning that it should accept a list of input values for each parameter. The lists should be of equal length (and be up to length `max_batch_size`). The function is then ***required*** to return a tuple of lists (even if there is only 1 output component), with each list in the tuple corresponding to one output component.

Parameter	Description
`max_batch_size` *int* **default: 4**	Maximum number of inputs to batch together if this is called from the queue (only relevant if batch=True)
`preprocess` *bool* **default: True**	If False, will not run preprocessing of component data before running 'fn' (e.g. leaving it as a base64 string if this method is called with the `Image` component).
`postprocess` *bool* **default: True**	If False, will not run postprocessing of component data before returning 'fn' output to the browser.
`cancels` *dict[str, Any] | list[dict[str, Any]] | None* **default: None**	A list of other events to cancel when this listener is triggered. For example, setting cancels=[click_event] will cancel the click_event, where click_event is the return value of another components .click method. Functions that have not yet run (or generators that are iterating) will be cancelled, but functions that are currently running will be allowed to finish.
`every` *float | None* **default: None**	Run this event 'every' number of seconds while the client connection is open. Interpreted in seconds. Queue must be enabled.
`trigger_mode` *Literal[('once', 'multiple', 'always_last')] | None* **default: None**	If "once" (default for all events except `.change()`) would not allow any submissions while an event is pending. If set to "multiple", unlimited submissions are allowed while pending, and "always_last" (default for `.change()` event) would allow a second submission after the pending event is complete.
`js` *str | None* **default: None**	Optional frontend js method to run before running 'fn'. Input arguments for js method are values of 'inputs' and 'outputs', return should be a list of values for output components.

Parameter	Description
`concurrency_limit` *int | None | Literal['default']* **default: "default"**	If set, this is the maximum number of this event that can be running simultaneously. Can be set to None to mean no concurrency_limit (any number of this event can be running simultaneously). Set to "default" to use the default concurrency limit (defined by the `default_concurrency_limit` parameter in `Blocks.queue()`, which itself is 1 by default).
`concurrency_id` *str | None* **default: None**	If set, this is the id of the concurrency group. Events with the same concurrency_id will be limited by the lowest set concurrency_limit.
`show_api` *bool* **default: True**	whether to show this event in the "view API" page of the Gradio app, or in the ".view_api()" method of the Gradio clients. Unlike setting api_name to False, setting show_api to False will still allow downstream apps to use this event. If fn is None, show_api will automatically be set to False.