plotly

Star 23,446

Dash Python > ***Dash Canvas***

Introduction to dash-canvas

Note: dash-canvas is a legacy package. The recommended way to annotate images is to use the **drawing tools of plotly figures**.

`dash-canvas` is a module for image annotation and image processing using Dash. It provides both the `DashCanvas` object for drawing and annotations on images, and a set of utility functions to process images using the annotations.

`dash-canvas` can be used in various fields in which user interaction with images is required, such as quality control in industry, identification and segmentation of cells or organs in life and medical sciences, quantification of phases in materials and geosciences, construction of training sets for machine learning, etc.

Install `dash-canvas` with

```
pip install dash-canvas
```

The source is on GitHub at **plotly/dash-canvas**.

DashCanvas: a canvas object for annotations

Let's get started with a simple canvas object.

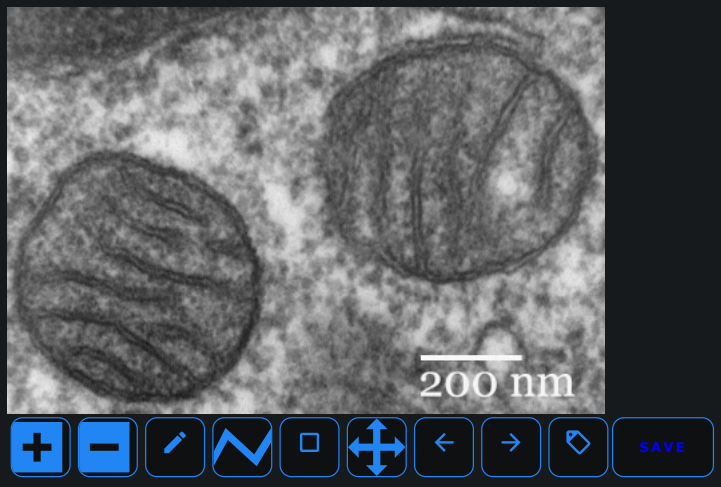
```
from dash import Dash, html
from dash_canvas import DashCanvas

app = Dash()
app.config.suppress_callback_exceptions = True

app.layout = html.Div([
    html.H5('Press down the left mouse button and draw inside the canvas'),
    DashCanvas(id='canvas_101')
    ])


if __name__ == '__main__':
    app.run(debug=True)
```

Press down the left mouse button and draw inside the canvas

You can draw inside the object with the freehand tool, and use the tool buttons to draw lines, zoom in and out, pan, select objects and move them inside the canvas.

`DashCanvas` comes with a set of properties which can be adjusted to control the geometry of the canvas, the default tool and its properties. You can pass a background image either as a filename (`filename` property) or as a data string (`image_content` property); more examples below.

```
from dash import Dash, html
from dash_canvas import DashCanvas

app = Dash()

filename = 'https://raw.githubusercontent.com/plotly/datasets/master/mitochondria.jpg'
canvas_width = 500


app.layout = html.Div([
    DashCanvas(id='canvaas_image',
               tool='line',
               lineWidth=5,
               lineColor='red',
               filename=filename,
               width=canvas_width)
    ])


if __name__ == '__main__':
    app.run(debug=True)
```



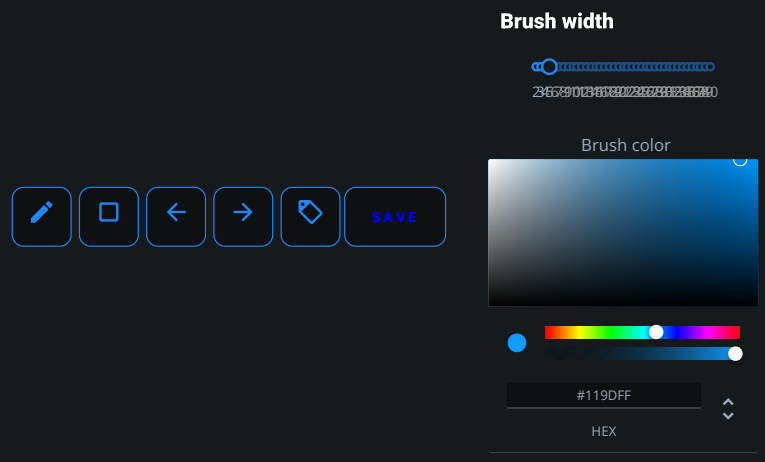
The height of the canvas is adjusted automatically by keeping the aspect ratio of the background image.

Basic callbacks to modify DashCanvas properties

Like any Dash component, the properties of a `DashCanvas` can be modified by other components, via callbacks. Please be sure to have read about **Basic Callbacks** in the Dash Fundamentals.

```
from dash_canvas import DashCanvas
from dash import Dash, html, dcc, Input, Output, callback
import dash_daq as daq

filename = 'https://www.publicdomainpictures.net/pictures/60000/nahled/flower-outline-coloring
canvas_width = 300

app = Dash()

app.layout = html.Div([
```

```
        html.Div([
            DashCanvas(
                id='canvaas-color',
                width=canvas_width,
                filename=filename,
                hide_buttons=['line', 'zoom', 'pan'],
                )
                ], className="six columns"),
        html.Div([
            html.H6(children=['Brush width']),
            dcc.Slider(
                id='bg-width-slider',
                min=2,
                max=40,
                step=1,
                value=5
            ),
            daq.ColorPicker(
                id='color-picker',
                label='Brush color',
                value=dict(hex='#119DFF')
            ),
        ], className="three columns"),
        ])


@callback(Output('canvaas-color', 'lineColor'), Input('color-picker', 'value'))
def update_canvas_linecolor(value):
    if isinstance(value, dict):
        return value['hex']
    else:
        return value


@callback(Output('canvaas-color', 'lineWidth'), Input('bg-width-slider', 'value'))
def update_canvas_linewidth(value):
    return value


if __name__ == '__main__':
    app.run(debug=True)
```



In the example above, a slider `dcc.Slider` and a color picker `daq.ColorPicker` are used to adjust the width and color of the drawing brush. We just created an image coloring tool in a few lines of code! You can learn more about available components in the **component libraries** section of the Dash documentation. Also note that the set of available buttons has been restricted through the `hide_buttons` properties, in order to keep the app design simple.

Retrieving the geometry of annotations and using utility functions

The geometry of annotations can be retrieved by pressing the bottom-right button of the `DashCanvas`. This button is called "Save" by default; the name can be customized through the `goButtonTitle` property. This

button updates the `json_data` property of `DashCanvas`, which is a JSON string with information about the background image and the geometry of annotations.

```
from dash import Dash, html, dash_table, Input, Output, callback
from dash.exceptions import PreventUpdate
from dash_canvas import DashCanvas
import json

app = Dash()

filename = 'https://raw.githubusercontent.com/plotly/datasets/master/mitochondria.jpg'
canvas_width = 500

columns = ['type', 'width', 'height', 'scaleX', 'strokeWidth', 'path']

app.layout = html.Div([
    html.H6('Draw on image and press Save to show annotations geometry'),
    DashCanvas(id='annot-canvas',
               lineWidth=5,
               filename=filename,
               width=canvas_width,
               ),
    dash_table.DataTable(id='canvaas-table',
              style_cell={'textAlign': 'left'},
              columns=[{"name": i, "id": i} for i in columns]),
    ])


@callback(Output('canvaas-table', 'data'),
              Input('annot-canvas', 'json_data'))
def update_data(string):
    if string:
        data = json.loads(string)
    else:
        raise PreventUpdate
    return data['objects'][1:]


if __name__ == '__main__':
    app.run(debug=True)
```

Draw on image and press Save to show annotations geometry

type	width	height	scaleX	strokeWidth	path

You can either write custom functions to parse the JSON string, or use the utility functions included in the `dash_canvas` package. In particular, `dash_canvas.utils.parse_json_string` returns a binary mask with non-zero pixels displaying the annotations:

```
import numpy as np
from skimage import io
```

```
from dash import Dash, html, Input, Output, callback
from dash.exceptions import PreventUpdate
from dash_canvas import DashCanvas
from dash_canvas.utils import array_to_data_url, parse_jsonstring

app = Dash()

filename = 'https://raw.githubusercontent.com/plotly/datasets/master/mitochondria.jpg'

canvas_width = 300

app.layout = html.Div([
    html.H6('Draw on image and press Save to show annotations geometry'),
    html.Div([
        DashCanvas(
            id='canvas',
            lineWidth=5,
            filename=filename,
            width=canvas_width,
        ),
    ], className="five columns"),
    html.Div(html.Img(id='my-iimage', width=300), className="five columns"),
    ])


@callback(Output('my-iimage', 'src'),
              Input('canvas', 'json_data'))
def update_data(string):
    if string:
        mask = parse_jsonstring(string, io.imread(filename, as_gray=True).shape)
    else:
        raise PreventUpdate
    return array_to_data_url((255 * mask).astype(np.uint8))


if __name__ == '__main__':
    app.run(debug=True)
```

Draw on image and press Save to show annotations geometry

The above example uses the `array_to_data_url` utility function to transform a `NumPy` array into an image data string.

Finally, `dash-canvas` provides utility functions to process images given the binary mask derived from annotations:

```
from dash import Dash, html, Input, Output, callback
from dash.exceptions import PreventUpdate
from dash_canvas import DashCanvas
from dash_canvas.utils import (array_to_data_url, parse_jsonstring,
                               watershed_segmentation)
from skimage import io, color, img_as_ubyte

app = Dash()
filename = 'https://raw.githubusercontent.com/plotly/datasets/master/mitochondria.jpg'
```

```
canvas_width = 300

app.layout = html.Div([
    html.H6('Annotate the two objects and the background'),
    html.Div([
        DashCanvas(
            id='segmentation-canvas',
            lineWidth=5,
            filename=filename,
            width=canvas_width,
        ),
    ], className="five columns"),
    html.Div(html.Img(id='segmentation-iimg', width=300), className="five columns"),
    ])


@callback(Output('segmentation-iimg', 'src'),
              Input('segmentation-canvas', 'json_data'))
def segmentation(string):
    if string:
        mask = parse_jsonstring(string, io.imread(filename, as_gray=True).shape)
        seg = watershed_segmentation(io.imread(filename, as_gray=True), mask)
        src = color.label2rgb(seg, image=io.imread(filename, as_gray=True))
    else:
        raise PreventUpdate
    return array_to_data_url(img_as_ubyte(src))


if __name__ == '__main__':
    app.run(debug=True)
```

Annotate the two objects and the background

These functions rely on **scikit-image** to process arrays as images. Here we used the **watershed algorithm** from scikit-image.

Updating the background image

The background image can be updated thanks to the `image_content` property (a `str`), for example using the `contents` property of `dcc.Upload` (an "open file" dialog). Updating `image_content` triggers the update of the `json_data` property containing the annotations.

More examples

A gallery of examples using `DashCanvas` is available at **plotly/canvas-portal**.

Dash Python ***> Dash Canvas***