Origami Documentation

Release

Avais Pagarkar, Ashish Chaudhary, Harsh Agarwal, Deshraj Yadav

Aug 26, 2017

Contents

CHAPTER 1

Introduction

Origami helps you build a web based demo out of Machine Learning Code. Origami provides a library to allow you to integrate your code with Origami and a web app to view the demos.

Origami Web app

The web app provides an interface to interact with demos created by using the library.

Configuration

A root user is created for an installation of Origami. Origami requires some configuration before it can connect to Github.

- **Step 1: Create a Github developer application**

Go to Github Developer Applications page and "Register a new application" and enter the application details.

Application Name Choose a suitable name for your application.

Homepage URL This is the base-URL of Origami application. This is the URL where this webapp is running. For *local deployments* on default port, it is "http://localhost:8000/". For the current Origami installation, it is "http://origami.cloudcv.org/".

Application description Choose a suitable description for your application.

Authorization callback URL This URL is Homepage URL + "/auth/github/login/callback". For *local deployments* on default port, it is "http://localhost:8000/auth/github/login/callback". For the current Origami installation, it is "http://origami.cloudcv.org/auth/github/login/callback".

Now click on the "Register application" button to register this application. On the subsequent page, note down the Client ID and Client Secret.

- **Step 2: Input details of Github application on initial setup page.**

This page is visible when the owner of this Origami installation runs this application for the first time. Following inputs are required:

Root user's Github username This is the Github username of the person who is the owner of this Origami installation.

Github Client ID This is the "Client ID" noted down in Step 1.

Github Client Secret This is the "Client Secret" noted down in Step 1.

Application IP address This is the IP address (or domain name) where the Origami webapp is running. It is pre-filled with "0.0.0.0".

Allow new users A root user can forbid other users (those who want to make demos) of this application from using it. In that case, only the root user can login and create demos.

Is this deployment by CloudCV Check this option if the deployment is by CloudCV. This adds some customizations.

Origami requires a Dropbox App key if you intend to allow users to download/upload from Dropbox

- **Step 3: Create a Dropbox developer application (optional)**

Do this step if you intend to allow users to upload images from their dropbox as input to the demos. Go to Dropbox Developers Page page and click on "Create your app".

In step 1, choose the *Dropbox API*

In step 2, select *Full Dropbox*. This allows the app access to the users full dropbox.

Now, Name your app and click on create app. The name needs to be *unique*.

Once the app is created, Dropbox redirects you to its configuration page. Note down the App Key

Also, find *Chooser/Saver domains* and add the domains 0.0.0.0 localhost origami.cloudcv.org

Paste the App Key into *outCalls/config.js*.

Creating a new app

- **Step 1: Login and provision an App**

After the initial setup, click on Create a Demo button on the homepage. This takes you to github for login. Authorize the application there when asked. Upon successful login, you are taken to the user profile that lists all his deployed apps.

Click on the + button here to create a new application. This takes you to the Registration page.

- **Step 2: Create an App**

Following inputs are required:

Appname This is the name of your application. This appears on the top of demo page.

IP of service This is the IP address of the system that will be running your machine learning code using Origami-lib. For local deployments, it is 0.0.0.0.

Port for service This is the preferred port for the service (machine learning code). This port must be free for Origami-lib to work.

Description (optional) Description for your application. This will be displayed below the application name of demo page.

Show Terminal on demo page This displays a Terminal style text box below the I/O components on the demo page. Additional data can be sent to this terminal using Origami-lib.

If an *error box* says "This webapp cannot be reached on it's public IP", you need to check the "Webapp is running locally" checkbox. Checking this checkbox will make the webapp check local connectivity to itself.

If you see a *green tick* symbol next to the token, your app is configured correctly. Copy this token for use in Origami-lib and click on "Save" button.

If you see a *red hand* symbol next to the token, your app is configured incorrectly. You may not be able to connect to your app.

I/O Components

The following procedure applies to both Input and Ouput components.

- **Step 1: Configuring the Input component**

After registering the application, you are taken to the input component selection page.

Choose the kind of Input component your machine learning code requires. If your code requires processing 1 (or more) images, you need **Image Input** component. If it requires both an image and a text input with it, you need **Text Image Input** component and so on.

After choosing the Input component of your choice, click on **Modify** button on the component you want. This opens the modification modal for that component. Each component can have different type of configuration. For example, **Text Input component** has an **Add Label** option that adds a new text field for input. The text entered here appears in the placeholder for the field in the Input component on demo page. You can add or delete any number of fields in Input component. Press **OK** to save the component.

- **Step 2: Previewing the Input component**

After configuring the Input component, you can preview it by pressing the **Preview** button on the component. This opens a modal that shows how the Input component will look like on the demo page.

If you are satisfied with the preview, you can click on **OK** and move to the next step. Otherwise you can edit the Input component and see the preview again.

- **Step 3: Using the Input component**

Once you are satisfied with the preview, you can press the Use button on the component to add to the demo page.

You can come back to this page anytime from the user profile page by clicking on **Modify** on the project and selecting **Input** thereafter.

Refreshing the page or going back refreshes the app-state. In that case, you have to go back to the Input component page from the user profile page by clicking **Modify** and selecting **Input** thereafter.

Publish a demo

A demo is published as soon as the app is registered.

The demo can be accessed by clicking on **Demo** button on the app on user profile page. A shortened URL for the demo can be created from the user profile page by clicking **Get permalink** on the app.

Modify/Delete an App

- **Modifying**

Registration data and I/O components can be modified later on as well from the **user profile page** by clicking on **Modify** button on the component and then in the modal that appears:

Modify Registration data Click on "Metadata"

Modify Input data Click on "Input"

Modify Output data Click on "Output"

- **Deleting**

An application can be deleted by visiting the **user profile** page and clicking on **Delete** button on the component.

Library

The origami-lib library lets you integrate your machine learning code with the Origami web app. All it takes is a couple of function calls. Let's get started!

Configuration

origami-lib supports python2 (on OSX and Linux) only as of now. Download origami.py from Github to your projects root directory (where the launcher python script is).

origami-lib requires installation of some additional packages:

```
sudo apt install python-pip python-dev python-numpy python-opencv
```

origami-lib has a file requirements.txt that contains dependency python packages:

```
pip install -r requirements.txt
```

Register a new app

origami-lib registration requires a TOKEN from the Origami webapp. This TOKEN can be copied from the registration page of the application. Or by clicking Get Token on the app on user profile page.

For a complete example, see this Gist. origami.py is imported to the launcher python script:

```
from origami import origami
```

origami-lib is registered with:

```
app = origami.register($TOKEN)
```

Note that $TOKEN here is replaced by the TOKEN obtained from Origami webapp. origami-lib requires a main function that is executed when a request is received. This function must be decorated with both:

```
@origami.crossdomain
@app.listen()
```

This function must return 'OK' in the end. Lastly, it should have a statement that starts the app,:

```
app.run()
```

Input functions

- ***getTextArray()*:**

    Arguments: None

Returns: Array of text elements

This function works with: Text Input Component

Text Image Input Component An example can bee seen at this gist.

- ***getImageArray()*:**

  Arguments:

  Mode (String):

  – file_path

  Returns an array of local paths to the uploaded images. This is the default mode. An example can be seen at this gist.

  – numpy_array

  Returns an Array/Tuple of the uploaded images as "numpy array" elements (like the image objects used in OpenCV) An example can be seen at this gist.

  Returns: Array of "local path of images" in text obtained after saving images to disk receievd from Origami webapp.

  This function works with: Image Input Component

  Text Image Input Component An example can be seen at this gist.

- Hybrid components that require multiple types of Input (like Text Image input component)

  Such components require usage of multiple functions at once. For example, for Text Image Input component,:

```
all_text = origami.getTextArray()
all_image_paths = origami.getImageArray()
```

Output functions

- ***sendTextArray()*** sendTextArray injects an array of text into fields in Output component.

  Arguments: Array/Tuple of text elements

  Returns: None

  This function works with: Text Output Component An example can bee seen at this gist.

- ***sendImageArray()*** origami.sendImageArray() injects an array of images into fields in Output component.

  Arguments: Array/Tuple of image data objects. These data objects can be of multiple types depending upon the mode.

  Mode (String)

  – file_path

  Array/Tuple of "local path of images on the disk" in text

  An example can be seen at this gist.

  – numpy_array

  Array/Tuple of "numpy array" elements (like the image objects used in OpenCV)

  An example can be seen at this gist.

Returns: None

This function works with: Image Output Component

- ***sendGraphArray()*** origami.sendGraphArray() injects an array of plot data into graph in Output component.

  Arguments: Array/Tuple of "arrays of plot dictionaries". Each entry in these arrays of plot dictionaries have two keys, 'x' and 'y' which take different values depending upon the type of graph.

  Type of Graph

  – Bar Graph

    x: INTEGER y: INTEGER 'x' and 'y' correspond to X-Axis and Y-Axis on the graph.

    An example can be seen at this gist.

  – Scatter Graph

    x: INTEGER y: INTEGER 'x' and 'y' correspond to X-Axis and Y-Axis on the graph.

    An example can be seen at this gist.

  – Area Graph

    x: INTEGER y: INTEGER 'x' and 'y' correspond to X-Axis and Y-Axis on the graph.

    An example can be seen at this gist.

  – Pie Chart

    x: STRING y: INTEGER 'x' correponds to the sectio name, 'y' correponds to share of that section in the pie.

    An example can be seen at this gist.

  Returns: None

  This function works with: Bar Graph Output Component Scatter Graph Component Area Graph Component Pie Chart Component

Terminal functions

To use the terminal, it must first be enabled for the app on its registration page. Go to user profile page and click on Modify button on the app then select Metadata thereafter to go to registration page. Tick the Show Terminal of demo page checkbox here.

sendTextArrayToTerminal() origami.sendTextArrayToTerminal() allows you to send text feedback to a terminal style interface on the demo page. This text data can be sent at any time (before or after the request processing is complete). Each element of the array will be put on a newline in the terminal.

Arguments: Array/Tuple of text elements

Returns: None

This function works with: All components An example can bee seen at this gist.