

VideoLab Manual

VideoLab is a tool for analyzing a series of consecutive images, such as videos. This tool is specifically designed for computer visioners, who needs to closely watch a series of images back and forth and do some basic processing to the images.

This tool supports the process of a series of image organized in a video file, a super ppm file, or a folder. These files can be from your local machine or from a remote server. With this tool, you can open as many image series as you want, browse them conveniently, process them with simple scripts, or convert them to different formats.

Table of Contents

| | |
|-------------------------------|-----------|
| Installation | 2 |
| Usage | 3 |
| Launch VideoLab | 3 |
| Open an image series | 3 |
| Browse the image series..... | 5 |
| Process images..... | 5 |
| Label images | 7 |
| Save image series | 8 |
| Close image series | 9 |
| Scripts..... | 10 |
| Advanced Topics | 11 |
| Create your own command | 11 |
| Use your own server | 11 |

Installation

To deploy VideoLab to your local machine, git clone the repository to your hard drive. You need to install conda (<https://conda.io/projects/conda/en/latest/user-guide/install/index.html>) and rsync (brew install rsync / sudo apt install rsync). Then, you can follow the commands listed below to establish the conda environment:

- conda create -n your_env_name python=3.8
- conda activate your_env_name
- pip install PyQt5
- pip install numpy
- pip install Pillow
- pip install opencv-python
- pip install opencv-contrib-python
- pip install decord
- pip install pexpect

To deploy VideoLab to Gypsum, you need to do the following steps: **(UNDER CONSTRUCTION)**

- git clone the repository to Gypsum. Copy the path you save the repository in the server as processor_path,
- run mkdir -p tmp/scripts in the cloned folder
- Download all the third party modules to utils/transformers/thirdparty,
- change configs/serverConfigs/gypsum.json to fit your need. More specifically, type your server address and your user name into field “server” and “username”, respectively. Paste processor_path to the corresponding field. You don’t need to edit template_path and its associated file since it has been adapted to Gypsum already,
- create the environment following what you did on your local machine. Notice: to use the third party module, you may need to install more packages. Please refer to section Advanced Topics.

If you are using other servers, please refer to section Advanced Topics.

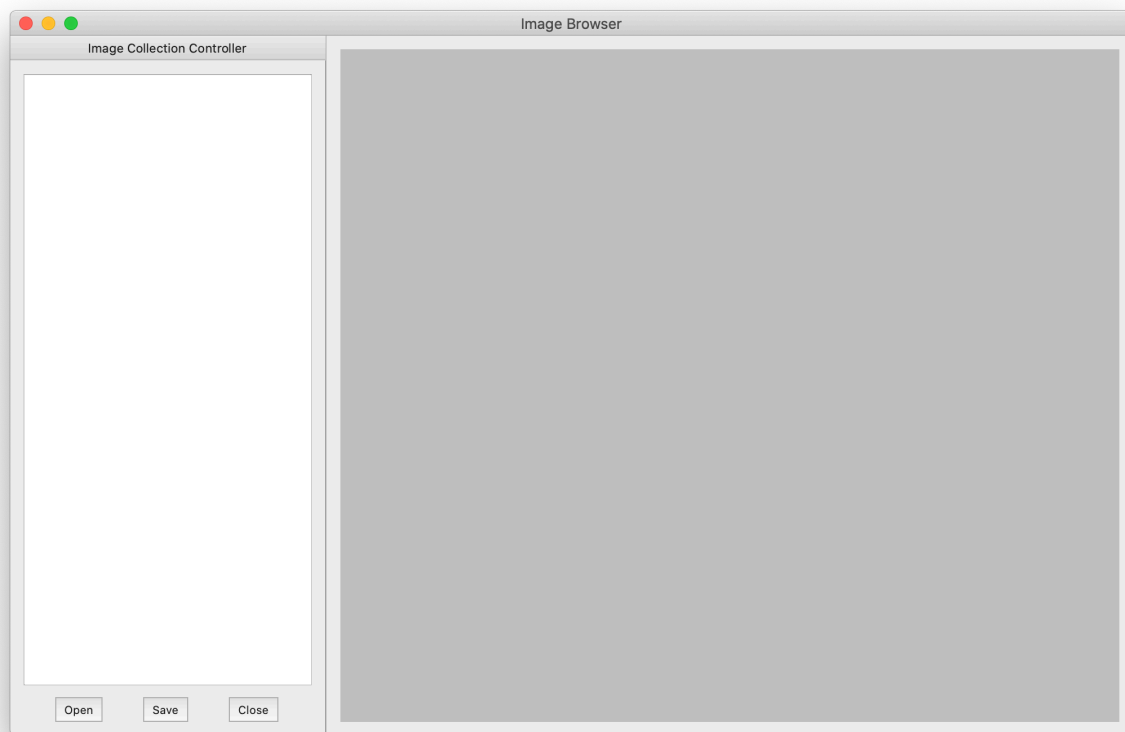
Usage

Launch VideoLab

To launch video lab, you need to open a terminal, activate the conda environment you deployed, and run the launch program:

- `conda activate your_env_name`
- `cd` to the root folder of the repository
- `python ImageBrowser.py`

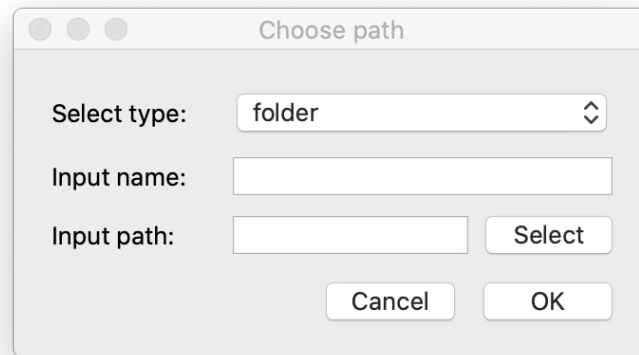
You will see a window like the following image:



The window has two parts: the left part is a dock widget which shows the image series you have opened. The right part is the area where you browse the image series. You can drag the left part to wherever you like or make it thinner or wider.

Open an image series

To open an image series, click the open button on the left bottom corner of the dock widget. You will see a dialog like this:

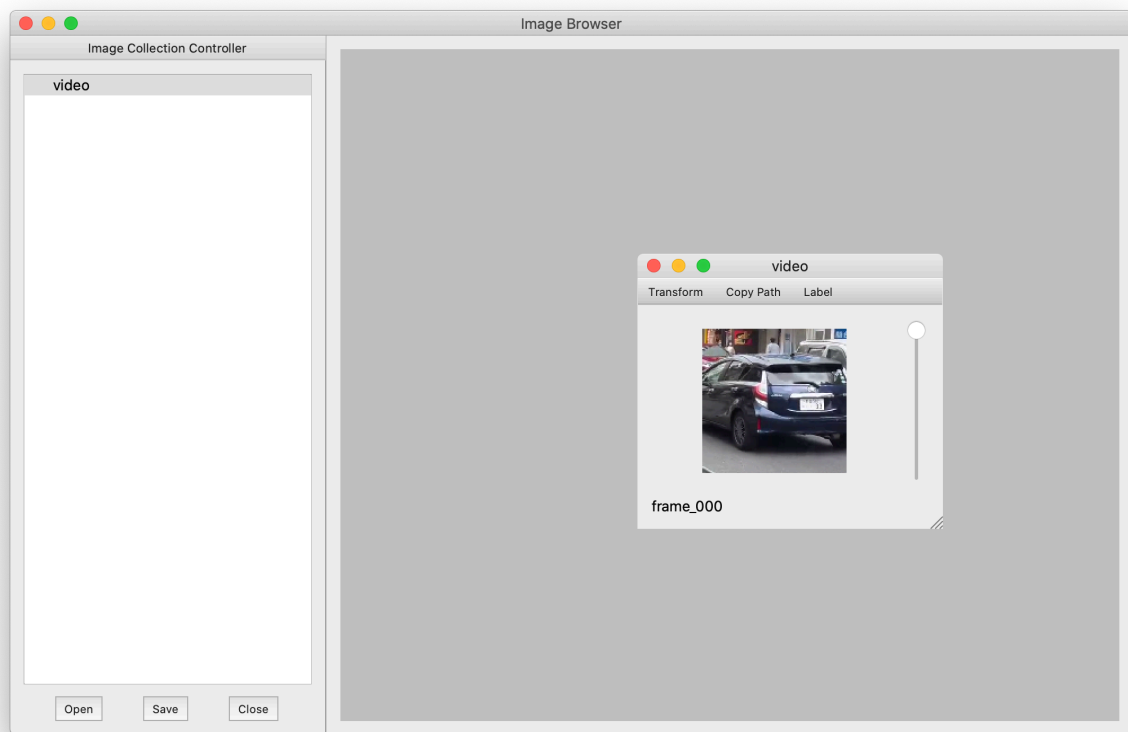


Select the format of the image series you want to open, input a unique nickname of the image series, which will be used to refer to the image series you opened, and type in the path of the image series (or use the file dialog to select a path by clicking the “select” button).

Currently, we support three types: folder, video, and ppm. Folder means a folder of images. If you select “folder” as the type, the corresponding path should be a folder which contains at least one image. We support video of mp4 and avi format for now, so the path should be a video file of those two formats. PPM refers to super PPM format, which is a special format we designed for extending the original PPM format. Super PPM format can store and show a series of images in a compact way and it can be opened like a normal PPM file. The corresponding path should be a file of ppm format if we select “ppm” as the type.

You can also input a path from the server to load image series directly from your server. An example of such input path is “sam@gypsum.cs.umass.edu:~/images.mp4”. This input format is of the same as the scp command.

Clicking OK button, the image series will be opened to the dock widget like the following screenshot (VideoLab may ask you for the password of the server when the input path is on the server end). Double click the nickname of the image series, you can see a subwindow is opened for you to display the image series.

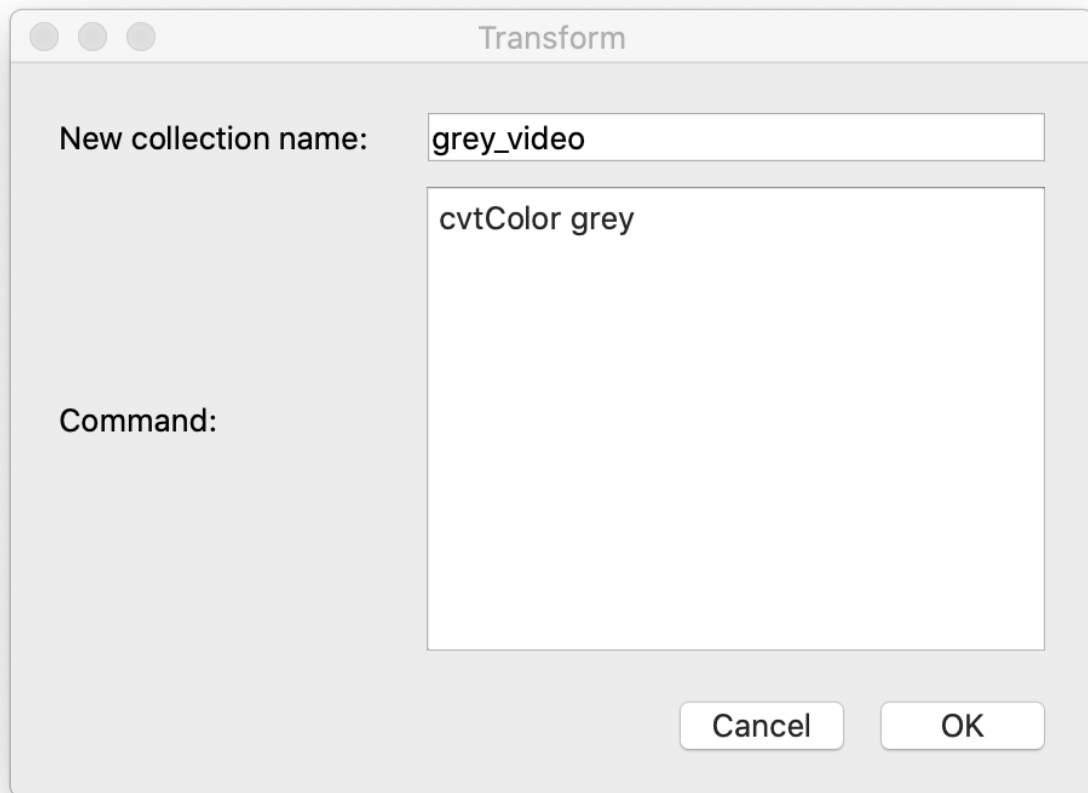


[Browse the image series](#)

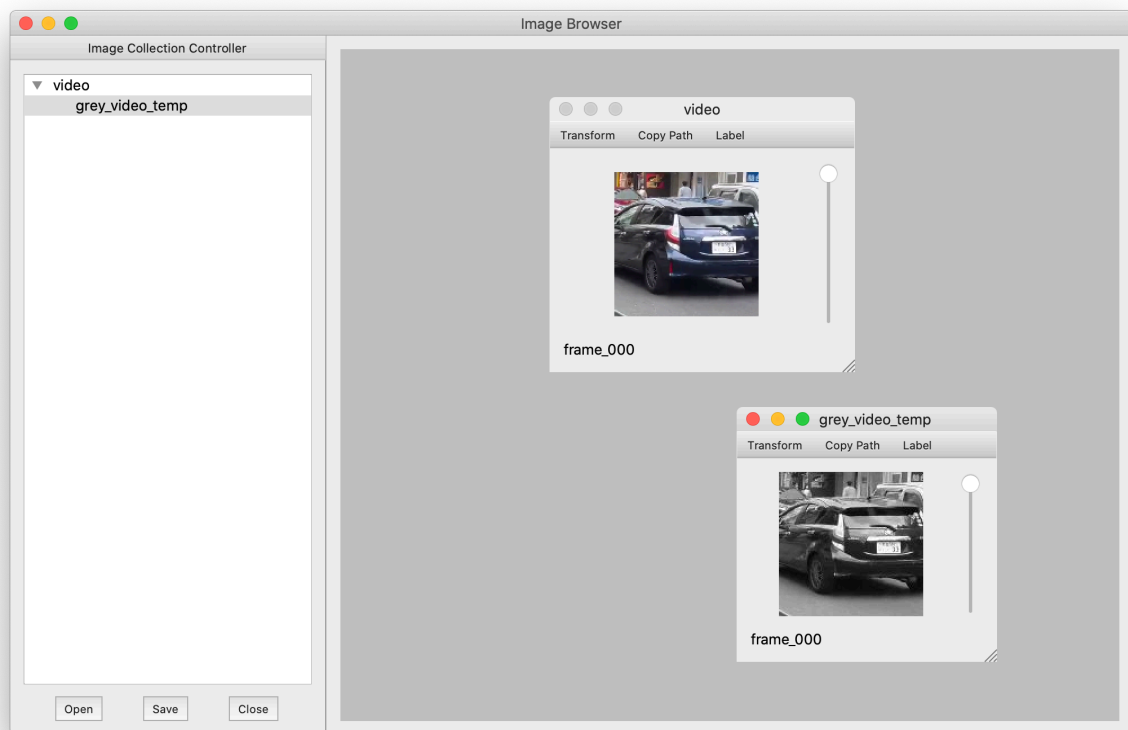
You can drag the slider of the sub to the right of the image to browse forward and backward rapidly. The left bottom corner shows the name of the image. There is a toolbox under the title of the subwindow. "Transform" button will help you process the image series you opened. You can refer to Process images subsection for details. "Copy Path" button will copy the path of the currently displayed image to your clipboard. "Label" button will pop up a menu, which can help you label some images of interest and browse the labeled subset of images in a separate window and save them to a separate path. Please refer to Label Images subsection for more detail.

[Process images](#)

Click "Transform" button on the subwindow, you can see the following dialog:



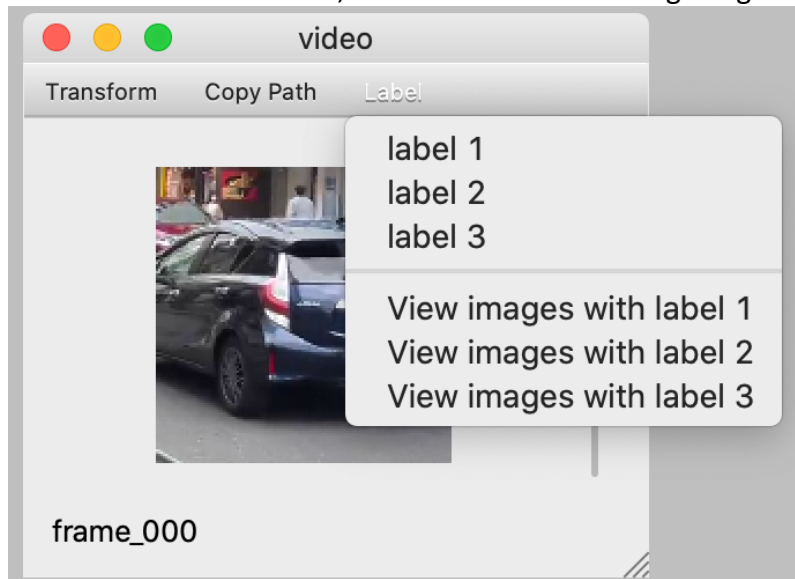
Type in a script for image processing, and the nickname for the new image series which will be generated after processing, and click OK button, VideoLab will run the script to process the image series (if the image series is opened from the server, the processing will run on the server. Otherwise, the processing will run on your local machine) and open the new image series as a sub-series of the original image series:



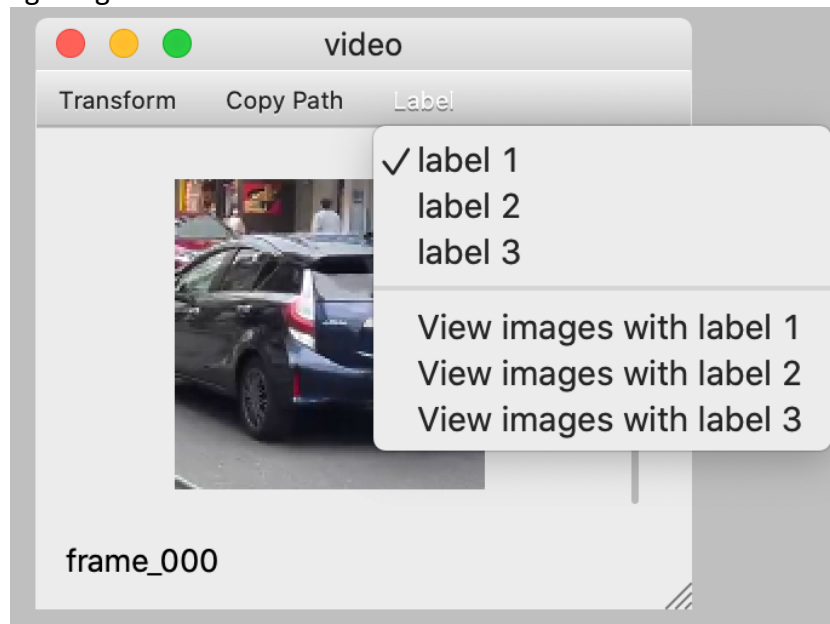
Notice that we added “_temp” to the nickname, meaning this newly generated image series are stored in a temporary folder. You need to save it to avoid losing it after you close VideoLab. Please refer to Save Image Series subsection for more detail. The grammar of the script VideoLab supports will be described in Scripts section.

Label images

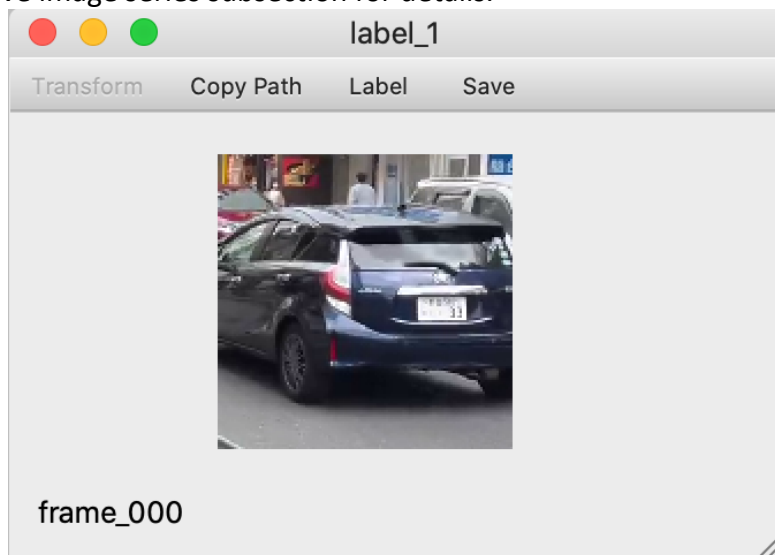
By clicking Label button on the subwindow, a menu like the following image will pop out.



By selecting “label x” in the menu, the currently displayed image will be added to the image subseries of “label x”, and you will see a tick on the item if the current image has been added to the corresponding image subseries:

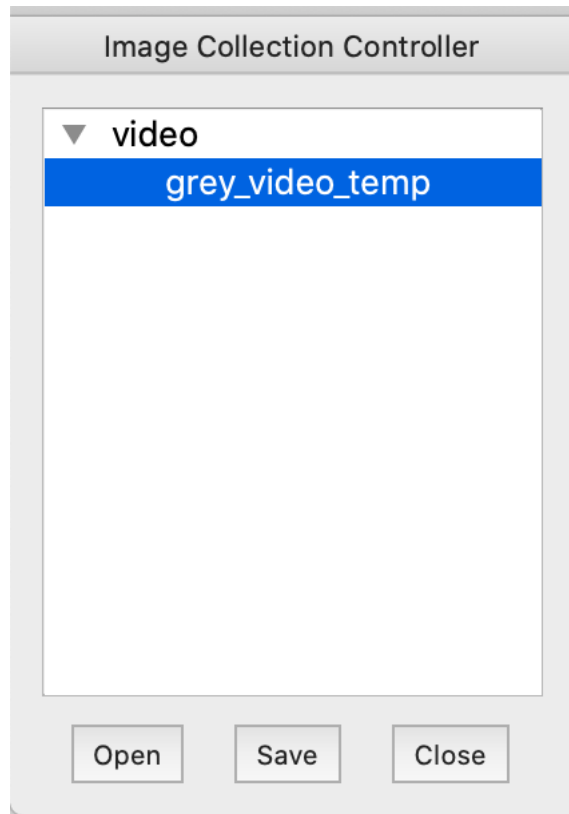


By selecting “View images with label x”, a new subwindow will be opened to show all images with label x. You can save this set of images by clicking “Save” button under the window title. Please refer to Save image series subsection for details.



[Save image series](#)

You can save any image series you opened to another path in another format. To do this, you need to select an image series by clicking it in the dock widget, and then click Save button on the bottom of the dock widget.



A dialog will pop up. You should select the format you want to save the selected image series, input the name for it and input the path you want to save it.

Notice that, if the selected type is folder, the input name is the folder name the image series will be saved, and the input path is the path where the newly created folder will be placed. If the selected type is a file like video or ppm, the input name must include the extension, and the input path is the path where the newly created file will be placed.

You can type in a path located in a server to save the image series to the server directly. In this case, you need to type in a path like "sam@gypsum.cs.umass.edu:~/images.mp4", which is the same as the scp command's format. VideoLab may ask you for the password for the server.

By the way, it is a little different to save an image sub series with labels since those subwindows will not be registered to the dock widget. Please refer to Label images subsection.

[Close image series](#)

Selecting an image series by clicking its nickname in the dock widget and clicking Close button in the bottom of the widget, you can close the image series you have opened. Notice that, the image series and its associated sub series will all be closed together.

Scripts

To process images in VideoLab, you can just type in a script with some simple commands without the need of writing an individual program. We provide many off-the-shelf commands which should suffice the common use. You can also design your own command easily to extend this powerful tool to suit your need. We will introduce how you can achieve this in section Advanced Topics.

A script is formed by several lines of commands. VideoLab will parse the script, divide it into commands, and run it as a pipeline. The basic grammar of each command is like the following format:

command args

For example, in “cvtColor grey”, cvtColor is the command part, and grey is the args part. These two parts are separated by a space.

The args part is a string of the format which can be processed by the built-in argparse module in python. VideoLab uses argparse to parse the arguments input by users. Please refer to <https://docs.python.org/3/library/argparse.html> for the use of argparse. Generally speaking, args can be something like:

--color grey --shape rectangle --workers 2

This is pretty like what we daily use in Linux, so the learning cost is minimized.

Of course, different commands need different arguments. The following table lists all off-the-shelf commands VideoLab currently supports and their associate information. You can also define your own command as I mentioned.

| Command | Arguments | Function | Example |
|------------------------|--|---------------------------------------|-----------------|
| pwcnet | | Compute optical flow with PWC-NET | pwcnet |
| | https://github.com/sniklaus/pytorch-pwc pytorch, cupy(pip install cupy-cuda111), flow_viz(pip install flow_viz) | | |
| cvtColor | grey | Convert the image series to greyscale | cvtColor grey |
| | | | |
| raft | see raft repo | Compute optical flow with RAFT | pwraft |
| | https://github.com/princeton-vl/RAFT pytorch, scipy | | |
| stabilizedTrack | | OpenCV stabilized tracker | stabilizedTrack |
| | | | |

Advanced Topics

Create your own command

You can design any command you need easily for VideoLab. Basically, you just need to write a python class derived from `utils/transformers/Transform_base`, implement the two abstract methods declared in `Transform_base`, and then register the class to `utils/transformers/registeredTransformers.json`.

Let's use `Transform_cvtColor` as an example. This class is defined in `utils/transformers/Transform_cvtColor.py`. As you see, it has a class variable named `command`. You need also define this variable and assign it with a string of the command you want to use. Three other methods are also defined. `__init__` should not have any argument except `self`. The main work for this method is to call the parent's `__init__` method and initialize your own code. The other two methods, `getArgParser` and `processImageCollection`, are the abstract methods you need to implement.

- `getArgParser` takes no argument except `self`, but output a parser of the type from `argparse` module. In this method, we just use `argparse` to customize our own parser to parse the args part of the command. If you don't need any argument for your command, just return `None`.
- `processImageCollection` takes three arguments, i.e. `self`, `model`, `args`. `model` is of a special type defined in VideoLab. You don't need to know the details of the type. The only thing you need to know is that it has two methods: `model.length()` and `model.get(idx)`. The previous will give you an integer indicating the number images in the image series to be processed. The latter will return a tuple of the image in **uint8** numpy array and its associated name. The image is of shape **(Height, Width, Channel)**, where channel is of the sequence **RGB**.

`Args` is the result parsed by the parser you returned from `getArgParser` with the args part of the command as the input. Thus you can use something like `args.target` to get the user's input. Don't forget to examine if the input is valid and raise an exception if needed.

The final thing you need to pay attention to is that we return images and its associated name as a tuple by **yield**, instead of return. The image should be of the same format as the input. You can change its name if you'd like.

After you implemented your own class, register this class to `registeredTransformers.json`. The format of the item to add is like "command": [module path, class name].

Use your own server

VideoLab is designed to be flexible enough to communicate with your server freely. To use your own server, you need to create your own server configuration file, and change `configs/serverConfigs/registeredServerConfig.json` to point to the configuration file you wrote. To create your own server configuration file, you can use `gypsum.json` as a template. You need to change the fields `server`, `username`, `processor_path` to your corresponding server as described in section Installation. The only field we haven't explored is `template_path`. You need to create your own template file and change this field to point to the file. The template file writes all commands you need to run if you want to run a program.

Let's use `gypsum_template.txt` as an example. On Gypsum we need to apply for a new node for computation instead of running programs on the main node, we input `srun` to apply for a new node. Then, we activate our deployed conda environment. The third and the fourth lines are special symbols which will be replaced automatically by VideoLab. `[GENSCRIPT]` will be replaced by a command which generates the script VideoLab will run for image processing. `[RUN]` will be replaced by a command which run the generated script. Finally, we need to type in `exit` to release the computing node we are using.

You need to customize your own template file to adapt to your own server. The template file is just like the way you operate your server daily. You only need to add `[GENSCRIPT]` and `[RUN]` to the template file.