Cross-environment function calls

Jacques Bourg

December 14, 2024

Abstract

In this document, I explain how to call functions from different conda environments in a project. This is useful, since one often wants to use in a same project different deep learning algorithms, as different parts of a pipeline or to benchmark different algorithms. The issue, is that each deep learning algorithm comes with its own environment, and that if one wants to try to install more than two or three of them in a same conda environment, dependency conflicts occur. Therefore, I present a way to call functions from different environments, in a scalable way.

Keywords Python, conda, bash, deep learning, project development.

1 Illustrative example of cross environment function call

In this example, environment $pipeline\_env$ which will be the environment in which we will run a pipeline that uses functions from other environments and $spotiflow\_env$ will be the environment from which we want to import a function. Spotiflow is an existing framework for detection of rna's in stacks of recoded images at different z's (these details are not relevant for our discussion). What matters is that we call a function $spot\_spotiflow$. Inside this function we will call a script in bash called $env\_activate\_spotiflow.sh$, and this bash script calls another function det_spots_sf that runs in the spotiflow environment.

1.1 Function $spot\_spotiflow$ in the $pipeline\_env$

```
def spot_spotiflow(self, file_data: str, target_dir: str):
    """
    Launches a script in bash (env_activate_spotiflow) which itself calls a function det_spots_sf
        that can run in the spotiflow environment.

    The function det_spots saves temporally a .npy file, the function spot_ufish reads it, outputs
        the numpy array with the coordinates of the detected spots (z,y,x) and finally erases the
        temp file.
    """
    current_folder = Path.cwd().parent.parent
    bash_script_path = str(current_folder / Path('src/detection_fish/env_activate_spotiflow.sh'))
    result = subprocess.run(['bash', bash_script_path, file_data, target_dir],
        capture_output=True, text=True)
    print(result)

    if os.path.isfile(target_dir):
        spots_sf = np.load(target_dir)
        Path(target_dir).unlink() # delete the temp file
        return spots_sf
    else:
        raise ImportError('The detection file from spotiflow was not generated !')
```

1.2 Function *activate_spotiflow.sh* in the *pipeline_env*

```
#!/bin/bash

file_data=$1
target_dir=$2

my_bash_function() {
    # Activate spotiflow environment
    # Run in it function spotiflow_detection
    # take target_dir and build
        script_path2='/home/user/Documents/FISH/Data_analysis/pipeline_smfish/ \
    #    src/detection_fish/ufish_detection.py'

    # Extract the base directory of the input string
    base_dir="$(dirname "$(dirname "$(dirname "$(dirname "${target_dir}")")")")"
    # Construct the desired output string
    script_path2="${base_dir}/src/detection_fish/ufish_detection.py"

    eval "$(conda shell.bash hook)"
    conda activate spotiflow_env
    export MPLBACKEND=agg
    result=$(python "$script_path2" "$file_data" "$target_dir")
    echo "Python result: $result"
    conda deactivate
    }
# Call the function
my_bash_function
```

1.3 Function *spotiflow_detection.py* in the spotiflow environment

```
import sys
from pathlib import Path
import numpy as np
from skimage import io
from spotiflow.api import Spotiflow

def det_spots(file_data: str, target_dir:str):
    """
    Detect spots using Spotiflow in spotiflow environment.

    Inputs:
        file_data: string, address of the .tiff file containing the fish.
        target_dir: the absolute path of where to store the temp .npy file
            (Analysis/temp/detected_ufish.npy)
    Output:
        spots: n by 3 np.array, where n is the number of spots.
    For each triplet: first dimension is the number of z stack
                              second and third dimension is the position in pixels.
    Saves as a tuple of size 3 :(z, y ,x) each spot in a .npy file.
    """
    # print("Inside the Spotiflow env")
    spotiflow = Spotiflow()
    spotiflow.load_weights()
    rna = io.imread(file_data)
    pred_spots, _ = spotiflow.predict(rna)
    np.save(target_dir, pred_spots.to_numpy())

if __name__ == "__main__":
    file_data  = sys.argv[1]
    target_file = sys.argv[2]
    det_spots(file_data, target_file)
```

2 Comments

The most important line of code from the first function $spot\_spotiflow$ is:

```
result = subprocess.run(['bash', bash_script_path, file_data, target_dir], capture_output=True,
    text=True)
```

With the instruction subprocess.run, we are able to run a bash command in python. In here we are referencing the path of the bash script $bash\_script\_path$ together with two arguments which are strings ($file\_data$ and $target\_dir$). File data is the path of the data file, and target dir, the name and the path of the output file. In this case, as we wanted to build a function $spot\_spotiflow$ which looks like a traditional python function, I made the choice to then go read this file, erase it and return the output as an numpy array.

On the second function $activate\_spotiflow$ there are many important details depending on your level of bash):

a) Catch the variables passed by the function $spot\_spotiflow$ and define them as global variables:

```
file_data=$1
target_dir=$2
```

b) Before typing the classical $conda\ activate$, use the command

```
eval "conda shell.bash hook"
```

If you proceed to do "conda activate ...", your script will not work. Conda shell.bash hook is a script provided by Conda that contains the necessary code to configure the current Bash shell for using Conda environments.

c) Activate your environment, run your command in python and then deactivate the environment.

d) Finally, call the function $my\_bash\_function$ that you just defined.

I just went trough the important steps. In this case in particular, before calling my python function in the spotiflow environment, I had to add a line $export\ MPLBACKEND = agg$, since the compilator was complaining. This is a line that ensures that your matplotlib scripts work correctly in environments without a graphical display. Therefore when you write your scripts, do not include this kind of line at first and let the compilator tell you to set the environment variable if needed.

On the third function $spotiflow\_detection.py$, there are fewer things to note. One is just running a python function in a given environment, so do the necessary imports, in here do the inference on the data, and save the results. As in the bash script, one has to define the function, and then call properly the function, with the right arguments. For use the $sys.argv$ command, and then call the function.

```
arg1 = sys.arg[1]
```

This approach is interesting and scalable as I said, but it has its drawbacks. For instance if the function we are calling in the other environment has several parameters, it is less easy to implement such approach.

Finally, as an advice in order to start playing with this cross-environment functions calls, use prints in bash or in the second environment to debug.

```
print("Inside the Spotiflow env")
```