PyCVF
Scientific and Multimedia Python Workshop (II-i)
by Bertrand Nouvel <nouvel@nii.ac.jp>

Japanese French Laboratory for Informatics  July, 2010
You may find these slides and the code snippets at:

URL: http://jfli.nii.ac.jp/wikijfli/PycvfTutorial201007/
What means doing scientific experiments for solving a problem?
1. Consider a problem
2. Suggest an improvement with respect to the state of the art
3. Consider an evaluation protocol
4. Gather the required elements a database
5. Implement the model and the evaluation procedure
6. Tune the model
7. Analyze the result
At some of these steps, software may help you

1. Consider a problem
2. Find something better
3. Consider an evaluation protocol
4. Gather the required elements a database
5. Implement the model and the evaluation
6. Tune the model
7. Analyze the result
PyCVF = help you to do the program.
PyCVF goals

- PyCVF is dedicated to researchers/engineers...
- PyCVF is first a framework
- Goal of PyCVF: bring more interoperability
- PyCVF framework principle: = DRY = Don’t repeat yourself
- The "PyCVF-extensions" are the libraries.
How do we install PyCVF?
Short answer:

- Use **Ubuntu Linux**
- Use the install script
- Upgrade to the latest version
Long answer:

- PyCVF theoretically work on any platforms
- PyCVF 0.1: (latest official release with install)
  PyCVF 0.1: many dependencies -> difficult to install on non-linux platform.
- PyCVF 0.2: (upcoming release) is based on "Extension" packages
  PyCVF reduced to minimal set of python file, that will be easy to install. Additional: in - charge of some software dependencies / new features.

**Have to build each package you are using!**
Building the extensions

For each extension that you want to build:

- run the setup. py script when there is one
- go into the wrappers directory: have a look at the dev-install. sh, eventually run it.
How do we maintain PyCVF up to date?
You may pull PyCVF from the two following branched:

- lp: bertrand-nouvel/+junk/pycvf-tranx  development version
- lp: bertrand-nouvel/pycvf/main    release

```
bzr pull  # for default repository
bzr pull lp:~bertrand-nouvel/+junk/pycvf-tranx
bzr pull lp:~bertrand-nouvel/+junk/pycvf-tranx    --remember
```
How do I get help about PyCVF?
By the **docstring**:

- By reviewing the **documentation online:**
  Contributions are also welcome.

- By asking questions on the **discussion groups**:

- By reading the files in the per-extension **documentation folder**:

- By having a look at the per-extension **demos**.
What is the core of PyCVF?
The essential folders

- build
- data
- debian
- demos
- devel
- dist
- documentation
- maintainer
- pycvf
- pycvfext
- LICENSE
- LICENSE.LGPL-3
- MANIFEST.in
- AUTHORS
- INSTALL
- install_additional_packages.py
- setup.py
- setup.pyc

Directory where the included extension modules are

Where the core object is
The other folders

Directory where the included extension modules are

Directory where the included extension modules are

Where the core object is
The PyCVF folder

PyCVF – structural elements/loaders... Utilities / Important libraria
The PyCVF subfolders

- apps
- core
- databases
- dotypes
- lib
- management
- nodes
- schemes
- stats
- structures
- ui
- __init__.py

Ways of making iteration. Explosion of the data.

Manage data representation on devices

Nodes for computation

Applications

Databases
The PyCVF core content

- **Base classes**
- **Core utilities**
- **Loader**
- **Deprecation/Unused**
The lib directory: small utilities
Many extensions
Content of a Typical package

- applications
- databases
- datas
- datatypes
- demos
- documentation
- experiences
- lib
- management
- nodes
- structures
- wrappers
- _init_.py
- _init_.pyc
- _init_.pyo
- AUTHORS
- LICENSE
- LGPL-3
- manifest.py
- manifest.pyc
**generic**: Anything displayed as the python interpreter will have had done it

**couple/tuple**: A couple/tuple of elements of specified types

**numericvector**: A one dimensional numeric vector

**numericarray**: A numpy array

**image**: a 2/3 dimensional numpy array / (so far, number of layers, datatype is not fixed)

**video**: A reader returning array images

**audio**: A reader returning audio buffer to be played

**histogram**: Alternative representation of a NumericVector
We now review some concepts that will help you to be more familiar with the framework.
In the interpreter you may create any datatype:

**Code (1)**

```python
from pycvf.datatypes import list as ldtt
from pycvf.datatypes import generic
from pycvfext.image.datatypes import import image

ldtt.Datatype(image.Datatype)
ldtt.Datatype(generic.Datatype)
```

In the command line use the DTP keyword to avoid having to make an import;

**Code (2)**

```python
DTP("list",DTP("image") )
```
Readers

The reader class:

Basically:

- A seekable iterator
- With a mode for callback
- Must provide __len__
- Must provide copy()
- May provide a __getitem__ operator
- x. step()

- A reader may be copied (Iterators may not - or with heavy memory requirements)
Readers provided for:

- Video / Audio readers
- Subsequence reader:
  Transform any sequence in a readable object
- TransformedReader:
  Apply a transformation to the elements of a reader.
PyCVF provides you a convenient/uniform expressions for describing dataflows and databases. These expressions are actually python expressions that are evaluated in a specific context.
Normally to used database or a node you shall do:

**Code (3)**

```python
import pycvf.databases.randomized.DB
db=pycvf.databases.randomized.DB()
```

It is a little bit long.
So we first use autoimp. Hence you may do:

```
Code (4)

db=pycvf.databases.randomized.DB()
```

which is already better.
By setting \_
call\_
=DB in the module, the syntax becomes:

```
Code (5)

db=pycvf.databases.randomized()
```
And by defining a correct environment for the imports

Code (6)

```
db=randomized()
```
For extension modules the "databases. " / "nodes. " will be automatically added at the good place in the path:

```python
Code (7)

db=pycvfext.image.databases.caltech256()
```

Can be written

```python
Code (8)

db=image.caltech256()
```
Database: image.caltech256(10,1)

Model: naive()
What are the core applications of PyCVF?
Core applications

- **pycvf_model_run**: May be used to run any dataflow, on any dataset.
Core applications

- **pycvf_view**
  roughly equivalent: "pycvf_model_run -m 'debug.display()"

- **pycvf_compute_features**

- **pycvf_features_view**
Other Important Applications

pycvf_build_index
Suggestions:

- `pycvf_compute_clusters`
- Do you have any?
How do I create my own extension module?
Code (9)

```bash
cd pycvf/pycvfext
mkdir mymodule; cd mymodule
touch __init__.py
cp ../image/manifest.xml; jed manifest.xml
```
How can I change the default settings of PyCVF
Default PyCVF settings

1. You may create a ". pycvf-settings. py" in your home directory.
2. You may also create per-project "pycvf-settings. py" files.
Example of content for pycvf-settings.py

```
YAHOOAPPID="lwKQaNjV34E0dQD_rX3vFW_VK71VOhSlhcZB073I4TCUGlksI3fNNGbwjDwh6"
CALTECH256PATH="/home/tranx/databases/pics/256_ObjectCategories/
GREEDYRSC_PATH="/home/tranx/pycvf-tranx/trunk/pycvfext/greedyRSC/wrappers/greedyRSC_1.1"
LABELMEROOT="/home/tranx/medias/per900a/raid0/tranx/labelme/
```
Example of content for pycvf-settings.py

**Code (11)**

```python
DISPLAY_DRIVER="qt"  # "caca"
AA_WITH_CURSES=False
WEKACLASSPATH="/home/tranx/lib/weka-3.6.2/weka.jar:/home/tranx/build/libsvm-2.85/java/libsvm.jar"
JAVA_CMD="java"
DEFAULT_DATABASE="from_list([(1,'You'),(2,'shall'),(3,'specify a db')])"
PER900AROOT="/"
```
What relationship PyCVF has with opencv?
OpenCV in PyCVF

- OpenCV is a separated package now, but some image-processing lib have dependencies on it.
- SWIG OpenCV wrappers provideda difficult to use with many functions. A Cython-based low-level wrapper for OpenCV 1.0 was developped in 2008 by myself. So far it was efficient, but no time/interest to port it to C++
- The new opencv 2.0 is promising (although some performance issues have been reported? )
Questions

Thanks

Q/A