pgcv - Computer Vision Objects for PostgreSQL

`pgcv` is a PostgreSQL extension for Computer Vision from the database server. The extension implements algorithms for image segmentation, in particular: digital mammogram segmentation.

The extension implements both data types and functions. The data types are PostgreSQL composite types and the functions were created using PL/Python, meaning the function's body is written in Python.

Requirements

This extension requires the following:

- PostgreSQL (version 10 recommended)
- Python3
- The following Python packages
    - Numpy
    - Scipy
    - scikit-image
    - Pillow
    - Pandas
- `plpython3u` installed in the PostgreSQL database

The mentioned Python packages can be installed by executing the following command on your terminal:

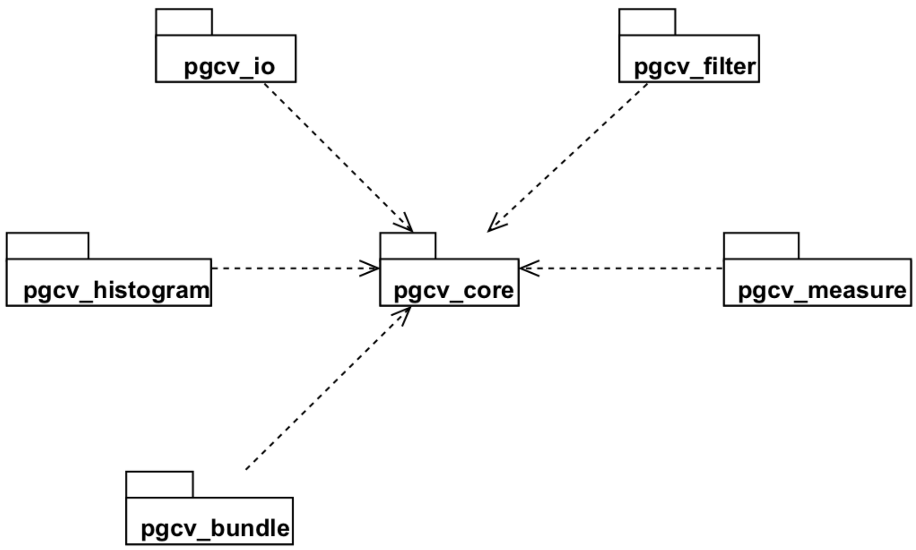
```
pip3 install numpy scipy scikit-image pandas Pillow
```

Structure

The extension was structured into SQL-schemas because it allows the possibility to modularize the functions and group them into logical packages.

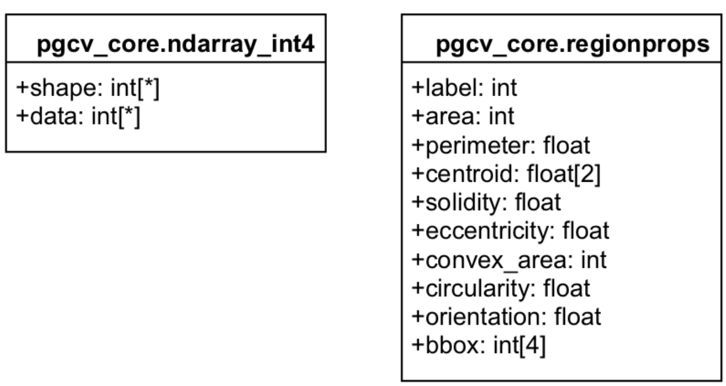
There are 7 schemas in the extension. `pgcv_core` defines the datatypes and the rest of the schemas define the functions that operate over theese datatypes.

The following diagram shows the dependency structure of these schemas:



Data Types

There are two datatypes in the `pgcv_core`: `ndarray_int4` and `pegionprops`, as shown in the figure bellow:



1. `pgcv_core.ndarray_int4`: N-dimensional array of int4 elements. Used to represent and store images. The shape is a tuple of N integers (one for each dimension) that provides information on how far the index can vary along that dimension. The data is a buffer which contains a flattened representation of the multidimensional array's data

2. `pgcv_core.regionprops`: Region properties of an object found in a binary image. The properties contained in this type are label, area, perimeter, centroid, silidity, eccentricity, convex_area, circularity, orientation and bbox (bounding box)

Function Modules

pgcv_io

This schema contains the image input and output functions to the filesystem. Meaning that this functions read and write images into files.

`image_read`

Reads an image from a file into an `ndarray_int4`.

```
-- having a filename of a grayscale image in disk
SELECT shape FROM pgcv_io.image_read('<filename>');
```

`image_write`

Writes an image from an ndarray_int4 into the specified filename (path).

```
-- having an image in the database and the output filename
SELECT pgcv_io.image_write(<image>, '<filename>');
```

---

pgcv_filter

This schema contains the image filtering functions. One example of this functions is the `median_blur` which replaces each pixel by the median of a local window array given by a kernel size.

`blur_median`

Perform a median filter on an N-dimensional array.

```
-- having an image in the database and
-- an odd kernel size (kernel size defaults to 5 if not specified)
SELECT pgcv_filter.median_blur(<image>, [<kernel size>]);
```

`threshold_otsu`

Calculates a threshold value based on Otsu's method.

```
```

```
-- having an image in the database
SELECT pgcv_filter.threshold_otsu(<image>);
```

`enhancement_otsu`

Enhances an image using the Otsu's threshold. Used for mammogram analysis.

This function uses a method designed by Johnny Villalobos that has proven to be quite effective for mammogram segmentation. It is described follows:

Let *t* be the *threshold* of an image calculated through the Otsu's method, *max* the maximum grayscale value of the image and *f* the enhancement factor so that

$$f = \frac{t}{255 - t}$$

the value of each enhanced pixel *p*' corresponds to

$$p' = (1 - f)(max - p(1 + f))$$

```
-- having an image in the database
SELECT pgcv_filter.enhancement_otsu(<image>);
```

`binarize`

Binarizes an image according to the supplied threshold.

```
-- having an image in the database and a threshold value
SELECT pgcv_filter.binarize(<image>, <threshold>);
```

---

pgcv_histogram

This schema contains the histogram computing functions. There are two main kinds of histograms in pgcv, both return an histogram and a set of bin features (either the center of the bins or the edges)

`hist_bin_edges`

Compute the histogram of a set of data and the bin edges

```
-- having an image in the database,
-- the number of bins (bins defaults to 10 if not specified)
-- and whether the histogram has to be normalized or not
SELECT * FROM pgcv_histogram.hist_bin_edges(<image>, [<bins>, [<as_float>]]);
```

`hist_bin_centers`

Compute the normalized histogram of a set of data and the bin centers

```
-- having an image in the database
-- and the number of bins (bins defaults to 10 if not specified)
SELECT * FROM pgcv_histogram.hist_bin_centers(<image>, [<bins>]);
```

---

pgcv_measure

This schema contains the functions that perform measure computations on the image. In particular, `pgcv_measure` includes de region properties functions, which find objects on a binarize image

`region_props_json`

Returns a json array with the region properties of a binary image

```
-- having a binarized image in the database
SELECT pgcv_measure.region_props_json(<image>);
```

`region_props`

Returns a set of region properties found in a binary image

```
-- having a binarized image in the database
-- this allows for the inclusion of WHERE conditions
-- for filter the properties
SELECT * FROM pgcv_measure.region_props(<image>);
```

---

pgcv_bundle

The bundle schema provides access to common successive operations performed to an image. The purpose of this schema is to reduce the overhead produced by the comunication from the PostgreSQL server and Python.

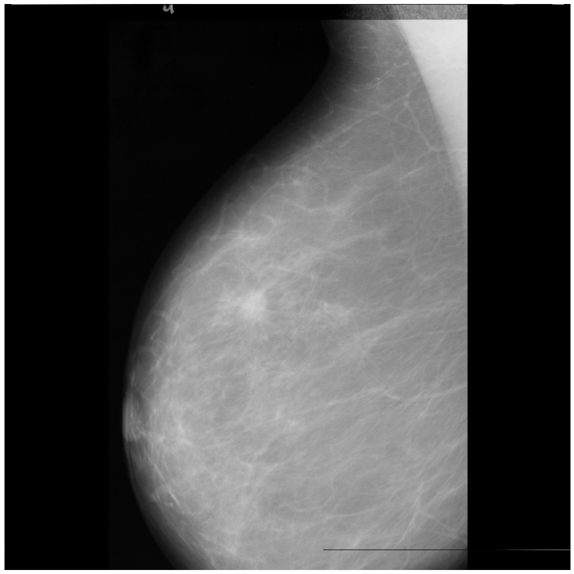
`mam_region_props`

Returns a set of region properties found in a mammogram image

```
-- having an image in the database
-- and odd kernel size (kernel size defaults to 5 if not specified)
SELECT * FROM pgcv_bundle.mam_region_props(<image>, [<kernel size>]);
```

Example

The following example shows the sequence of SQL commands needed to perform a mammogram segmentation using the `pgcv`. The image used for this example is an mammogram taken from the MIAS MiniMammographic Database:



This example is divided into two steps: the image segmentation using the Otsu enhancement and the object extraction using the region properties

Image Segmentation

The following steps show the needed steps to perform the segmentation

- Read the image from the file system
- Perform a median_blur
- Compute the Otsu threshold of the image
- Enhance the image
- Binarize the image using the threshold

```
DO $_$
DECLARE
  image pgcv_core.ndarray_int4;
  result pgcv_core.ndarray_int4;
  thresh float;
BEGIN
  image :=  (SELECT pgcv_filter.blur_median(
                      pgcv_io.image_read('/path/to/original/image.png'), 5
                    ));
  thresh := (SELECT pgcv_filter.threshold_otsu(image));
  result := (SELECT pgcv_filter.binarize(pgcv_filter.enhancement_otsu(image),
thresh));
  PERFORM pgcv_io.image_write(result, '/path/to/binarized/image.png');
END
$_$
```

The result of this process is the following image:

Object Extraction

The object extraction consists of a single SQL query that computes the region properties and allows to filter them through a WHERE clause

```
SELECT
  label, area, perimeter, centroid, circularity
FROM pgcv_measure.region_props(pgcv_io.image_read('/Users/ro/Desktop/prueba.png'))
WHERE area > 15 AND area < 55;
-- you could also include solidity, eccentricity, convex_area, orientation and bbox
in the query
```

The result of this query is the following table of region properties

label	area	perimeter	centroid	circularity
4	20	15.071067811865474	{299.75000000000000000,832.80000000000000000}	1.1065010026804611
5	19	14.242640687119286	{420.26315789473680000,427.31578947368420000}	1.1770161688565013
10	36	20.727922061357855	{473.47222222222223000,481.33333333333330000}	1.0529332270693823
13	16	13.242640687119284	{490.68750000000000000,461.25000000000000000}	1.1465174146811834
19	17	13.071067811865474	{544.00000000000000000,580.88235294117650000}	1.250364543402942
24	48	26.485281374238568	{577.29166666666660000,400.58333333333330000}	0.8598880610108875
25	27	17.65685424949238	{578.70370370370370000,379.37037037037040000}	1.0882958272169045
29	30	22.727922061357855	{637.46666666666670000,391.53333333333336000}	0.7298131021442217
35	30	20.14213562373095	{661.76666666666670000,467.10000000000000000}	0.929223291202501
36	34	26.106601717798213	{670.52941176470590000,451.55882352941177000}	0.6268853111775277

label	area	perimeter	centroid	circularity
39	24	19.692388155425117	{676.58333333333340000,464.00000000000000000}	0.7777219038741342
41	26	19.44974746830583	{687.46153846153850000,449.30769230769230000}	0.8636848033284439
44	17	13.071067811865476	{687.00000000000000000,492.52941176470586000}	1.2503645434029418
46	17	13.071067811865476	{722.23529411764710000,590.00000000000000000}	1.2503645434029418
55	23	15.65685424949238	{734.34782608695650000,462.91304347826090000}	1.1790403893488048
62	20	15.071067811865474	{750.40000000000000000,426.75000000000000000}	1.1065010026804611
64	32	20.520815280171306	{758.00000000000000000,461.71875000000000000}	0.9549279835285656
65	50	32.935028842544405	{768.68000000000000000,506.98000000000000000}	0.5792469719204167
67	25	17.071067811865476	{768.28000000000000000,537.20000000000000000}	1.0780241689052945
69	18	13.071067811865476	{773.83333333333340000,366.72222222222223000}	1.3239153988972323
70	51	25.556349186104047	{785.56862745098040000,284.41176470588240000}	0.98125619872571
98	26	18.485281374238568	{868.50000000000000000,457.76923076923080000}	0.9561611214257792
99	23	16.485281374238568	{872.34782608695650000,422.21739130434780000}	1.0635183109500363
108	30	25.727922061357855	{942.16666666666660000,428.43333333333334000}	0.5695366755033309
112	35	23.556349186104047	{975.22857142857140000,635.28571428571430000}	0.7926143695103078