BPDL Documentation

Release 0.2.3

Jiri Borovec

Oct 28, 2020

CONTENTS

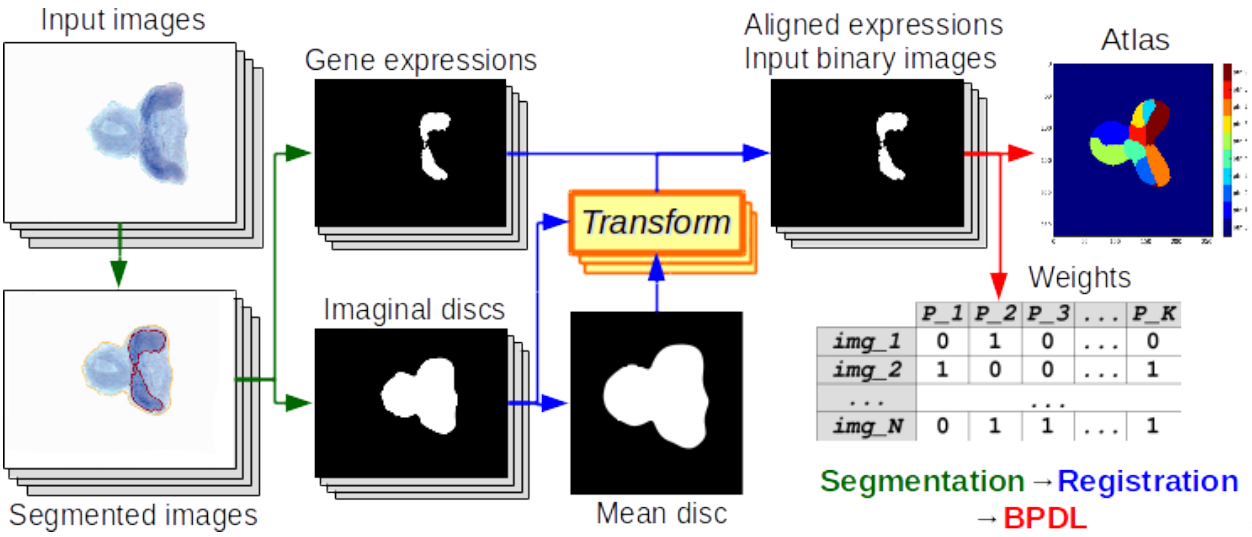
CHAPTER

ONE

CONTENTS

1.1 Binary Pattern Dictionary Learning

We present a final step of image processing pipeline which accepts a large number of images, containing spatial expression information for thousands of genes in Drosophila imaginal discs. We assume that the gene activations are binary and can be expressed as a union of a small set of non-overlapping spatial patterns, yielding a compact representation of the spatial activation of each gene. This lends itself well to further automatic analysis, with the hope of discovering new biological relationships. Traditionally, the images were labelled manually, which was very time-consuming. The key part of our work is a binary pattern dictionary learning algorithm, that takes a set of binary images and determines a set of patterns, which can be used to represent the input images with a small error.



schema

For the image segmentation and individual object detection, we used Image segmentation toolbox.

1.1.1 Comparable (SoA) methods

We have our method BPDL and also we compare it to State-of-the-Art, see Faces dataset decompositions:

- **Fast ICA**, derived from sklearn.decomposition.FastICA
- **Sparse PCA**, derived from sklearn.decomposition.SparsePCA
- **Non-negative Matrix Factorisation**, derived from sklearn.decomposition.NMF
- **Dictionary Learning** with Matching pursuit, derived from sklearn.decomposition.DictionaryLearning
- **Spectral Clustering** used in SPEX2, derived from sklearn.cluster.SpectralClustering

- **CanIca & MSDL** used for observing spatial activation in fMRI, derived from nilearn.decomposition.CanICA and nilearn.decomposition.DictLearning
- our **Binary Pattern Dictionary Learning**

1.1.2 Installation and configuration

Configure local environment

Create your local environment, for more see the User Guide, and install dependencies requirements.txt contains a list of packages and can be installed as

```
@duda:~$ cd pyBPDL
@duda:~/pyBPDL$ virtualenv env
@duda:~/pyBPDL$ source env/bin/activate
(env)@duda:~/pyBPDL$ pip install -r requirements.txt
(env)@duda:~/pyBPDL$ python ...
```

moreover, in the end, terminating. . .

```
(env)@duda:~/pyBPDL$ deactivate
```

Installation

The package can be installed via pip

```
pip install git+https://github.com/Borda/pyBPDL.git
```

alternatively, using `setuptools` from a local folder

```
python setup.py install
```

1.1.3 Data

We work on synthetic and also real images.

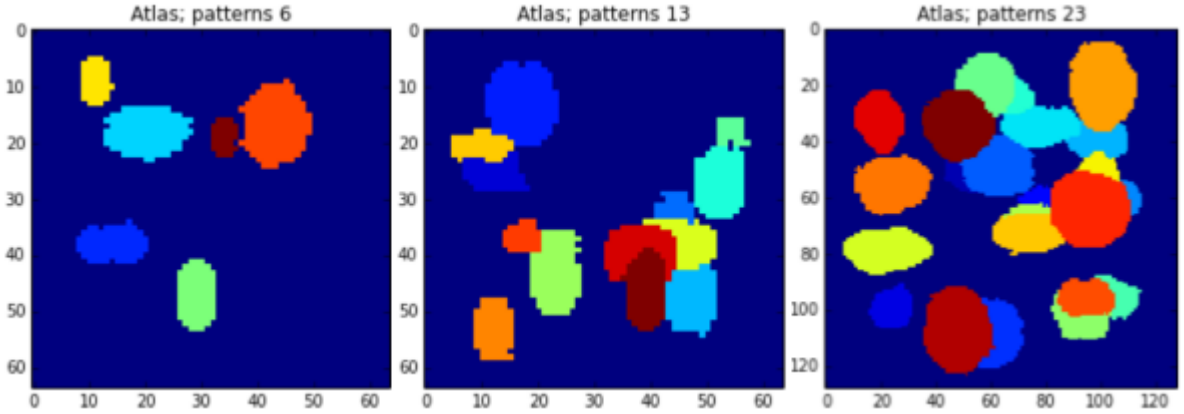
Synthetic datasets

We have script `run_dataset_generate.py` which generate a dataset with the given configuration. The images subsets are:

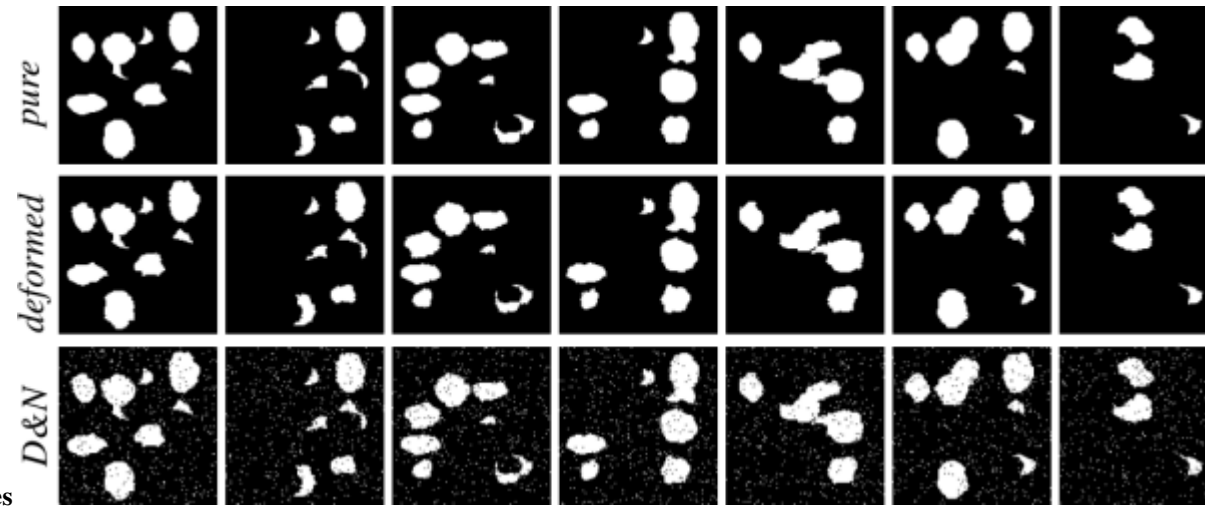
1. **pure** images meaning they are generated just from the atlas
2. **noise** images from (1) with added binary noise
3. **deform** images from (1) with applied small elastic deformation
4. **deform&noise** images from (3) with added binary noise

both for binary and fuzzy images.Some parameters like number of patterns and image size (2D or 3D) are parameters passed to the script Other parameters like noise and deformation ratio, are specified in the script.

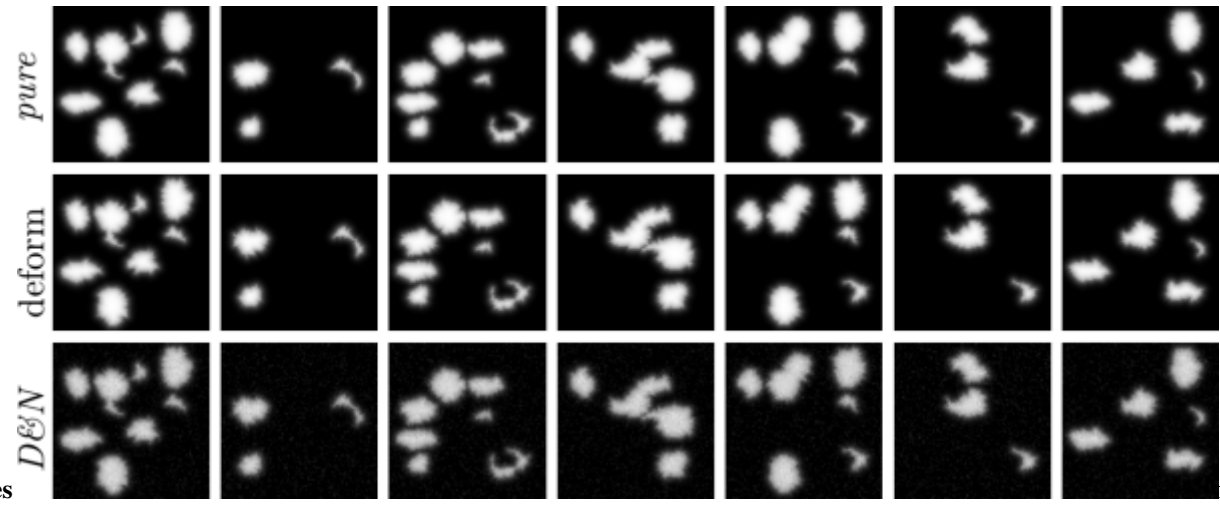
```
python experiments/run_dataset_generate.py \
    -p ~/DATA/apdDataset_vX \
    --nb_samples 600 --nb_patterns 9 --image_size 128 128
```



Sample atlases atlases



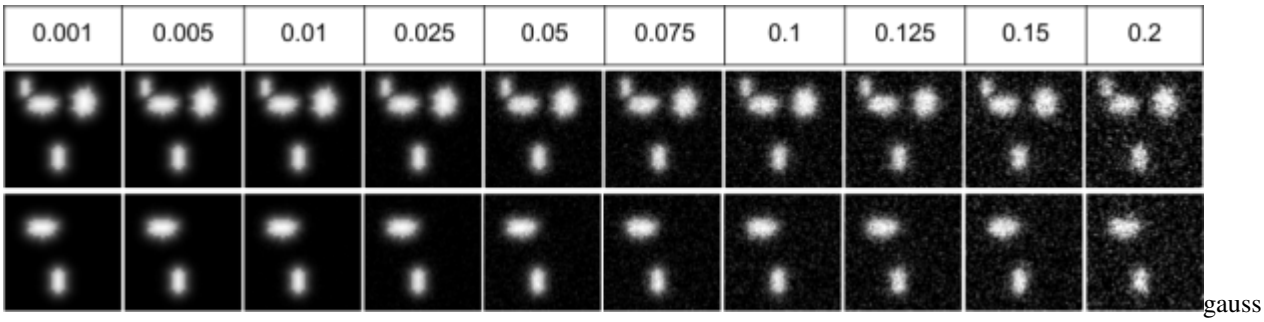
Sample binary images samples



Sample fuzzy images samples

For adding Gaussian noise with given sigmas use following script:

```
python experiments/run_dataset_add_noise.py \
    -p ~/Medical-drosophila/synthetic_data \
    -d apdDataset_vX --sigma 0.01 0.1 0.2
```



gauss noise

Real images

We can use as input images, either binary segmentation or fuzzy values. For the activation extraction we used pyImSegm package.

Drosophila imaginal discs

For extracting gene activations, we used unsupervised segmentation because the colour is appearing variate among images, so we segment the gene in each image independently.

To cut the set of images to the minimal size with reasonable information (basically removing background starting from image boundaries) you can use the following script

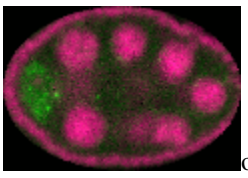
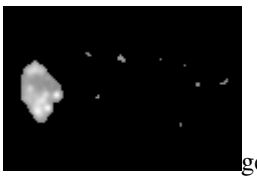
```
python experiments/run_cut_minimal_images.py \
    -i "./data_images/imaginal_discs/gene/*.png" \
    -o ./data_images/imaginal_discs/gene_cut -t 0.001
```

Drosophila ovary

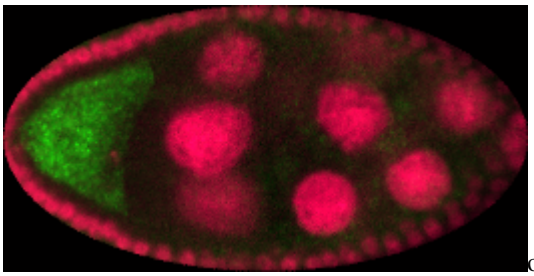
Here the gene activation is presented in the separate channel - green. So we just take this information and normalise it. Further, we assume that this activation is fuzzy based on intensities on the green channel.

```
python experiments/run_extract_fuzzy_activation.py \
    -i "./data_images/ovary_stage-2/image/*.png" \
    -o ./data_images/ovary_stage-2/gene
```

Ovary in development stage 2


ovary stage 2 
gene activation s2

Ovary in development stage 3

 ovary stage 3  gene activation s3

1.1.4 Experiments

We run an experiment for debugging and also evaluating performances. To collect the results we use `run_parse_experiments_result.py` which visit all experiments and aggregate the configurations with results together into one large CSV file.

```
python run_parse_experiments_result.py \
    -i ~/Medical-drosophila/TEMPORARY/experiments_APDL_synth \
    --fname_results results.csv --func_stat mean
```

Binary Pattern Dictionary Learning

We run just our method on both synthetic/real images using `run_experiment_apd_bpdl.py` where each configuration have several runs in debug mode (saving more log information and also exporting all partially estimated atlases)

1. **Synthetic datasets**

```
python experiments/run_experiments.py \
    --type synth --method BPDL \
    -i ./data_images/syntheticDataset_vX \
    -o ./results -c ./data_images/sample_config.yml \
    --debug
```

2. **Real images - drosophila**

```
python experiments/run_experiments.py \
    --type real --method BPDL  \
    -i ~/Medical-drosophila/TEMPORARY/type_1_segm_reg_binary \
    -o ~/Medical-drosophila/TEMPORARY/experiments_APDL_real \
    --dataset gene_small
```

Using configuration YAML file `-cfg` we can set several parameters without changing the code and parametrise experiments such way that we can integrate over several configurations. While a parameter is a list it is aromatically iterated, and you set several iterations, then it runs as each to each option, for instance

```
nb_labels: [5, 10]
init_tp: 'random'
connect_diag: true
overlap_major: true
gc_reinit: true
ptn_compact: false
```

(continues on next page)

(continued from previous page)

```
ptn_split: false
gc_regul: 0.000000001
tol: 0.001
max_iter: 25
runs: 1
deform_coef: [null, 0.0, 1.0, 0.5]
```

will run 2 * 4 = 8 experiment - two numbers of patterns and four deformation coefficients.

All methods

We can run all methods in the equal configuration mode on given synthetic/real data using `run_experiments_all.py` running in info mode, just a few printing

1. **Synthetic datasets**

```
python experiments/run_experiments.py \
    -i ~/Medical-drosophila/synthetic_data/atomicPatternDictionary_v1 \
    -o ~/Medical-drosophila/TEMPORARY/experiments_APDL_synth1 \
    --method PCA ICA DL NMF BPDL
```

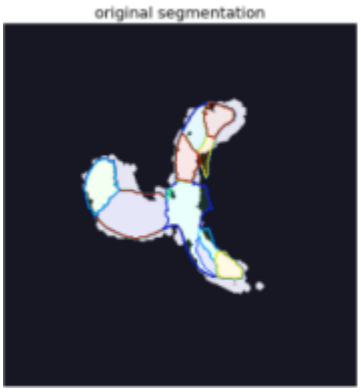
2. **Real images - drosophila**

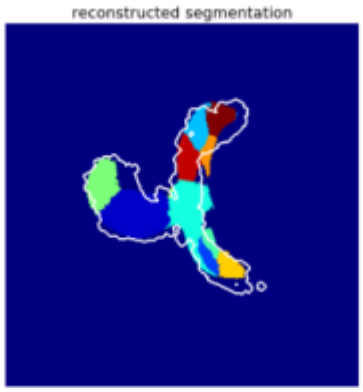
```
python experiments/run_experiments.py --type real \
    -i ~/Medical-drosophila/TEMPORARY/type_1_segm_reg_binary \
    -o ~/Medical-drosophila/TEMPORARY/experiments_APD_real \
    --dataset gene_small
```

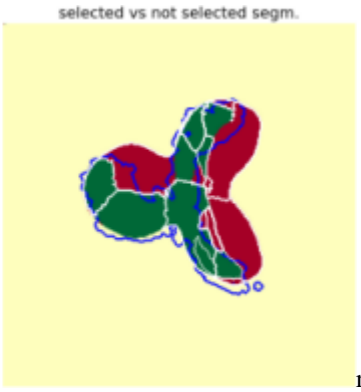
1.1.5 Visualisations

Since we have a result in the form of estimated atlas and encoding (binary weights) for each image, we can simply see the back reconstruction

```
python experiments/run_reconstruction.py \
    -e ./results/ExperimentBPDL_real_imaginal_disc_gene_small \
    --nb_workers 1 --visual
```







reconstruction

Aggregating results

The result from multiple experiments can be simple aggregated into single CVS file

```
python experiments/run_parse_experiments_results.py \
    --path ./results --name_results results.csv  \
    --name_config config.yaml --func_stat none
```

In case you need to add or change an evaluation you do not need to return all experiment since the aliases and encoding is done, you can just rerun the elevation phase generating new results `results_NEW.csv`

```
python experiments/run_recompute_experiments_result.py -i ./results
```

and parsing the new results

```
python experiments/run_parse_experiments_results.py \
    --path ./results --name_results results_NEW.csv  \
    --name_config config.yaml --func_stat none
```

1.1.6 References

For complete references see bibtex.

1. Borovec J., Kybic J. (2016) **Binary Pattern Dictionary Learning for Gene Expression Representation in Drosophila Imaginal Discs.** In: Computer Vision – ACCV 2016 Workshops. Lecture Notes in Computer Science, vol 10117, Springer, DOI: 10.1007/978-3-319-54427-4_40.

1.2 bpdl package

1.2.1 Submodules

bpdl.data_utils module

bpdl.dictionary_learning module

bpdl.metric_similarity module

Introducing some used similarity measures fro atlases and etc.

Copyright (C) 2015-2020 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`bpdl.metric_similarity.`**`compare_atlas_adjusted_rand`**(*a1*, *a2*)

using adjusted rand and transform original values from (-1, 1) to (0, 1) http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

Parameters

- **a1** – np.array<height, width>
- **a2** – np.array<height, width>

Return float with 0 means no difference

```
>>> atlas1 = np.zeros((7, 15), dtype=int)
>>> atlas1[1:4, 5:10] = 1
>>> atlas1[5:7, 6:13] = 2
>>> atlas2 = np.zeros((7, 15), dtype=int)
>>> atlas2[2:5, 7:12] = 1
>>> atlas2[4:7, 7:14] = 2
>>> compare_atlas_adjusted_rand(atlas1, atlas1)
0.0
>>> compare_atlas_adjusted_rand(atlas1, atlas2)
0.656...
```

bpdl.metric_similarity.**compare_atlas_rnd_pairs**(*a1*, *a2*, *rand_seed=None*)

compare two atlases as taking random pixels pairs from both and evaluate that the are labeled equally of differently

Parameters

- **a1** – np.array<height, width>
- **a2** – np.array<height, width>
- **rand_seed** – random initialization

Return float with 0 means no difference

```
>>> atlas1 = np.zeros((7, 15), dtype=int)
>>> atlas1[1:4, 5:10] = 1
>>> atlas1[5:7, 6:13] = 2
>>> atlas2 = np.zeros((7, 15), dtype=int)
>>> atlas2[2:5, 7:12] = 1
>>> atlas2[4:7, 7:14] = 2
>>> compare_atlas_rnd_pairs(atlas1, atlas1)
0.0
>>> round(compare_atlas_rnd_pairs(atlas1, atlas2, rand_seed=0), 5)
0.37143
```

bpdl.metric_similarity.**compare_matrices**(*m1*, *m2*)

sum all element differences and divide it by number of elements

Parameters

- **m1** – np.array<height, width>
- **m2** – np.array<height, width>

Return float

```
>>> seg1 = np.zeros((7, 15), dtype=int)
>>> seg1[1:4, 5:10] = 3
>>> seg1[5:7, 6:13] = 2
>>> seg2 = np.zeros((7, 15), dtype=int)
>>> seg2[2:5, 7:12] = 1
>>> seg2[4:7, 7:14] = 3
>>> compare_matrices(seg1, seg1)
0.0
>>> compare_matrices(seg1, seg2)
0.819...
```

bpdl.metric_similarity.**compare_weights**(*c1*, *c2*)

Parameters

- **c1** – np.array<height, width>
- **c2** – np.array<height, width>

Return float

```
>>> np.random.seed(0)
>>> compare_weights(np.random.randint(0, 2, (10, 5)),
...                 np.random.randint(0, 2, (10, 5)))
0.44
```

bpdl.metric_similarity.**compute_classif_metrics**(*y_true*, *y_pred*, *metric_averages=('macro', 'weighted')*)

compute standard metrics for multi-class classification

Parameters

- **metric_averages** (*list(str)*) –
- **y_true** (*list(int)*) –
- **y_pred** (*list(int)*) –

Return {**str** float}:

```
>>> y_true = np.array([0] * 3 + [1] * 5)
>>> y_pred = np.array([0] * 5 + [1] * 3)
>>> dist_sm = compute_classif_metrics(y_true, y_pred)
>>> pair_sm = [(n, dist_sm[n]) for n in sorted(dist_sm.keys())]
>>> pair_sm
[('ARS', 0.138...),
 ('accuracy', 0.75),
 ('confusion', [[3, 0], [2, 3]]),
 ('f1_macro', 0.800...), ('f1_weighted', 0.849...),
 ('precision_macro', 0.800...), ('precision_weighted', 0.75),
 ('recall_macro', 0.749...), ('recall_weighted', 0.749...),
 ('support_macro', None), ('support_weighted', None)]
>>> y_true = np.array([0] * 5 + [1] * 5 + [2] * 5)
>>> y_pred = np.array([0] * 5 + [1] * 3 + [2] * 7)
>>> dist_sm = compute_classif_metrics(y_true, y_pred)
>>> pair_sm = [(n, dist_sm[n]) for n in sorted(dist_sm.keys())]
>>> pair_sm
[('ARS', 0.641...),
 ('accuracy', 0.866...),
 ('confusion', [[5, 0, 0], [0, 3, 2], [0, 0, 5]]),
 ('f1_macro', 0.904...), ('f1_weighted', 0.904...),
 ('precision_macro', 0.866...), ('precision_weighted', 0.866...),
 ('recall_macro', 0.861...), ('recall_weighted', 0.861...),
 ('support_macro', None), ('support_weighted', None)]
```

bpdl.metric_similarity.**compute_labels_overlap_matrix**(*seg1*, *seg2*)

compute overlap between tho segmentation atlasess) with same sizes

Parameters

- **seg1** – np.array<height, width>
- **seg2** – np.array<height, width>

Return ndarray np.array<height, width>

```
>>> seg1 = np.zeros((7, 15), dtype=int)
>>> seg1[1:4, 5:10] = 3
>>> seg1[5:7, 6:13] = 2
>>> seg2 = np.zeros((7, 15), dtype=int)
>>> seg2[2:5, 7:12] = 1
>>> seg2[4:7, 7:14] = 3
>>> compute_labels_overlap_matrix(seg1, seg1)
array([[76,  0,  0,  0],
       [ 0,  0,  0,  0],
       [ 0,  0, 14,  0],
       [ 0,  0,  0, 15]])
>>> compute_labels_overlap_matrix(seg1, seg2)
array([[63,  4,  0,  9],
       [ 0,  0,  0,  0],
       [ 2,  0,  0, 12],
       [ 9,  6,  0,  0]])
```

bpdl.metric_similarity.**relabel_max_overlap_merge**(*seg_ref*, *seg_relabel*, *keep_bg=True*)

relabel the second segmentation cu that maximise relative overlap for each pattern (object), if one pattern in reference atlas is likely composed from multiple patterns in relabel atlas, it merge them NOTE: it skips background class - 0

Parameters

- **seg_ref** (*ndarray*) – segmentation
- **seg_relabel** (*ndarray*) – segmentation
- **keep_bg** (*bool*) –

Return ndarray

```
>>> atlas1 = np.zeros((7, 15), dtype=int)
>>> atlas1[1:4, 5:10] = 1
>>> atlas1[5:7, 3:13] = 2
>>> atlas2 = np.zeros((7, 15), dtype=int)
>>> atlas2[0:3, 7:12] = 1
>>> atlas2[3:7, 1:7] = 2
>>> atlas2[4:7, 7:14] = 3
>>> atlas2[:2, :3] = 5
>>> relabel_max_overlap_merge(atlas1, atlas2, keep_bg=True)
array([[1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0]])
>>> relabel_max_overlap_merge(atlas2, atlas1, keep_bg=True)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0],
       [0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0]])
>>> relabel_max_overlap_merge(atlas1, atlas2, keep_bg=False)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0]])
```

bpdl.metric_similarity.**relabel_max_overlap_unique**(*seg_ref*, *seg_relabel*, *keep_bg=True*)

relabel the second segmentation cu that maximise relative overlap for each pattern (object), the relation among patterns is 1-1 NOTE: it skips background class - 0

Parameters

- **seg_ref** (*ndarray*) – segmentation
- **seg_relabel** (*ndarray*) – segmentation
- **keep_bg** (*bool*) –

Return ndarray

```
>>> atlas1 = np.zeros((7, 15), dtype=int)
>>> atlas1[1:4, 5:10] = 1
>>> atlas1[5:7, 3:13] = 2
>>> atlas2 = np.zeros((7, 15), dtype=int)
>>> atlas2[0:3, 7:12] = 1
>>> atlas2[3:7, 1:7] = 2
>>> atlas2[4:7, 7:14] = 3
>>> atlas2[:2, :3] = 5
>>> relabel_max_overlap_unique(atlas1, atlas2, keep_bg=True)
array([[5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0]])
>>> relabel_max_overlap_unique(atlas2, atlas1, keep_bg=True)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0],
       [0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0]])
>>> relabel_max_overlap_unique(atlas1, atlas2, keep_bg=False)
array([[5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0]])
```

bpdl.pattern_atlas module

bpdl.pattern_weights module

Estimating pattern weight vector for each image

Copyright (C) 2015-2020 Jiri Borovec <jiri.borovec@fel.cvut.cz>

bpdl.pattern_weights.**convert_weights_binary2indexes**(*weights*)
convert binary matrix oof weights to list of indexes o activated ptns

Parameters **weights** (*ndarray*) – np.array<nb_imgs, nb_lbs>

Return list(list(int))

```
>>> weights = np.array([[ 0,  0,  1,  0],
...                     [ 0,  0,  0,  1],
...                     [ 1,  0,  0,  0],
...                     [ 0,  0,  1,  1],
...                     [ 1,  0,  0,  0]])
>>> convert_weights_binary2indexes(weights)
[array([2]), array([3]), array([0]), array([2, 3]), array([0])]
```

bpdl.pattern_weights.**initialise_weights_random**(*nb_imgs*, *nb_patterns*, *ratio_select=0.2*, *rand_seed=None*)

Parameters

- **nb_imgs** (*int*) – number of all images
- **nb_patterns** (*int*) – number of all available labels
- **ratio_select** (*float*) – number <0, 1> defining how many should be set on, 1 means all and 0 means none
- **rand_seed** – random initialization

Return ndarray np.array<nb_imgs, nb_labels>

```
>>> initialise_weights_random(5, 3, rand_seed=0)
array([[ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  1.],
       [ 1.,  0.,  0.,  0.]])
```

bpdl.pattern_weights.**weights_image_atlas_overlap_major**(*img*, *atlas*)

Parameters

- **img** (*ndarray*) – image np.array<height, width>
- **atlas** (*ndarray*) – image np.array<height, width>

Return list(int) list(int) * nb_lbs of values {0, 1}

```
>>> atlas = np.zeros((8, 10), dtype=int)
>>> atlas[:3, 0:4] = 1
>>> atlas[3:7, 5:10] = 2
>>> img = np.array([0, 1, 0])[atlas]
>>> weights_image_atlas_overlap_major(img, atlas)
[1, 0]
```

(continues on next page)

(continued from previous page)

```
>>> img = [[0.46, 0.62, 0.62, 0.46, 0.2,  0.04, 0.01, 0.0,  0.0,  0.0],
...        [0.44, 0.59, 0.59, 0.44, 0.2,  0.06, 0.04, 0.04, 0.04, 0.04],
...        [0.33, 0.44, 0.44, 0.34, 0.2,  0.17, 0.19, 0.2,  0.2,  0.2],
...        [0.14, 0.19, 0.19, 0.17, 0.2,  0.34, 0.44, 0.46, 0.47, 0.47],
...        [0.03, 0.04, 0.04, 0.06, 0.2,  0.44, 0.59, 0.62, 0.62, 0.62],
...        [0.0,  0.0,  0.01, 0.04, 0.19, 0.44, 0.59, 0.62, 0.62, 0.62],
...        [0.0,  0.0,  0.0,  0.03, 0.14, 0.33, 0.44, 0.46, 0.47, 0.47],
...        [0.0,  0.0,  0.0,  0.01, 0.06, 0.14, 0.19, 0.2,  0.2,  0.2]]
>>> weights_image_atlas_overlap_major(np.array(img), atlas)
[0, 1]
```

bpdl.pattern_weights.**weights_image_atlas_overlap_partial**(*img*, *atlas*)

Parameters

- **img** (*ndarray*) – image np.array<height, width>
- **atlas** (*ndarray*) – image np.array<height, width>

Return list(int) list(int) * nb_lbs of values {0, 1}

```
>>> atlas = np.zeros((8, 10), dtype=int)
>>> atlas[:3, 0:4] = 1
>>> atlas[3:7, 5:10] = 2
>>> img = np.array([0, 1, 0])[atlas]
>>> weights_image_atlas_overlap_partial(img, atlas)
[1, 0]
>>> img = [[0.46, 0.62, 0.62, 0.46, 0.2,  0.04, 0.01, 0.0,  0.0,  0.0],
...        [0.44, 0.59, 0.59, 0.44, 0.2,  0.06, 0.04, 0.04, 0.04, 0.04],
...        [0.33, 0.44, 0.44, 0.34, 0.2,  0.17, 0.19, 0.2,  0.2,  0.2],
...        [0.14, 0.19, 0.19, 0.17, 0.2,  0.34, 0.44, 0.46, 0.47, 0.47],
...        [0.03, 0.04, 0.04, 0.06, 0.2,  0.44, 0.59, 0.62, 0.62, 0.62],
...        [0.0,  0.0,  0.01, 0.04, 0.19, 0.44, 0.59, 0.62, 0.62, 0.62],
...        [0.0,  0.0,  0.0,  0.03, 0.14, 0.33, 0.44, 0.46, 0.47, 0.47],
...        [0.0,  0.0,  0.0,  0.01, 0.06, 0.14, 0.19, 0.2,  0.2,  0.2]]
>>> weights_image_atlas_overlap_partial(np.array(img), atlas)
[1, 1]
```

bpdl.pattern_weights.**weights_image_atlas_overlap_threshold**(*img*, *atlas*, *threshold=1.0*)

estimate what patterns are activated with given atlas and input image compute overlap matrix and eval nr of overlapping and non pixels and threshold

Parameters

- **img** (*ndarray*) – image np.array<height, width>
- **atlas** (*ndarray*) – image np.array<height, width>
- **threshold** (*float*) – represent the ration between overlapping and non pixels

Return list(int) list(int) * nb_lbs of values {0, 1}

bpdl.pattern_weights.**weights_label_atlas_overlap_threshold**(*imgs*, *atlas*, *label*, *threshold=1.0*)

estimate what patterns are activated with given atlas and input image compute overlap matrix and eval nr of overlapping and non pixels and threshold

Parameters

- **imgs** (*list(ndarray)*) – list of images np.array<height, width>

- **atlas** (*ndarray*) – image np.array<height, width>
- **label** (*int*) –
- **threshold** (*float*) – represent the ration between overlapping and non pixels

Return ndarray np.array<nb_imgs> of values {0, 1}

```
>>> atlas = np.zeros((8, 12), dtype=int)
>>> atlas[:3, 1:5] = 1
>>> atlas[3:7, 6:12] = 2
>>> luts = np.array([[0, 1, 0]] * 3 + [[0, 0, 1]] * 3 + [[0, 1, 1]] * 3)
>>> imgs = [lut[atlas] for lut in luts]
>>> atlas[atlas == 2] = 0
>>> weights_label_atlas_overlap_threshold(imgs, atlas, 2)
array([0, 0, 0, 0, 0, 0, 0, 0, 0])
```

bpdl.registration module

bpdl.utilities module

The basic module for generating synthetic images and also loading / exporting

Copyright (C) 2015-2020 Jiri Borovec <jiri.borovec@fel.cvut.cz>

bpdl.utilities.**convert_numerical**(*s*)
try to convert a string tu numerical

Parameters **s** (*str*) – input string

Returns

```
>>> convert_numerical('-1')
-1
>>> convert_numerical('-2.0')
-2.0
>>> convert_numerical('.1')
0.1
>>> convert_numerical('-0.')
-0.0
>>> convert_numerical('abc58')
'abc58'
```

bpdl.utilities.**create_clean_folder**(*path_dir*)
create empty folder and while the folder exist clean all files

Parameters **path_dir** (*str*) – path

Return str

```
>>> path_dir = os.path.abspath('sample_dir')
>>> path_dir = create_clean_folder(path_dir)
>>> os.path.exists(path_dir)
True
>>> shutil.rmtree(path_dir, ignore_errors=True)
```

bpdl.utilities.**estimate_max_circle**(*point*, *direction*, *points*, *max_diam=1000*, *step_tol=0.001*)
find maximal circe from a given pont in orthogonal direction which just touch the curve with points

Parameters

- **point** (*tuple(float,float)*) – particular point on curve
- **direction** (*tuple(float,float)*) – orthogonal direction
- **float]] points** (*[[float,*) – list of point on curve
- **max_diam** (*float*) – maximal diameter
- **step_tol** (*float*) – tolerance step in dividing diameter interval

Returns

```
>>> y = [1] * 10
>>> pts = np.array(list(zip(range(len(y)), y)))
>>> estimate_max_circle([5, 1], [0, 1], pts)
999.99...
>>> y = [1] * 6 + [2] * 4
>>> pts = np.array(list(zip(range(len(y)), y)))
>>> estimate_max_circle([4, 1], [0, 1], pts)
4.99...
```

bpdl.utilities.**estimate_point_max_circle**(*idx*, *points*, *tangent_smooth=1*, *orient=1.0*, *max_diam=1000000.0*, *step_tol=0.001*)

estimate maximal circle from a particular point on curve

Parameters

- **idx** (*int*) – index or point on curve
- **float]] points** (*[[float,*) – list of point on curve
- **tangent_smooth** (*int*) – distance for tangent
- **direct** (*float*) – positive or negative ortogonal
- **max_diam** (*float*) – maximal diameter
- **step_tol** (*float*) – tolerance step in dividing diameter interval

Returns

```
>>> y = [1] * 25 + list(range(1, 50)) + [50] * 25
>>> pts = np.array(list(zip(range(len(y)), y)))
>>> estimate_point_max_circle(0, pts)
60.38...
>>> estimate_point_max_circle(30, pts)
17.14...
>>> estimate_point_max_circle(90, pts)
999999.99...
```

bpdl.utilities.**estimate_rolling_ball**(*points*, *tangent_smooth=1*, *max_diam=1000000.0*, *step_tol=0.001*)

roll a ball over curve and get for each particular position a maximal ball which does not intersect the rest of curve

Parameters

- **points** –
- **tangent_smooth** –
- **max_diam** –
- **step_tol** –

Returns

```
>>> y = [1] * 6 + [2] * 4
>>> pts = np.array(list(zip(range(len(y)), y)))
>>> diams = estimate_rolling_ball(pts)
>>> list(map(int, diams[0]))
[24, 18, 12, 8, 4, 1, 9, 999999, 999999, 999999]
>>> list(map(int, diams[1]))
[999999, 999999, 999999, 999999, 999999, 10, 1, 4, 8, 12]
```

bpdl.utilities.**generate_gauss_2d**(*mean*, *std*, *im_size=None*, *norm=None*)
Generating a Gaussian distribution in 2D image

Parameters

- **norm** (*float*) – normalise the maximal value
- **mean** (*list (int)*) – mean position
- **std** (*list (list (int))*) – STD
- **im_size** (*tuple (int, int)*) – optional image size

Return ndarray

```
>>> im = generate_gauss_2d([4, 5], [[1, 0], [0, 2]], (8, 10), norm=1.)
>>> np.round(im, 1)
array([[ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0.1,  0.1,  0.1,  0.1,  0.1,  0. ,  0. ],
       [ 0. ,  0.1,  0.2,  0.4,  0.5,  0.6,  0.5,  0.4,  0.2,  0.1],
       [ 0. ,  0.1,  0.3,  0.6,  0.9,  1. ,  0.9,  0.6,  0.3,  0.1],
       [ 0. ,  0.1,  0.2,  0.4,  0.5,  0.6,  0.5,  0.4,  0.2,  0.1],
       [ 0. ,  0. ,  0. ,  0.1,  0.1,  0.1,  0.1,  0.1,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ]])
>>> im = generate_gauss_2d([2, 3], [[1., 0], [0, 1.2]])
>>> np.round(im, 2)
array([[ 0.  ,  0.  ,  0.01,  0.02,  0.01,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.02,  0.06,  0.08,  0.06,  0.02,  0.  ,  0.  ],
       [ 0.01,  0.03,  0.09,  0.13,  0.09,  0.03,  0.01,  0.  ],
       [ 0.  ,  0.02,  0.06,  0.08,  0.06,  0.02,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.01,  0.02,  0.01,  0.  ,  0.  ,  0.  ]])
```

bpdl.utilities.**is_iterable**(*var*)
check if the variable is iterable

Parameters **var** –

Return bool

```
>>> is_iterable('abc')
False
>>> is_iterable(123.)
False
>>> is_iterable((1, ))
True
>>> is_iterable(range(2))
True
```

bpdl.utilities.**is_list_like**(*var*)
check if the variable is iterable

Parameters **var** –

Return bool

```
>>> is_list_like('abc')
False
>>> is_list_like(123.)
False
>>> is_list_like([0])
True
>>> is_list_like((1, ))
True
>>> is_list_like(range(2))
True
```

1.2.2 Module contents

BPDL - Binary pattern Dictionary Learning

1.3 experiments package

1.3.1 Submodules

experiments.experiment_general module

experiments.experiment_methods module

1.3.2 Module contents

1.4 Examples

1.4.1 Binary Pattern Dictionary Learning

We present an image processing pipeline which accepts a large number of images, containing spatial expression information for thou- sands of genes in Drosophila imaginal discs. We assume that the gene activations are binary and can be expressed as a union of a small set of non-overlapping spatial patterns, yielding a compact representation of the spatial activation of each gene. This lends itself well to further automatic analysis, with the hope of discovering new biological relation- ships. Traditionally, the images were labeled manually, which was very time consuming. The key part of our work is a binary pattern dictionary learning algorithm, that takes a set of binary images and determines a set of patterns, which can be used to represent the input images with a small error.

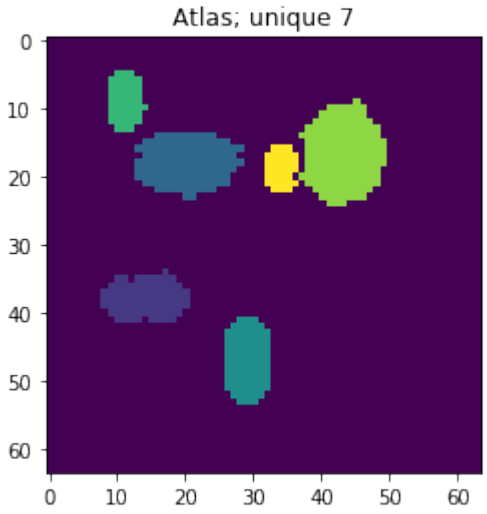
[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

```
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

load datset

[2]:
```
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

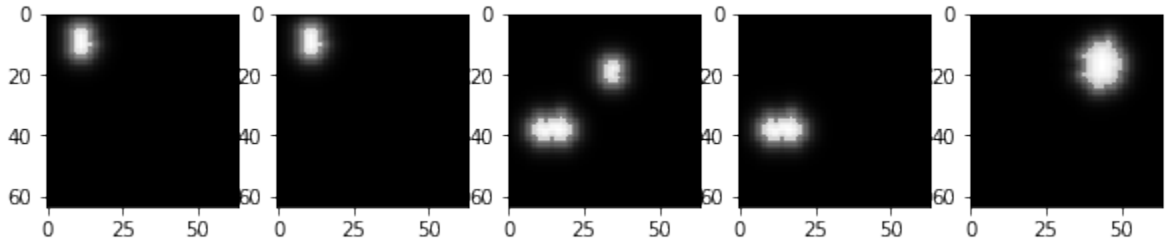
```
loading dataset: (True) exists -> /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_
↪v0/datasetFuzzy_raw
loading (True) <- /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_v0/dictionary/
↪atlas.png
```



[3]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded # images: ', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)
```

```
('loaded # images: ', 800)
('image shape:', (64, 64))
```

[8]:
```
plt.figure(figsize=(10, 2))
for i in range(5):
    plt.subplot(1, 5, i + 1), plt.imshow(list_imgs[i], cmap=plt.cm.Greys_r)
```

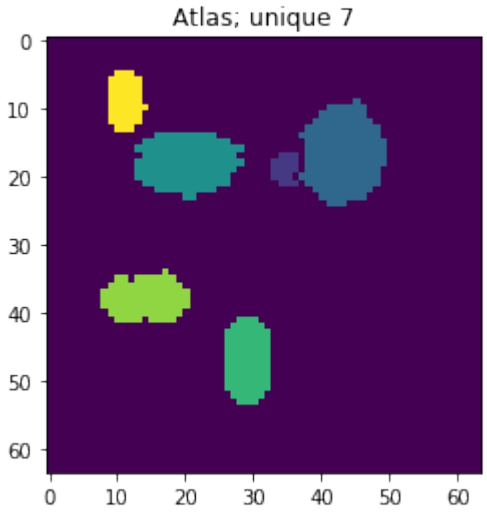


BPDL

[9]:
```
from bpdl import dictionary_learning as dl
from bpdl import pattern_atlas as ptn_dict

init_atlas_msc = ptn_dict.init_atlas_mosaic(list_imgs[0].shape, 6)
# init_encode_rnd = ptn_weigth.initialise_weights_random(len(imgs), np.max(atlas))

atlas, w_bins, deforms = dl.bpdl_pipeline(list_imgs, init_atlas=init_atlas_msc, max_
↪iter=9)
```

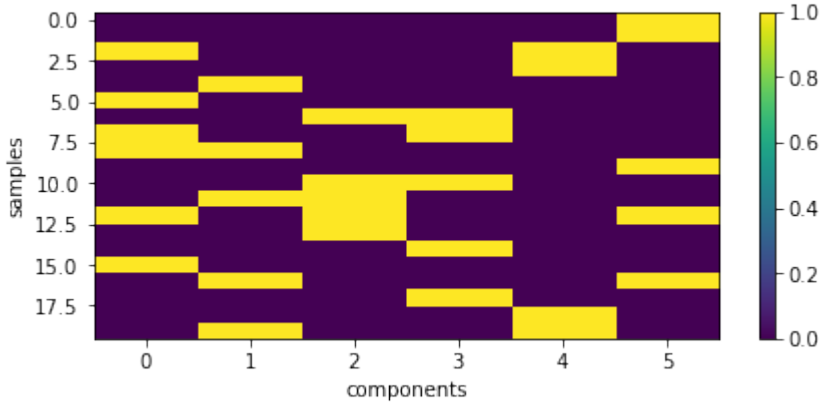
show the estimated components - dictionary

[10]:
```
plt.imshow(atlas, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % len(np.unique(atlas)))
```



particular coding of each sample

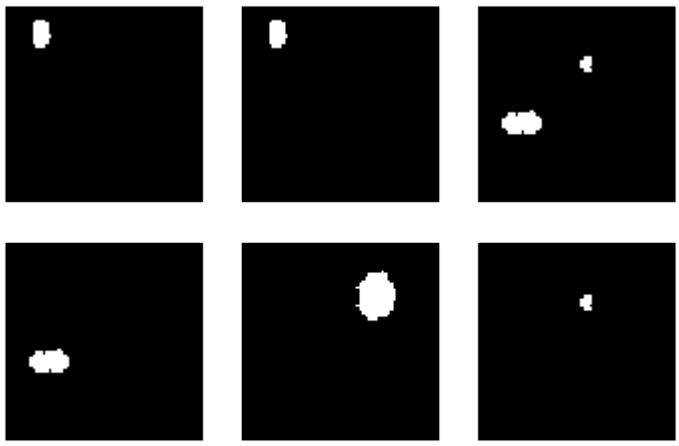
[11]:
```
plt.figure(figsize=(7, 3))
plt.imshow(w_bins[:20, :], interpolation='nearest', aspect='auto'), plt.colorbar()
_= plt.xlabel('components'), plt.ylabel('samples')
```



backword reconstruction from encoding and dictionary

[12]:
```
img_rct = ptn_dict.reconstruct_samples(atlas, w_bins)
```

[17]:
```
plt.figure(figsize=(6, 4))
for i in range(6):
    plt.subplot(2, 3, i + 1), plt.imshow(img_rct[i], cmap=plt.cm.Greys_r), plt.axis(
↪'Off')
```



[ ]:
```
```

1.4.2 Generic dictionary learning

Reference: Dictionary Learning

Dictionary learning is a matrix factorization problem that amounts to finding a(usually overcomplete) dictionary that will perform good at sparsely encoding the fitted data.

Representing data as sparse combinations of atoms from an overcomplete dictionary is suggested to be the way the mammal primary visual cortex works. Consequently, dictionary learning applied on image patches has been shown to give good results in image processing tasks such as image completion, inpainting and denoising, as well as for supervised recognition tasks.

Dictionary learning is an optimization problem solved by alternatively updating the sparse code, as a solution to multiple Lasso problems, considering the dictionary fixed, and then updating the dictionary to best fit the sparse code.

$$(U^{\cdot}V^{)} = argmin_{(U,V)}0.5||Y - UV||_2^2 + alpha * ||U||_1$$

with $||V_k||_2 = 1$ for all $0 <= k < n_{components}$

After using such a procedure to fit the dictionary, the transform is simply a sparse coding step that shares the same implementation with all dictionary learning objects(see Sparse coding with a precomputed dictionary).

Using: sklearn.decomposition.DictionaryLearning

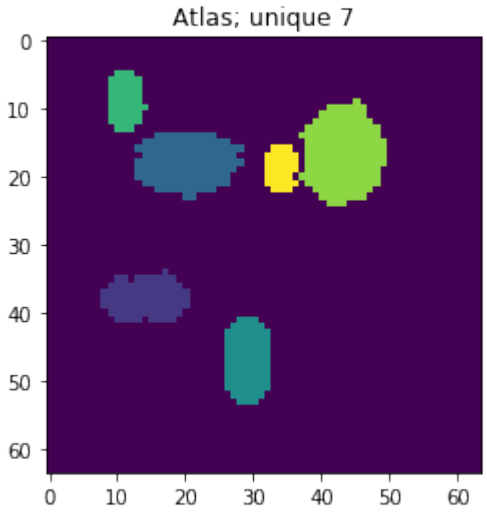
[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

```
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

load datset

[2]:
```
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

```
loading dataset: (True) exists -> /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_
↪v0/datasetFuzzy_raw
loading (True) <- /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_v0/dictionary/
↪atlas.png
```



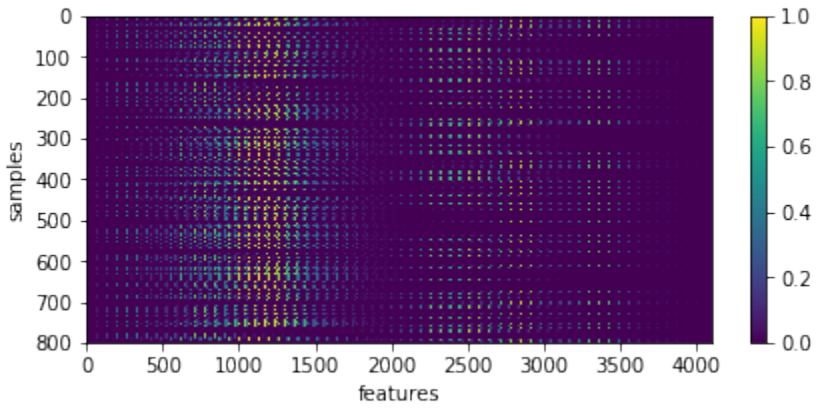
[3]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded # images: ', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)
```

```
loaded # images:  800
image shape: (64, 64)
```

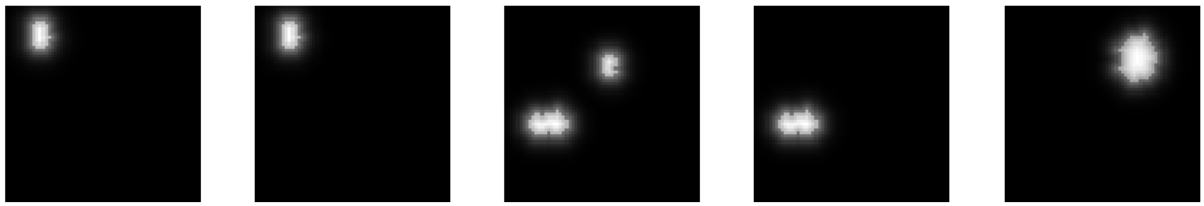
Pre-Processing

[5]:
```
X = np.array([im.ravel() for im in list_imgs]) # - 0.5
print ('input data shape:', X.shape)

plt.figure(figsize=(7, 3))
_= plt.imshow(X, aspect='auto'), plt.xlabel('features'), plt.ylabel('samples'), plt.
↪colorbar()
```

```
input data shape: (800, 4096)
```



[6]:
```
uts.show_sample_data_as_imgs(X, img_shape, nb_rows=1, nb_cols=5)
```



Generic dictionary learning

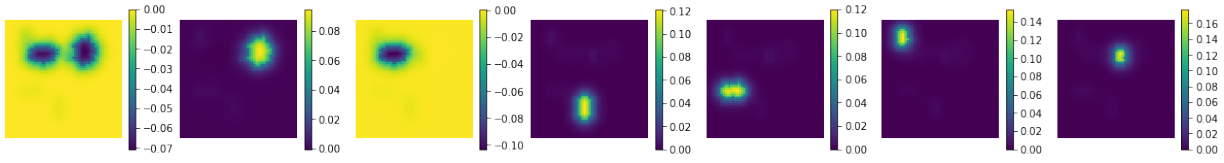
[7]:
```
from sklearn.decomposition import DictionaryLearning
dl = DictionaryLearning(n_components=nb_patterns, fit_algorithm='lars', transform_
↪algorithm='omp',
                        n_jobs=-1, max_iter=200, split_sign=False)

X_new = dl.fit_transform(X[:1200, :])
print ('fitting parameters:', dl.get_params())
print ('number of iteration:', dl.n_iter_)
```

```
fitting parameters: {'n_jobs': -1, 'code_init': None, 'max_iter': 200, 'dict_init':␣
↪None, 'fit_algorithm': 'lars', 'random_state': None, 'n_components': 7, 'tol': 1e-
↪08, 'transform_algorithm': 'omp', 'alpha': 1, 'transform_alpha': None, 'transform_n_
↪nonzero_coefs': None, 'split_sign': False, 'verbose': False}
number of iteration: 50
```

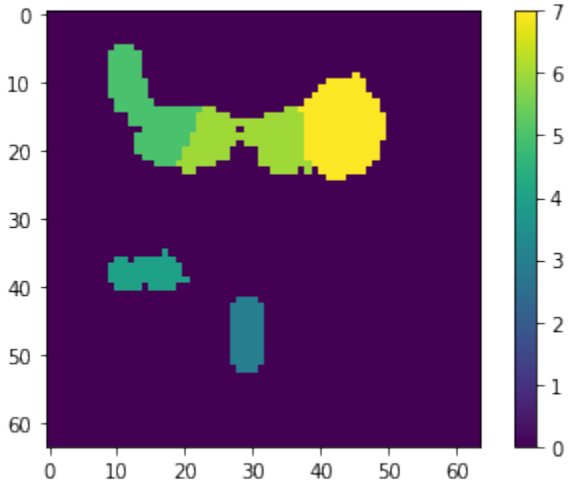
show the estimated components - dictionary

[8]:
```
comp = dl.components_
coefs = np.sum(np.abs(X_new), axis=0)
print ('estimated component matrix:', comp.shape)

compSorted = [c[0] for c in sorted(zip(comp, coefs), key=lambda x: x[1],␣
↪reverse=True) ]
uts.show_sample_data_as_imgs(np.array(compSorted), img_shape, nb_cols=nb_patterns,␣
↪bool_clr=True)
```

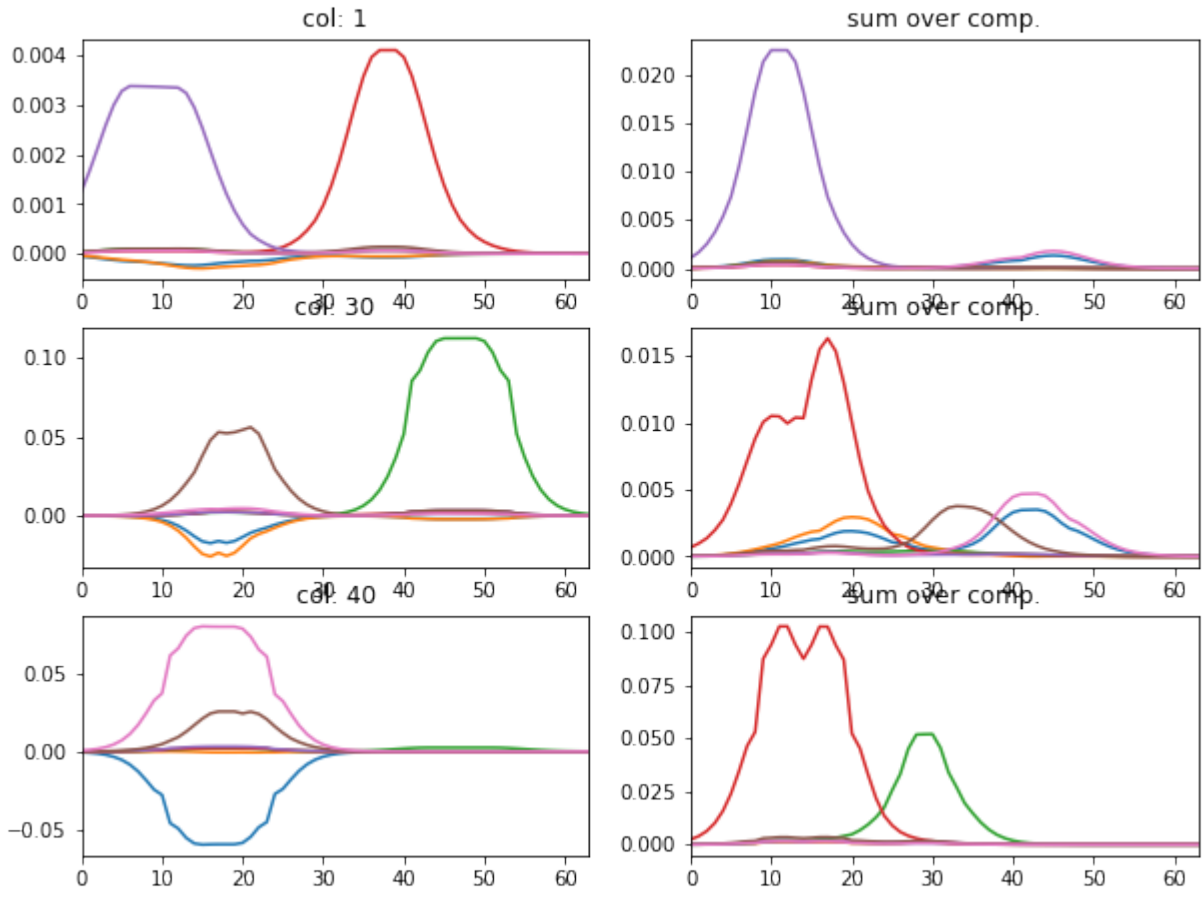
```
estimated component matrix: (7, 4096)
```



[9]:
```
ptn_used = np.sum(np.abs(X_new), axis=0) > 0
atlas_ptns = comp[ptn_used, :].reshape((-1, ) + list_imgs[0].shape)

atlas_ptns = comp.reshape((-1, ) + list_imgs[0].shape)
atlas_estim = np.argmax(atlas_ptns, axis=0) + 1
atlas_sum = np.sum(np.abs(atlas_ptns), axis=0)
atlas_estim[atlas_sum < 1e-1] = 0
_ = plt.imshow(atlas_estim, interpolation='nearest'), plt.colorbar()
```

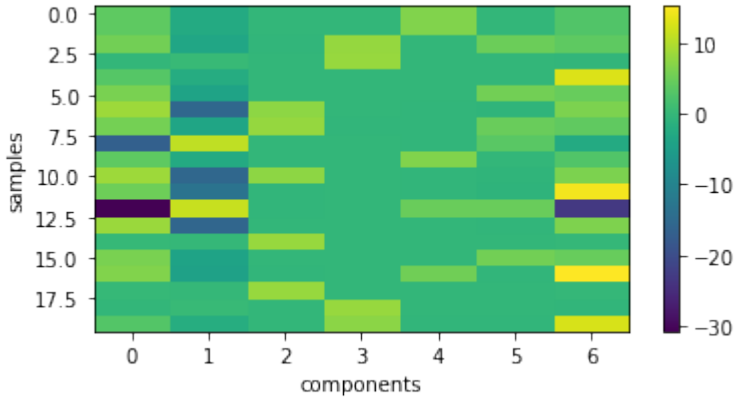


[10]:
```
l_idx = [1, 30, 40]
fig, axr = plt.subplots(len(l_idx), 2, figsize=(10, 2.5*len(l_idx)))
for i, idx in enumerate(l_idx):
    axr[i, 0].plot(atlas_ptns[:,:,idx].T), axr[i, 0].set_xlim([0, 63])
    axr[i, 0].set_title('col: {}'.format(idx))
    axr[i, 1].plot(np.abs(atlas_ptns[:,idx,:].T)), axr[i, 1].set_xlim([0, 63])
    axr[i, 1].set_title('sum over comp.')
```



particular coding of each sample

[11]:
```
plt.figure(figsize=(6, 3))
plt.imshow(X_new[:20,:], interpolation='nearest', aspect='auto'), plt.colorbar()
_= plt.xlabel('components'), plt.ylabel('samples')

coefs = np.sum(np.abs(X_new), axis=0)
# print 'used coeficients:', coefs.tolist()
```

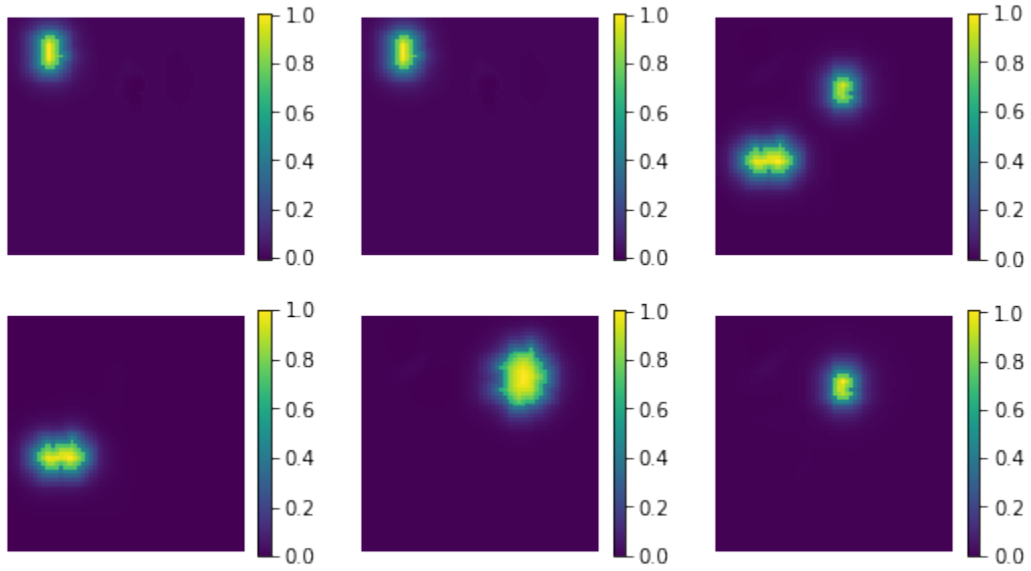


backword reconstruction from encoding and dictionary

[12]:
```
res = np.dot(X_new, comp)
print ('model applies by reverting the unmixing', res.shape)
```

```
model applies by reverting the unmixing (200, 4096)
```

[15]:
```
uts.show_sample_data_as_imgs(res, img_shape, nb_rows=2, nb_cols=3, bool_clr=True)
```



[ ]:

1.4.3 Indipendent Component Analyses

sklearn.decomposition.FastICA

Independent component analysis separates a multivariate signal into additive subcomponents that are maximally independent. It is implemented in scikit-learn using the Fast ICA algorithm. Typically, ICA is not used for reducing dimensionality but for separating superimposed signals. Since the ICA model does not include a noise term, for the model to be correct, whitening must be applied. This can be done internally using the whiten argument or manually using one of the PCA variants.

Usable examples: Faces dataset decompositions

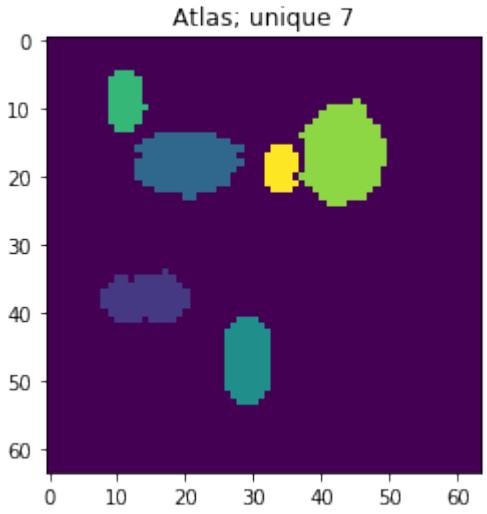
[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

```
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

load datset

[2]:
```
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

```
loading dataset: (True) exists -> /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_
↪v0/datasetFuzzy_raw
loading (True) <- /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_v0/dictionary/
↪atlas.png
```



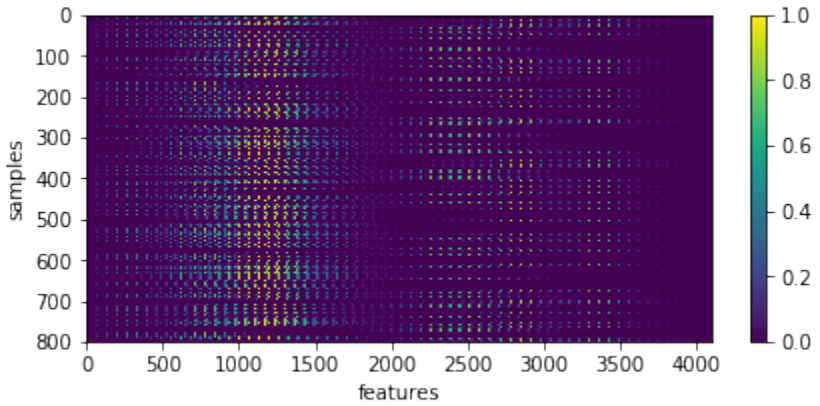
[3]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded images #', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)
```

```
loaded images # 800
image shape: (64, 64)
```

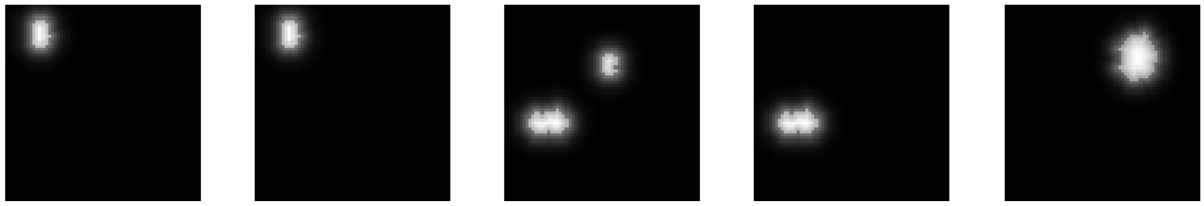
Pre-Processing

[15]:
```
X = np.array([im.ravel() for im in list_imgs]) # - 0.5
print ('input data shape:', X.shape)

plt.figure(figsize=(7, 3))
_= plt.imshow(X, aspect='auto'), plt.xlabel('features'), plt.ylabel('samples'), plt.
↪colorbar()
```

```
input data shape: (800, 4096)
```



[6]:
```
uts.show_sample_data_as_imgs(X, img_shape, nb_rows=1, nb_cols=5)
```



FastICA

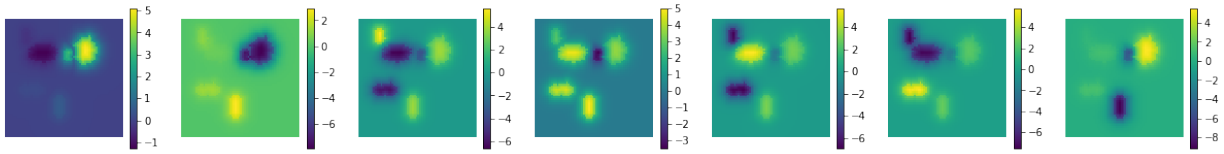
[7]:
```
from sklearn.decomposition import FastICA
ica = FastICA(n_components=nb_patterns, max_iter=999, whiten=True)

X_new = ica.fit_transform(X) # Reconstruct signals
print ('fitting parameters:', ica.get_params())
print ('number of iteration:', ica.n_iter_)
```

```
fitting parameters: {'fun_args': None, 'algorithm': 'parallel', 'max_iter': 999,
↪'random_state': None, 'n_components': 7, 'tol': 0.0001, 'fun': 'logcosh', 'w_init':␣
↪None, 'whiten': True}
number of iteration: 19
```

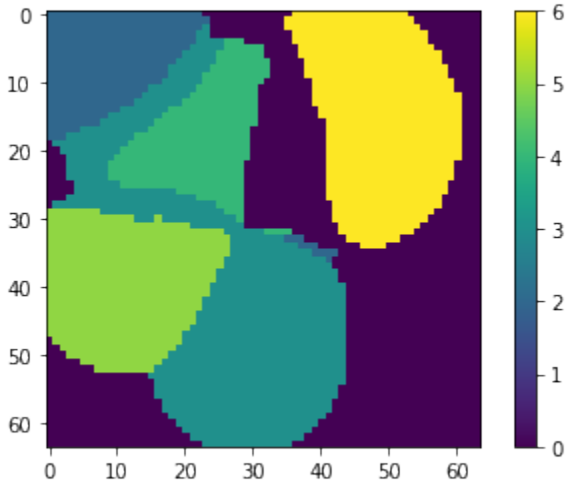
representation of estimated components - disctionary

[8]:
```
print ('estimated mixing matrix:', ica.mixing_.shape)
# print 'ICA mean:', ica.mean_

uts.show_sample_data_as_imgs(ica.mixing_.T, img_shape, nb_cols=nb_patterns, bool_
↪clr=True)
```

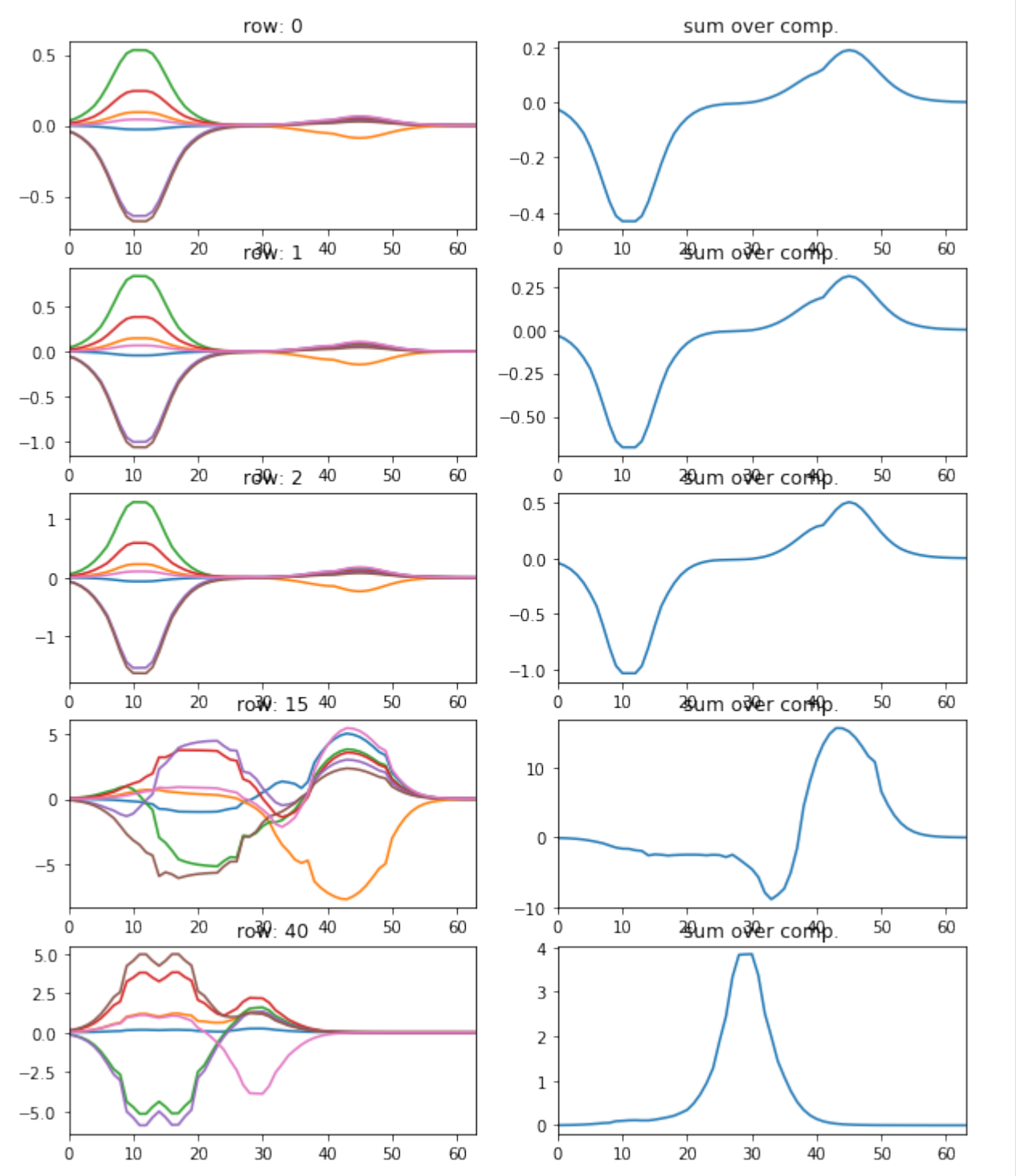
```
estimated mixing matrix: (4096, 7)
```



[9]:
```
comp = ica.mixing_.T
ptn_used = np.sum(np.abs(X_new), axis=0) > 0
atlas_ptns = comp[ptn_used, :].reshape((-1, ) + list_imgs[0].shape)

atlas_ptns = comp.reshape((-1, ) + list_imgs[0].shape)
atlas_estim = np.argmax(atlas_ptns, axis=0)
atlas_sum = np.sum(np.abs(atlas_ptns), axis=0)
atlas_estim[atlas_sum < 1e-1] = 0
_ = plt.imshow(atlas_estim, interpolation='nearest'), plt.colorbar()
```

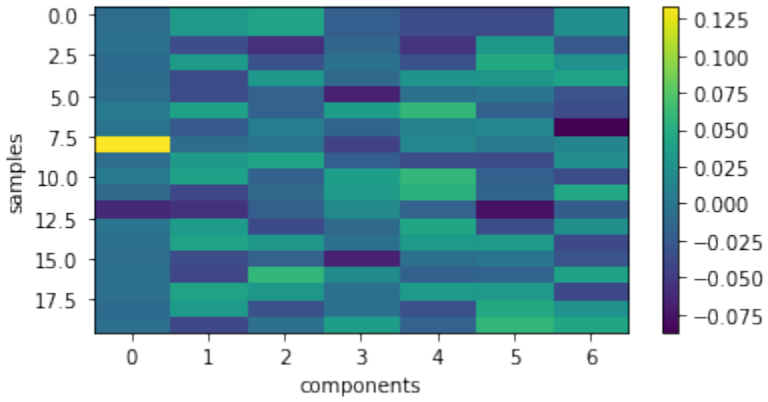


[11]:
```
l_idx = [0, 1, 2, 15, 40]
fig, axr = plt.subplots(len(l_idx), 2, figsize=(10, 2.5 * len(l_idx)))
for i, idx in enumerate(l_idx):
    axr[i, 0].plot(atlas_ptns[:,idx,:].T), axr[i, 0].set_xlim([0, 63])
    axr[i, 0].set_title('row: {}'.format(idx))
    axr[i, 1].plot(np.sum(atlas_ptns[:,idx,:].T, axis=1)), axr[i, 1].set_xlim([0, 63])
    axr[i, 1].set_title('sum over comp.')
```



show the particular coding of each sample

[12]:
```
plt.figure(figsize=(6, 3))
plt.imshow(X_new[:20,:], interpolation='nearest', aspect='auto'), plt.colorbar()
_= plt.xlabel('components'), plt.ylabel('samples')
```

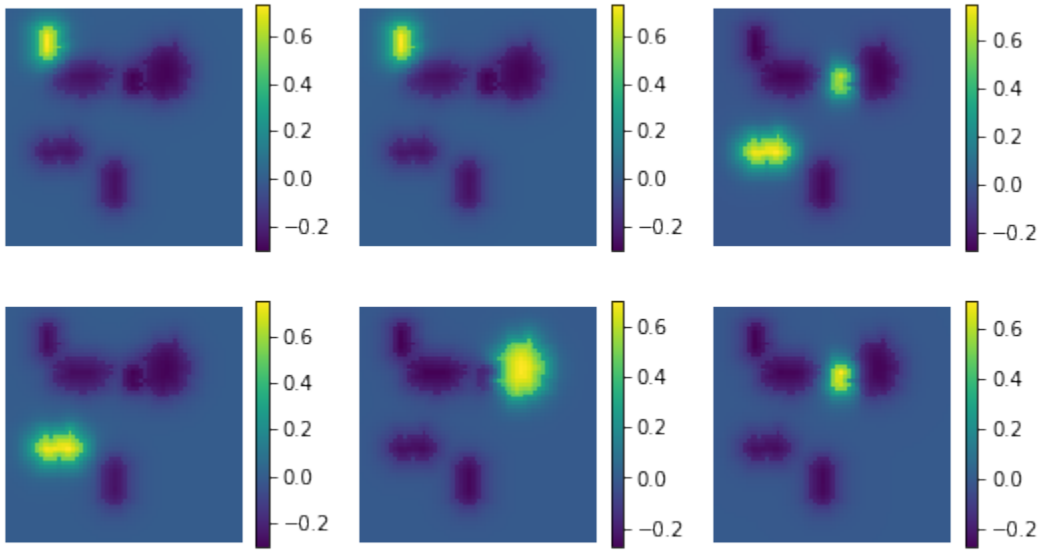


backword reconstruction from encoding and dictionary

[13]:
```
res = np.dot(X_new, ica.mixing_.T)# + ica.mean_
print ('ICA mean', ica.mean_)
print ('ICA model by reverting the unmixing', res.shape)
```

```
ICA mean [ 0.          0.          0.         ...,  0.00305882  0.00227451
  0.00213725]
ICA model by reverting the unmixing (200, 4096)
```

[17]:
```
uts.show_sample_data_as_imgs(res, img_shape, nb_rows=2, nb_cols=3, bool_clr=True)
```



[ ]:

1.4.4 Multi-subject dictionary learning & CanICA

Reference:

- Dictionary Learning and ICA for doing group analysis of resting-state fMRI
- nilearn.decomposition.DictLearning
- nilearn.decomposition.CanICA
- CanICA

This example applies dictionary learning and ICA to resting-state data, visualizing resulting components using atlas plotting tools.

Dictionary learning is a sparsity based decomposition method for extracting spatial maps. It extracts maps that are naturally sparse and usually cleaner than ICA. CanICA is an ICA method for group-level analysis of fMRI data. Compared to other strategies, it brings a well-controlled group model, as well as a thresholding algorithm controlling for specificity and sensitivity with an explicit model of the signal.

- G. Varoquaux et al. "A group model for stable multi-subject ICA on fMRI datasets", NeuroImage Vol 51 (2010), p. 288-299
- G. Varoquaux et al. "ICA-based sparse features recovery from fMRI datasets", IEEE ISBI 2010, p. 1177
- Gael Varoquaux et al. Multi-subject dictionary learning to segment an atlas of brain spontaneous activity Information Processing in Medical Imaging, 2011, pp. 562-573, Lecture Notes in Computer Science

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
import nibabel as nib
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

```
/mnt/datagrid/personal/borovec/Applications/vEnv2/lib/python2.7/site-packages/h5py/__
↪init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from␣
↪`float` to `np.floating` is deprecated. In future, it will be treated as `np.
↪float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
/mnt/datagrid/personal/borovec/Applications/vEnv2/lib/python2.7/site-packages/IPython/
↪html.py:14: ShimWarning: The `IPython.html` package has been deprecated. You should␣
↪import from `notebook` instead. `IPython.html.widgets` has moved to `ipywidgets`.
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

load datset

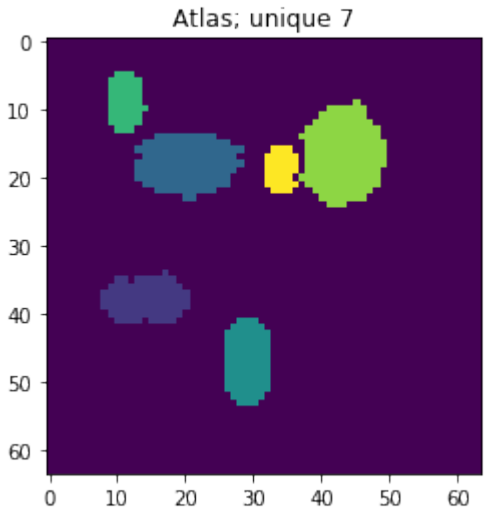
[2]:
```
# uts.DEFAULT_PATH = '/datagrid/Medical/microscopy/drosophila/synthetic_data/
↪atomicPatternDictionary_v0'
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
```

(continues on next page)

(continued from previous page)

```
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

```
loading dataset: (True) exists -> /datagrid/Medical/microscopy/drosophila/synthetic_
↪data/atomicPatternDictionary_v0/datasetFuzzy_raw
loading (True) <- /datagrid/Medical/microscopy/drosophila/synthetic_data/
↪atomicPatternDictionary_v0/dictionary/atlas.png
```

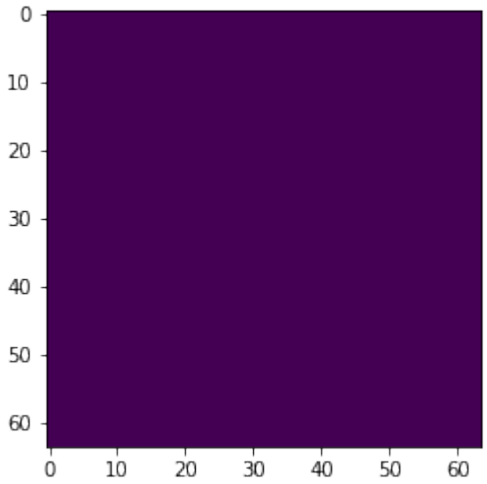


[3]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded # images: ', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)

nii_images = [nib.Nifti1Image(np.expand_dims(img, axis=0), affine=np.eye(4)) for img␣
↪in list_imgs]
mask_full = nib.Nifti1Image(np.expand_dims(np.ones(atlas_gt.shape, dtype=np.int8),␣
↪axis=0), affine=np.eye(4))
```

```
('loaded # images: ', 800)
('image shape:', (64, 64))
```

[17]:
```
from nilearn.input_data import NiftiMasker
masker = NiftiMasker(low_pass=0.5, verbose=0)
masker.fit(nii_images)
print ('shape:', masker.mask_img_.get_data().shape)
print ('values:', np.unique(masker.mask_img_.get_data()))
_= plt.imshow(masker.mask_img_.get_data()[0])
```

```
('shape:', (1, 64, 64))
('values:', array([1], dtype=int8))
```



CanICA

CanICA is an ICA method for group-level analysis of fMRI data. Compared to other strategies, it brings a well-controlled group model, as well as a thresholding algorithm controlling for specificity and sensitivity with an explicit model of the signal.

nilearn.decomposition.CanICA

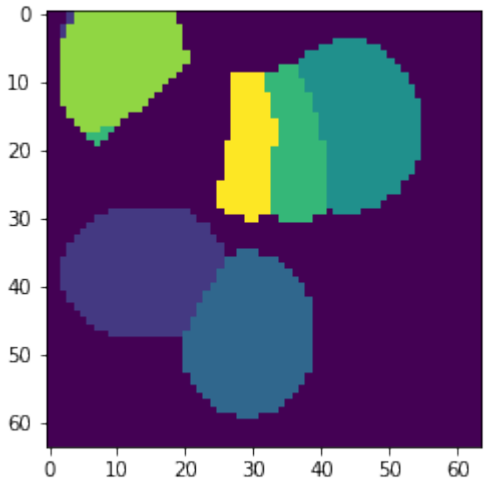
[10]:
```
from nilearn.decomposition import CanICA
canica = CanICA(mask=mask_full, n_components=nb_patterns,
                mask_strategy='background',
                threshold='auto', n_init=5, verbose=0)
canica.fit(nii_images)
```

```
/mnt/datagrid/personal/borovec/Applications/vEnv2/lib/python2.7/site-packages/nilearn/
↪signal.py:139: UserWarning: Detrending of 3D signal has been requested but would␣
↪lead to zero values. Skipping.
  warnings.warn('Detrending of 3D signal has been requested but '
/mnt/datagrid/personal/borovec/Applications/vEnv2/lib/python2.7/site-packages/nilearn/
↪signal.py:51: UserWarning: Standardization of 3D signal has been requested but␣
↪would lead to zero values. Skipping.
  warnings.warn('Standardization of 3D signal has been requested but '
```

[10]:
```
CanICA(detrend=True, do_cca=True, high_pass=None, low_pass=None,
    mask=<nibabel.nifti1.Nifti1Image object at 0x7f8634425510>,
    mask_args=None, mask_strategy='background',
    memory=Memory(cachedir=None), memory_level=0, n_components=7, n_init=5,
    n_jobs=1, random_state=None, smoothing_fwhm=6, standardize=True,
    t_r=None, target_affine=None, target_shape=None, threshold='auto',
    verbose=0)
```

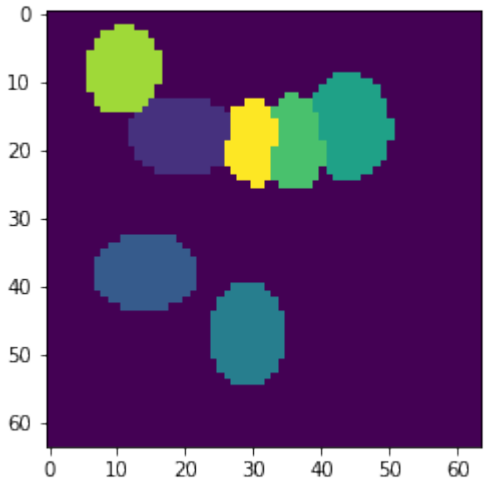
[11]:
```
components = np.argmax(canica.components_, axis=0) + 1
atlas = components.reshape(atlas_gt.shape)
plt.imshow(atlas)
```

[11]:
```
<matplotlib.image.AxesImage at 0x7f862c411ed0>
```



[12]:
```
max_ptn = np.max(canica.components_, axis=0).reshape(atlas_gt.shape)
atlas[max_ptn < np.mean(max_ptn[max_ptn > 0])] = 0
plt.imshow(atlas)
```

[12]:
```
<matplotlib.image.AxesImage at 0x7f862c31c190>
```



Dictionary Learning

Perform a map learning algorithm based on spatial component sparsity, over a CanICA initialization. This yields more stable maps than CanICA.

nilearn.decomposition.DictLearning

[15]:
```
from nilearn.decomposition import DictLearning
dict_learn = DictLearning(mask=mask_full, n_components=nb_patterns,
                          mask_strategy='background',
                          verbose=0, n_epochs=10)
dict_learn.fit(nii_images)
```

[15]:
```
DictLearning(alpha=10, batch_size=20, detrend=True, dict_init=None,
       high_pass=None, low_pass=None,
```

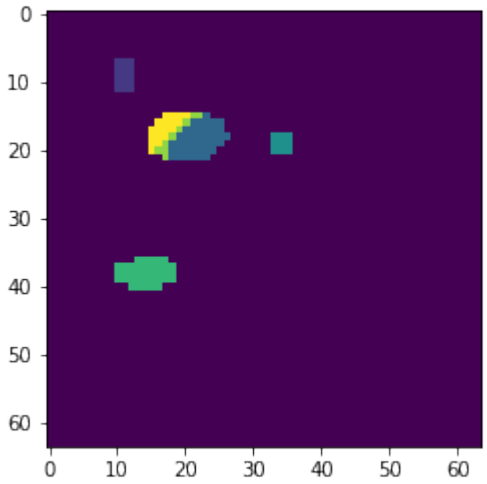
(continues on next page)

(continued from previous page)

```
        mask=<nibabel.nifti1.Nifti1Image object at 0x7f8634425510>,
        mask_args=None, mask_strategy='background',
        memory=Memory(cachedir=None), memory_level=0, method='cd',
        n_components=7, n_epochs=10, n_jobs=1, random_state=None,
        reduction_ratio='auto', smoothing_fwhm=4, standardize=True,
        t_r=None, target_affine=None, target_shape=None, verbose=0)
```

[16]:
```
components = np.argmax(dict_learn.components_, axis=0) + 1
atlas = components.reshape(atlas_gt.shape)
plt.imshow(atlas)
```

[16]:
```
<matplotlib.image.AxesImage at 0x7f862c25f3d0>
```



[ ]:
```
```

1.4.5 Non-negative matrix factorization

Reference: Non-negative matrix factorization

NMF is an alternative approach to decomposition that assumes that the data and the components are non-negative. NMF can be plugged in instead of PCA or its variants, in the cases where the data matrix does not contain negative values. It finds a decomposition of samples X into two matrices W and H of non-negative elements, by optimizing for the squared Frobenius norm:

$$\arg\min_{W,H} \frac{1}{2}||X - WH||^2_{Fro} = \frac{1}{2}\sum_{i,j}(X_{ij} - WH_{ij})^2$$

This norm is an obvious extension of the Euclidean norm to matrices. (Other optimization objectives have been suggested in the NMF literature, in particular Kullback-Leibler divergence, but these are not currently implemented.)

Using: sklearn.decomposition.NMF

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
```

(continues on next page)

(continued from previous page)

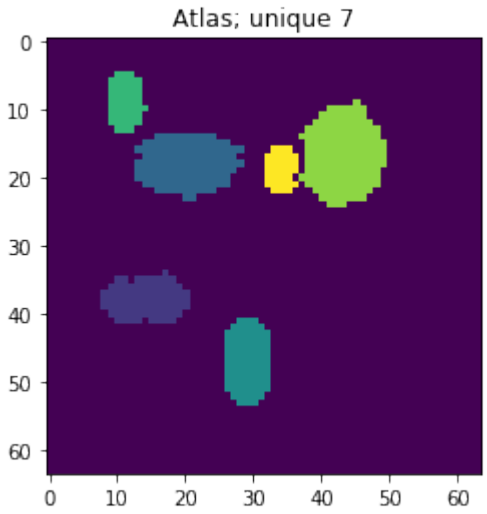
```
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

```
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

load datset

[2]:
```
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

```
loading dataset: (True) exists -> /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_
↪v0/datasetFuzzy_raw
loading (True) <- /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_v0/dictionary/
↪atlas.png
```



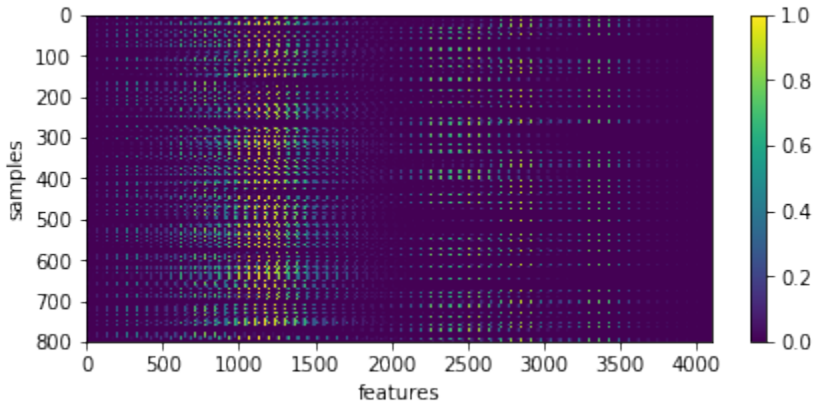
[3]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded # images: ', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)
```

```
loaded # images:  800
image shape: (64, 64)
```

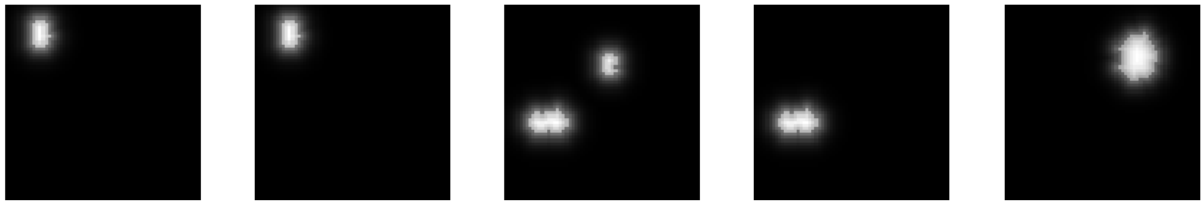
Pre-Processing

[5]:
```
X = np.array([im.ravel() for im in list_imgs])  # - 0.5
print ('input data shape:', X.shape)

plt.figure(figsize=(7, 3))
_= plt.imshow(X, aspect='auto'), plt.xlabel('features'), plt.ylabel('samples'), plt.
↪colorbar()
```

```
input data shape: (800, 4096)
```



[6]:
```
uts.show_sample_data_as_imgs(X, img_shape, nb_rows=1, nb_cols=5)
```



Non-negative matrix factorization

[7]:
```
from sklearn.decomposition import NMF
nmf = NMF(n_components=nb_patterns, max_iter=99)

X_new = nmf.fit_transform(X[:1200,:])
print ('fitting parameters:', nmf.get_params())
print ('number of iteration:', nmf.n_iter_)
```

```
fitting parameters: {'beta_loss': 'frobenius', 'shuffle': False, 'verbose': 0, 'solver
↪': 'cd', 'l1_ratio': 0.0, 'max_iter': 99, 'init': None, 'random_state': None, 'n_
↪components': 7, 'tol': 0.0001, 'alpha': 0.0}
number of iteration: 98
```

show the estimated components - dictionary

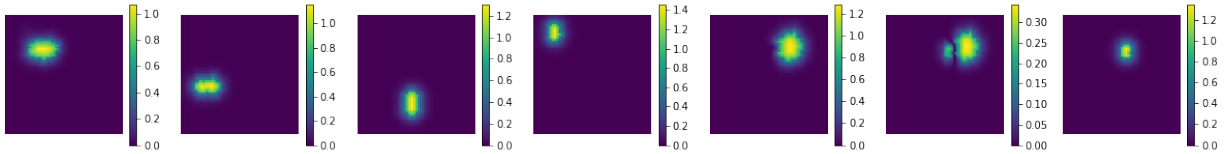
[8]:
```
comp = nmf.components_
coefs = np.sum(np.abs(X_new), axis=0)
print ('estimated component matrix:', comp.shape)
```

(continues on next page)

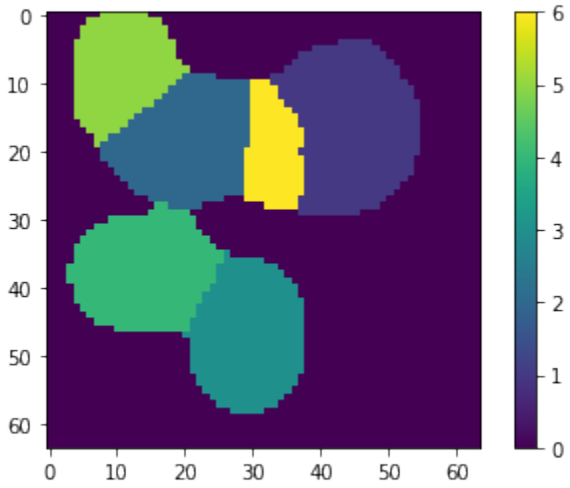
(continued from previous page)

```
compSorted = [c[0] for c in sorted(zip(comp, coefs), key=lambda x: x[1],␣
↪reverse=True) ]
uts.show_sample_data_as_imgs(np.array(compSorted), img_shape, nb_cols=nb_patterns,␣
↪bool_clr=True)
```

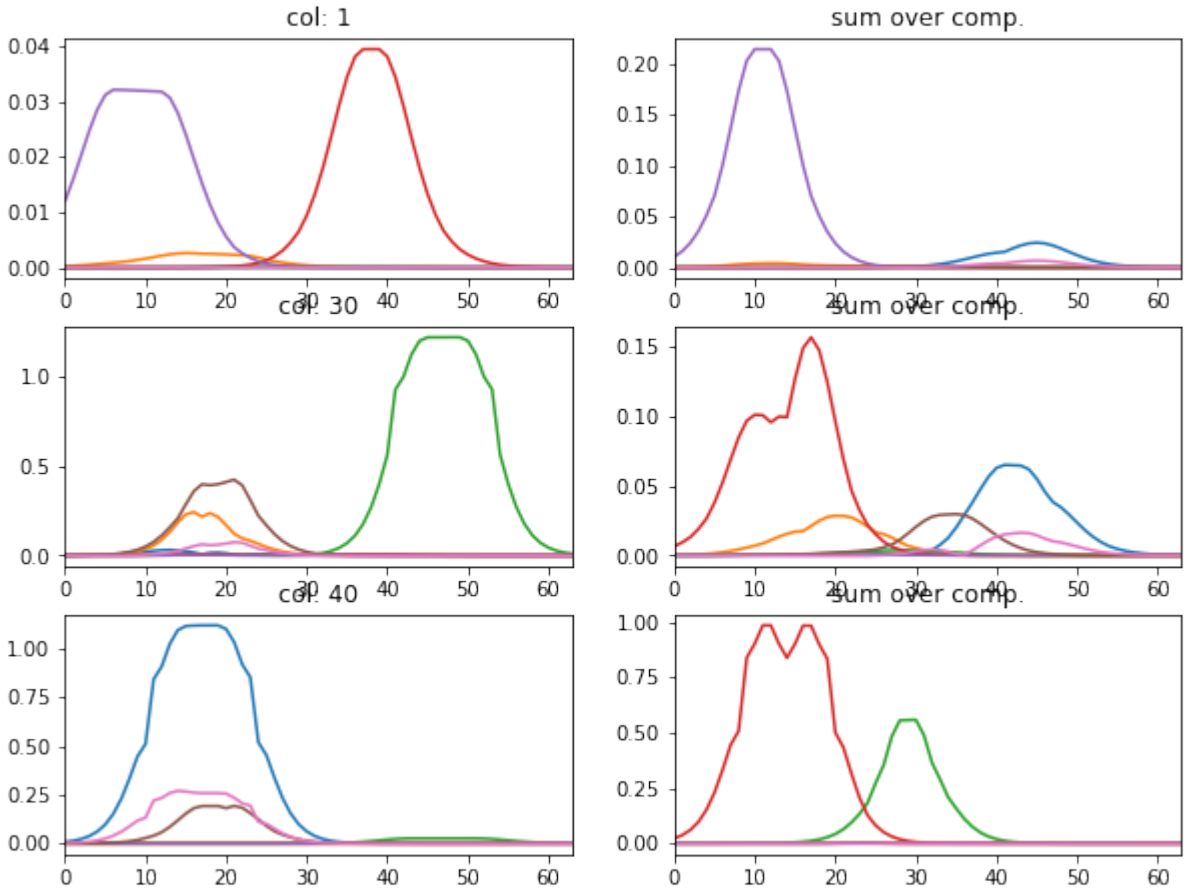
```
estimated component matrix: (7, 4096)
```



[9]:
```
ptn_used = np.sum(np.abs(X_new), axis=0) > 0
atlas_ptns = comp[ptn_used, :].reshape((-1, ) + list_imgs[0].shape)

atlas_ptns = comp.reshape((-1, ) + list_imgs[0].shape)
atlas_estim = np.argmax(atlas_ptns, axis=0) + 1
atlas_sum = np.sum(np.abs(atlas_ptns), axis=0)
atlas_estim[atlas_sum < 1e-1] = 0
_ = plt.imshow(atlas_estim, interpolation='nearest'), plt.colorbar()
```

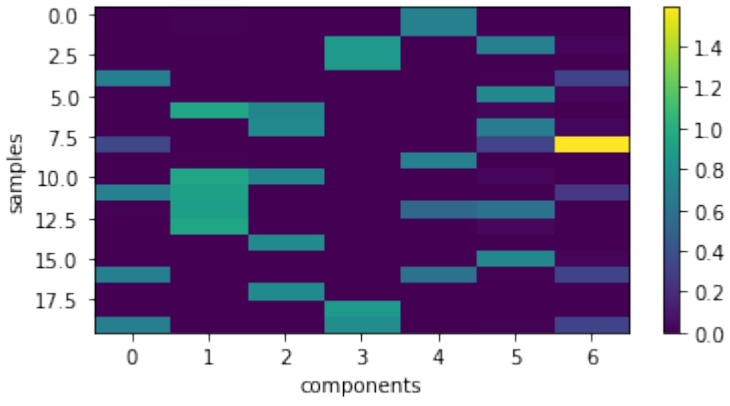


[12]:
```
l_idx = [1, 30, 40]
fig, axr = plt.subplots(len(l_idx), 2, figsize=(10, 2.5 * len(l_idx)))
for i, idx in enumerate(l_idx):
    axr[i, 0].plot(atlas_ptns[:,:,idx].T), axr[i, 0].set_xlim([0, 63])
    axr[i, 0].set_title('col: {}'.format(idx))
    axr[i, 1].plot(np.abs(atlas_ptns[:,idx,:].T)), axr[i, 1].set_xlim([0, 63])
    axr[i, 1].set_title('sum over comp.')
```



particular coding of each sample

[16]:
```
plt.figure(figsize=(6, 3))
plt.imshow(X_new[:20,:], interpolation='nearest', aspect='auto'), plt.colorbar()
_= plt.xlabel('components'), plt.ylabel('samples')

coefs = np.sum(np.abs(X_new), axis=0)
# print 'used coeficients:', coefs.tolist()
```

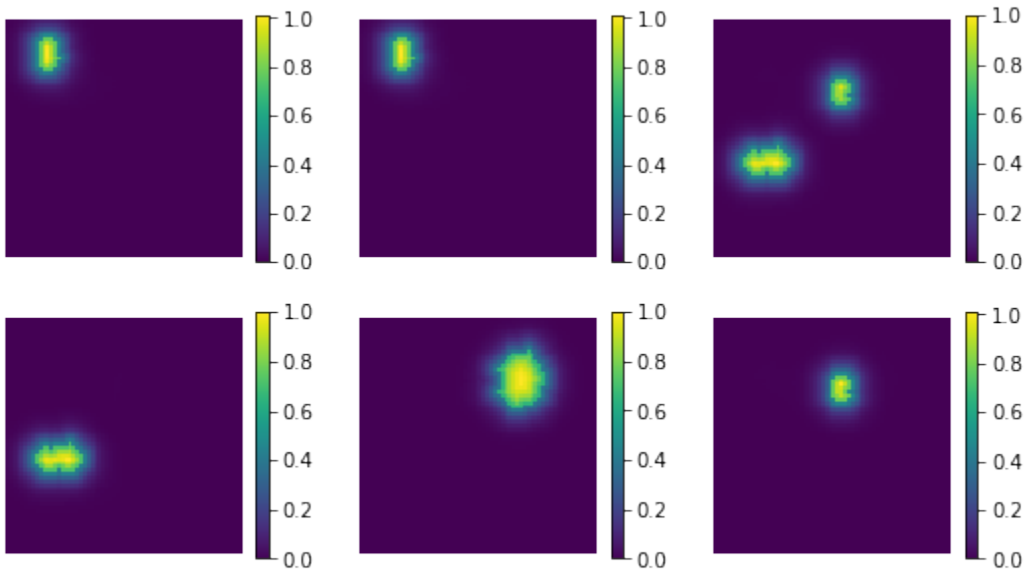


backword reconstruction from encoding and dictionary

[17]:
```
res = np.dot(X_new, comp)
print ('model applies by reverting the unmixing', res.shape)
```

```
model applies by reverting the unmixing (200, 4096)
```

[20]:
```
uts.show_sample_data_as_imgs(res, img_shape, nb_rows=2, nb_cols=3, bool_clr=True)
```



[ ]:
```
```

1.4.6 Sparse PCA

sklearn.decomposition.SparsePCA

Principal component analysis (PCA) has the disadvantage that the components extracted by this method have exclusively dense expressions, i.e. they have non-zero coefficients when expressed as linear combinations of the original variables. This can make interpretation difficult. In many cases, the real underlying components can be more naturally imagined as sparse vectors; for example in face recognition, components might naturally map to parts of faces.

Sparse principal components yields a more parsimonious, interpretable representation, clearly emphasizing which of the original features contribute to the differences between samples.

Usable examples: Faces dataset decompositions

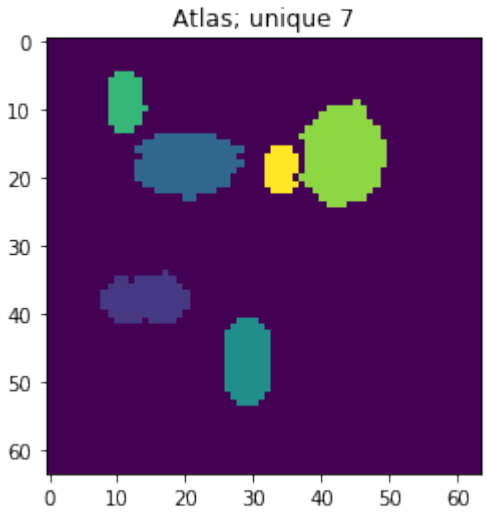
[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

```
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

load datset

[2]:
```
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

```
loading dataset: (True) exists -> /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_
↪v0/datasetFuzzy_raw
loading (True) <- /mnt/30C0201EC01FE8BC/TEMP/atomicPatternDictionary_v0/dictionary/
↪atlas.png
```



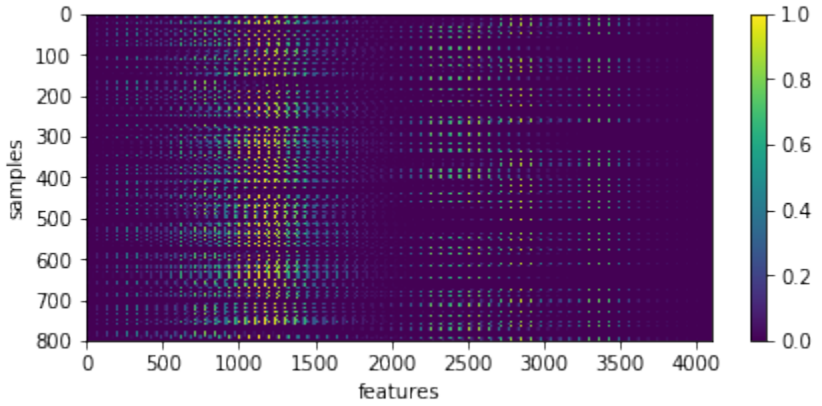
[3]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded # images: ', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)
```

```
loaded # images:  800
image shape: (64, 64)
```

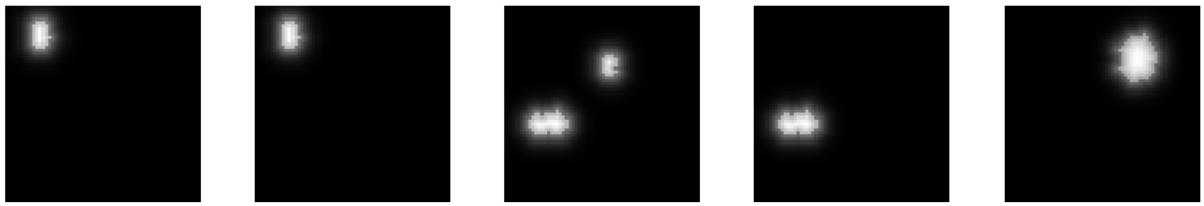
Pre-Processing

[13]:
```
X = np.array([im.ravel() for im in list_imgs])  # - 0.5
print ('input data shape:', X.shape)

plt.figure(figsize=(7, 3))
_= plt.imshow(X, aspect='auto'), plt.xlabel('features'), plt.ylabel('samples'), plt.
↪colorbar()
```

```
input data shape: (800, 4096)
```



[5]:
```
uts.show_sample_data_as_imgs(X, img_shape, nb_rows=1, nb_cols=5)
```



SparsePCA

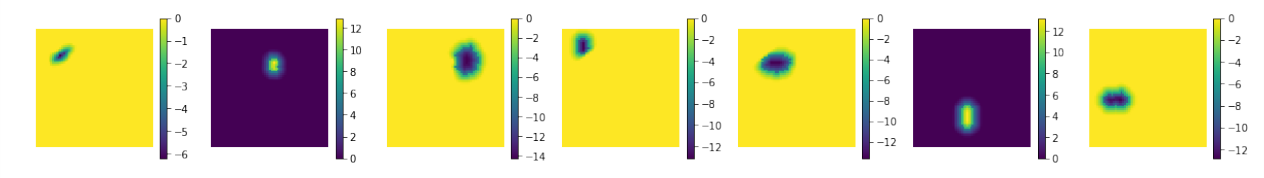
[6]:
```
from sklearn.decomposition import SparsePCA
spca = SparsePCA(n_components=nb_patterns, max_iter=999, n_jobs=-1)

X_new = spca.fit_transform(X[:1200,:])
print ('fitting parameters:', spca.get_params())
print ('number of iteration:', spca.n_iter_)
```

```
fitting parameters: {'n_jobs': -1, 'verbose': False, 'ridge_alpha': 0.01, 'max_iter':␣
↪999, 'V_init': None, 'U_init': None, 'random_state': None, 'n_components': 7, 'tol':
↪ 1e-08, 'alpha': 1, 'method': 'lars'}
number of iteration: 46
```

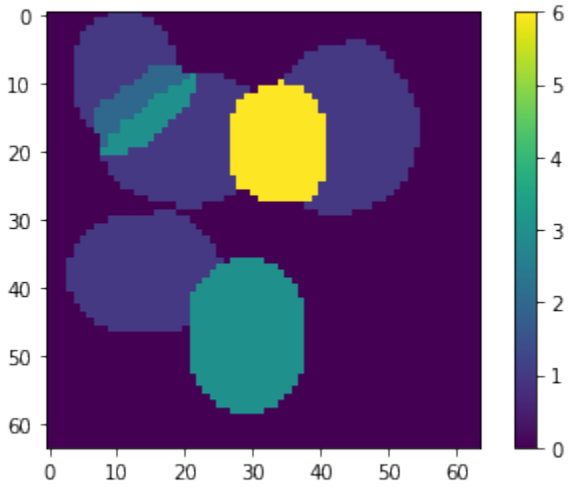
show the estimated components - dictionary

[7]:
```
comp = spca.components_
coefs = np.sum(np.abs(X_new), axis=0)
# print 'used coeficients:', coefs.tolist()
print ('estimated component matrix:', comp.shape)

compSorted = [c[0] for c in sorted(zip(comp, coefs), key=lambda x: x[1],␣
↪reverse=True) ]
uts.show_sample_data_as_imgs(np.array(compSorted), img_shape, nb_cols=nb_patterns,␣
↪bool_clr=True)
```

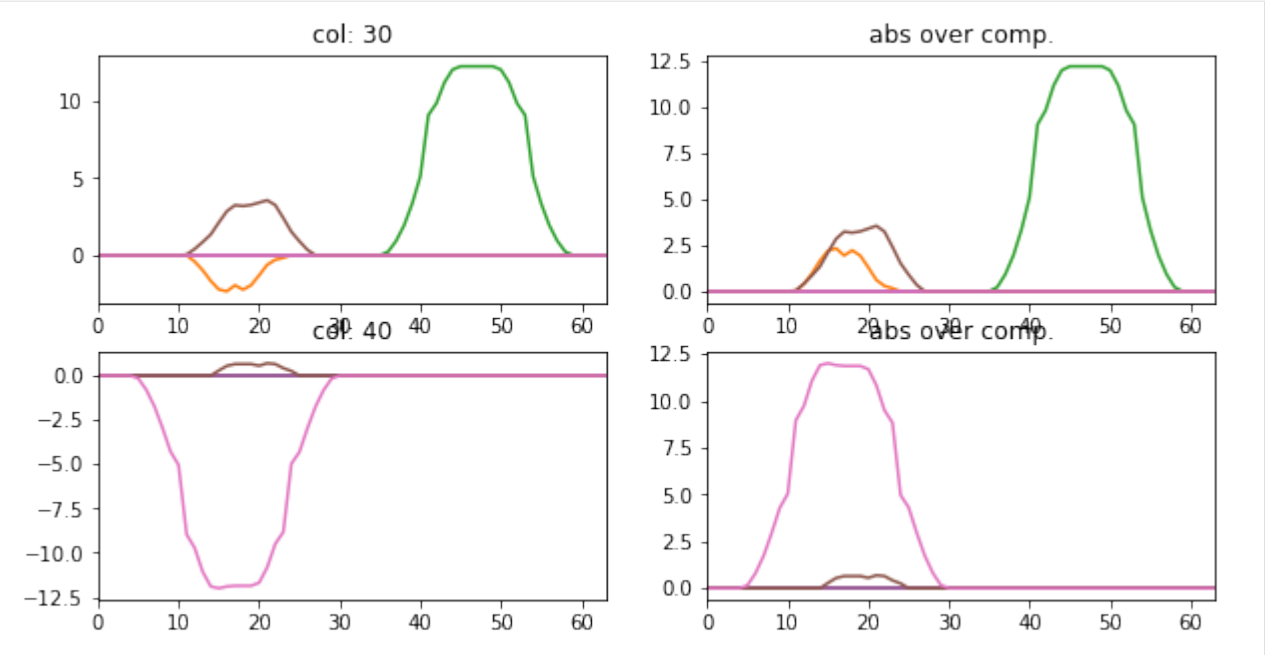
```
estimated component matrix: (7, 4096)
```



[8]:
```
ptn_used = np.sum(np.abs(X_new), axis=0) > 0
atlas_ptns = comp[ptn_used, :].reshape((-1, ) + list_imgs[0].shape)

atlas_ptns = comp.reshape((-1, ) + list_imgs[0].shape)
atlas_estim = np.argmax(atlas_ptns, axis=0) + 1
atlas_sum = np.sum(np.abs(atlas_ptns), axis=0)
atlas_estim[atlas_sum < 1e-1] = 0
_ = plt.imshow(atlas_estim, interpolation='nearest'), plt.colorbar()
```

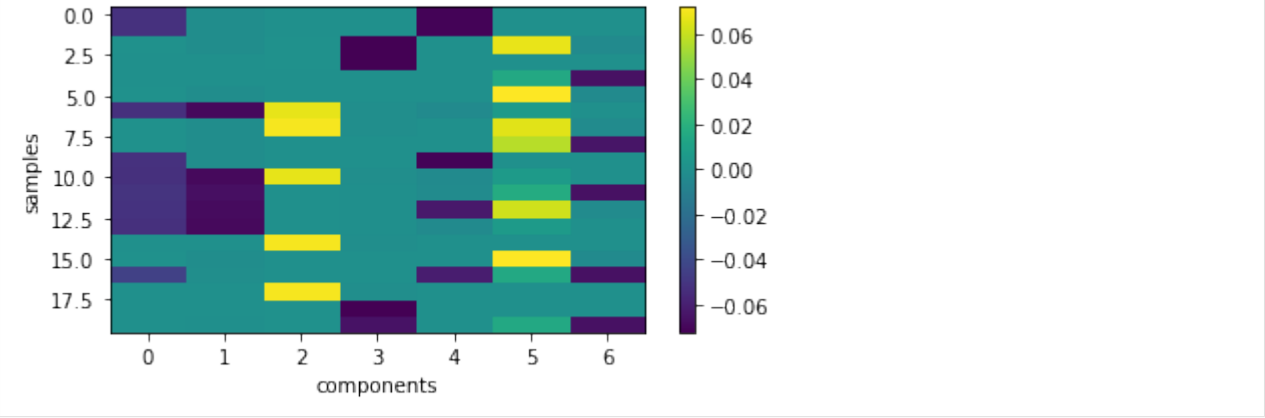


[9]:
```
l_idx = [30, 40]
fig, axr = plt.subplots(len(l_idx), 2, figsize=(10, 2.5 * len(l_idx)))
for i, idx in enumerate(l_idx):
    axr[i, 0].plot(atlas_ptns[:,:,idx].T), axr[i, 0].set_xlim([0, 63])
    axr[i, 0].set_title('col: {}'.format(idx))
    axr[i, 1].plot(np.abs(atlas_ptns[:,:,idx].T)), axr[i, 1].set_xlim([0, 63])
    axr[i, 1].set_title('abs over comp.')
```



particular coding of each sample

[10]:
```
plt.figure(figsize=(6, 3))
plt.imshow(X_new[:20,:], interpolation='nearest', aspect='auto'), plt.colorbar()
_= plt.xlabel('components'), plt.ylabel('samples')
```

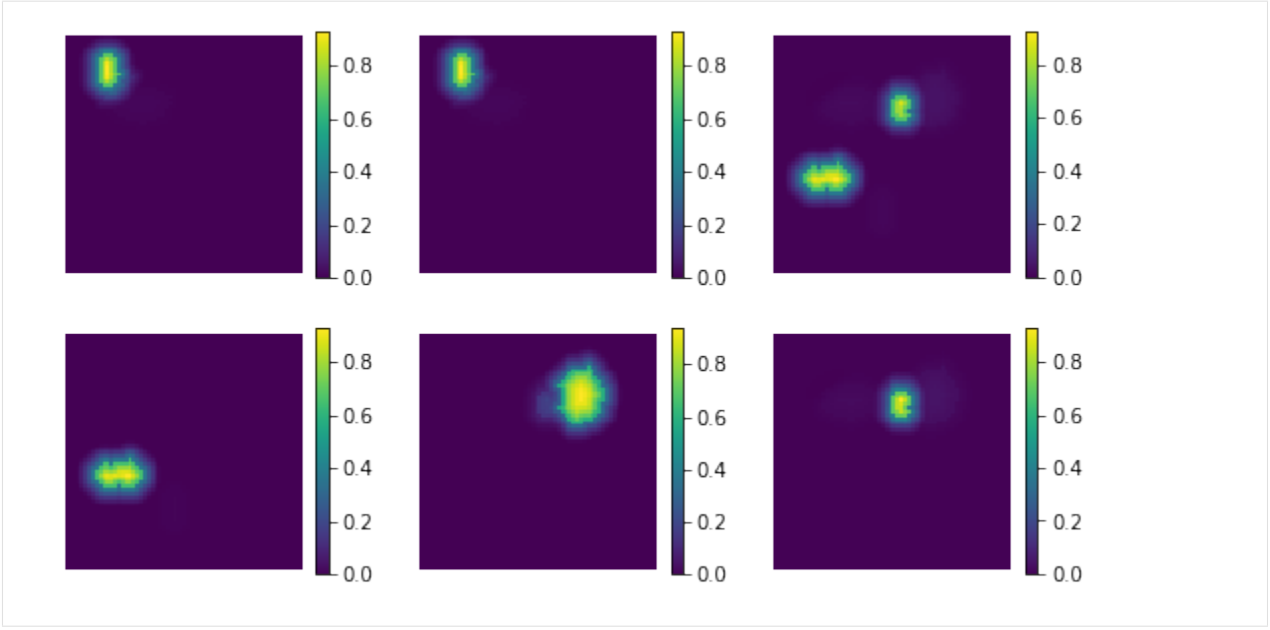


backword reconstruction from encoding and dictionary

[11]:
```
res = np.dot(X_new, comp)
print ('model applies by reverting the unmixing', res.shape)
```

```
model applies by reverting the unmixing (800, 4096)
```

[15]:
```
uts.show_sample_data_as_imgs(res, img_shape, nb_rows=2, nb_cols=3, bool_clr=True)
```



[ ]:

1.4.7 Spectral Clustering

sklearn.cluster.SpectralClustering

Spectral Clustering does a low-dimension embedding of the affinity matrix between samples, followed by a KMeans in the low dimensional space. It is especially efficient if the affinity matrix is sparse and the pyamg module is installed. SpectralClustering requires the number of clusters to be specified. It works well for a small number of clusters but is not advised when using many clusters.

For two clusters, it solves a convex relaxation of the normalised cuts problem on the similarity graph: cutting the graph in two so that the weight of the edges cut is small compared to the weights of the edges inside each cluster. This criteria is especially interesting when working on images: graph vertices are pixels, and edges of the similarity graph are a function of the gradient of the image.

Usable examples: Segmenting the picture of a raccoon face in regions

[2]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import numpy as np
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import notebooks.notebook_utils as uts
```

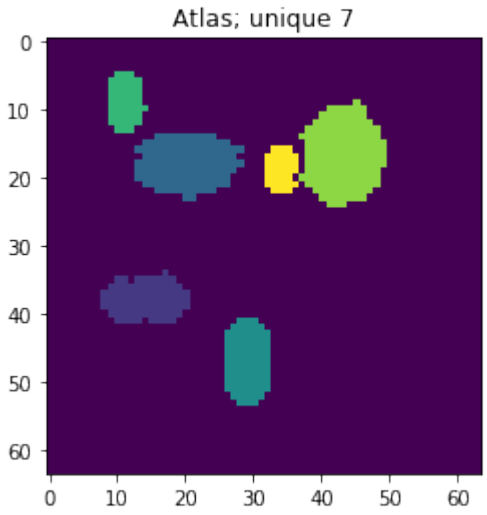
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
/mnt/datagrid/personal/borovec/Applications/vEnv2/lib/python2.7/site-packages/IPython/
↪html.py:14: ShimWarning: The `IPython.html` package has been deprecated. You should␣
↪import from `notebook` instead. `IPython.html.widgets` has moved to `ipywidgets`.
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

load datset

[7]:
```
p_dataset = os.path.join(uts.DEFAULT_PATH, uts.SYNTH_DATASETS_FUZZY[0])
print ('loading dataset: ({}) exists -> {}'.format(os.path.exists(p_dataset), p_
↪dataset))

p_atlas = os.path.join(uts.DEFAULT_PATH, 'dictionary/atlas.png')
atlas_gt = io.imread(p_atlas)
nb_patterns = len(np.unique(atlas_gt))
print ('loading ({}) <- {}'.format(os.path.exists(p_atlas), p_atlas))
plt.imshow(atlas_gt, interpolation='nearest')
_ = plt.title('Atlas; unique %i' % nb_patterns)
```

```
loading dataset: (True) exists -> /datagrid/Medical/microscopy/drosophila/synthetic_
↪data/atomicPatternDictionary_v0/datasetFuzzy_raw
loading (True) <- /datagrid/Medical/microscopy/drosophila/synthetic_data/
↪atomicPatternDictionary_v0/dictionary/atlas.png
```



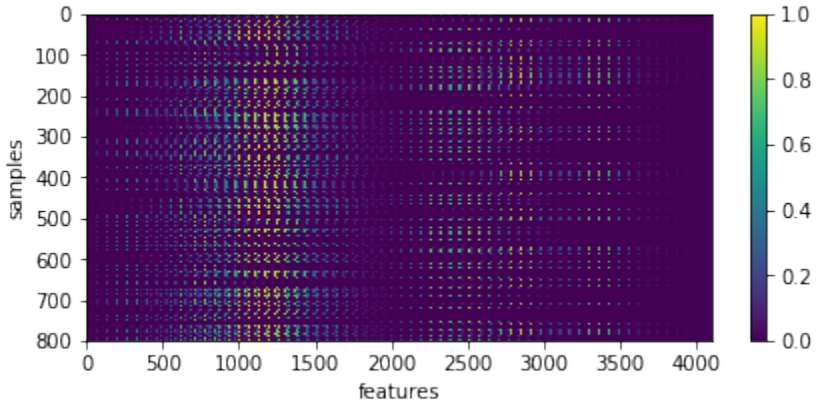
[8]:
```
list_imgs = uts.load_dataset(p_dataset)
print ('loaded # images: ', len(list_imgs))
img_shape = list_imgs[0].shape
print ('image shape:', img_shape)
```

```
('loaded # images: ', 800)
('image shape:', (64, 64))
```

Pre-Processing

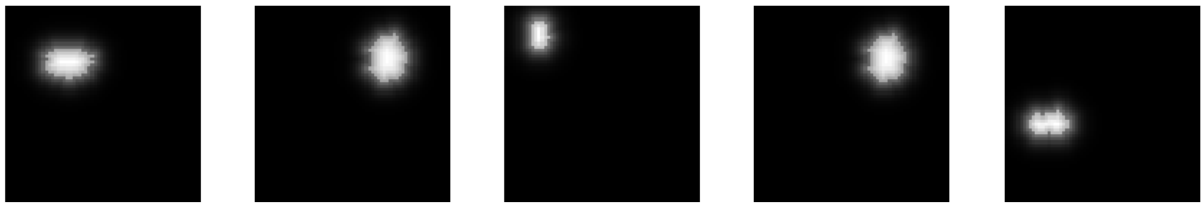
[9]:
```
X = np.array([im.ravel() for im in list_imgs])  # - 0.5
print ('input data shape:', X.shape)

plt.figure(figsize=(7, 3))
_= plt.imshow(X, aspect='auto'), plt.xlabel('features'), plt.ylabel('samples'), plt.
↪colorbar()
```

```
('input data shape:', (800, 4096))
```



[10]:
```
uts.show_sample_data_as_imgs(X, img_shape, nb_rows=1, nb_cols=5)
```

```
<matplotlib.figure.Figure at 0x7fb2827ee7d0>
```



SparsePCA

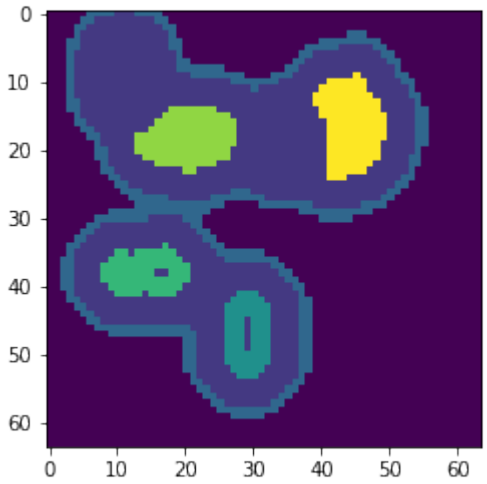
[18]:
```
from sklearn.cluster import SpectralClustering
sc = SpectralClustering(n_clusters=nb_patterns, n_jobs=-1) # , assign_labels=
↪'discretize'

sc.fit(X[:1200, :].T)
atlas = sc.labels_.reshape(atlas_gt.shape)
```

[18]:
```
SpectralClustering(affinity='rbf', assign_labels='kmeans', coef0=1, degree=3,
          eigen_solver=None, eigen_tol=0.0, gamma=1.0, kernel_params=None,
          n_clusters=7, n_init=10, n_jobs=-1, n_neighbors=10,
          random_state=None)
```

[19]:
```
plt.imshow(atlas)
```

```
('labels:', (4096,))
```

[19]:
```
<matplotlib.image.AxesImage at 0x7fb275e30a90>
```



[ ]:

1.4.8 Synthetic datasets

This brief presentation shows a few sample images from generated synthetic data focusing on Atomic Pattern Dictionary and sprse encoding

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys, glob
import pandas
import numpy as np
from skimage import io
import matplotlib.pylab as plt
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import bpdl.data_utils as tl_utils
```

[3]:
```
DEFAULT_PATH = '/mnt/30C0201EC01FE8BC/TEMP'
DEFAULT_DATASET = 'atomicPatternDictionary_v0'
DEFAULT_IMG_POSIX = '.png'
pathDataset = os.path.join(DEFAULT_PATH, DEFAULT_DATASET)
print ([os.path.basename(p) for p in glob.glob(os.path.join(pathDataset, '*'))])
```

```
['combination.csv', 'datasetBinary_defNoise', 'datasetBinary_deform', 'datasetBinary_
↪noise', 'datasetBinary_raw', 'datasetFuzzy_defNoise', 'datasetFuzzy_deform',
↪'datasetFuzzy_noise', 'datasetFuzzy_raw', 'datasetFuzzy_raw_gauss-0.001',
↪'datasetFuzzy_raw_gauss-0.005', 'datasetFuzzy_raw_gauss-0.010', 'datasetFuzzy_raw_
↪gauss-0.025', 'datasetFuzzy_raw_gauss-0.050', 'datasetFuzzy_raw_gauss-0.075',
↪'datasetFuzzy_raw_gauss-0.100', 'datasetFuzzy_raw_gauss-0.125', 'datasetFuzzy_raw_
↪gauss-0.150', 'datasetFuzzy_raw_gauss-0.200', 'dictionary']
```

[4]:
```
def showDatasetImages(pathBase, dataset, imPattern='*', nbSamples=None, perRow=5):
    imgs, names = tl_utils.dataset_load_images(os.path.join(pathBase, dataset),␣
↪imPattern, nbSamples)
    nbRows = int(np.ceil(float(len(imgs)) / perRow))
    plt.figure(figsize=(10, nbRows * perRow))
```

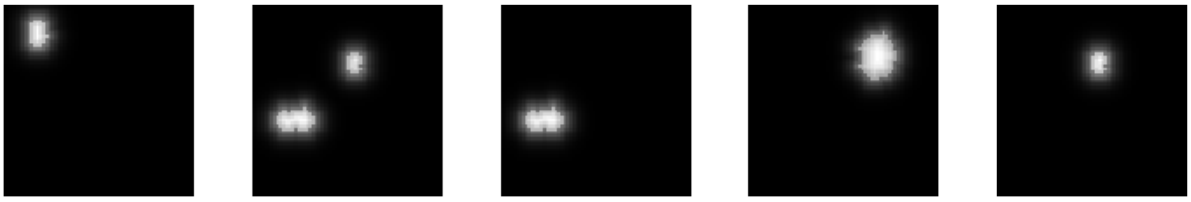
(continues on next page)

(continued from previous page)

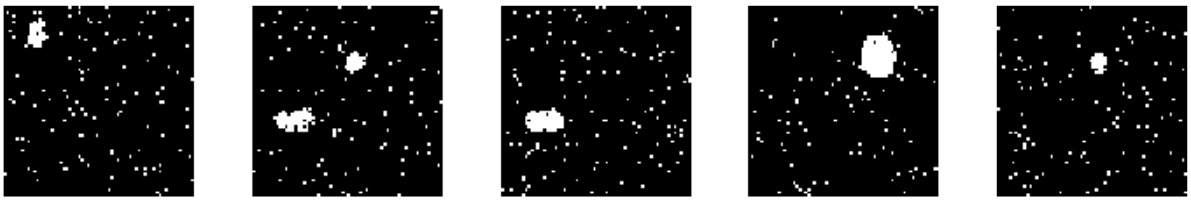
```
    for i, img in enumerate(imgs):
        plt.subplot(nbRows, perRow, i + 1), plt.imshow(img, cmap=plt.cm.gray), plt.
↪axis('off')
    plt.tight_layout()
```

[5]:
```
showDatasetImages(pathDataset, 'datasetBinary_raw', nbSamples=5)
```

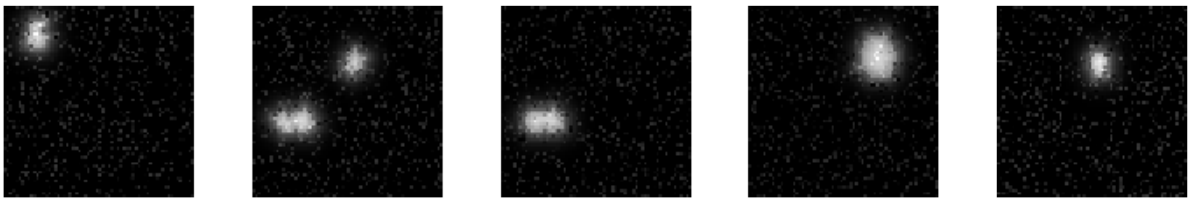
[6]:
```
showDatasetImages(pathDataset, 'datasetFuzzy_raw', nbSamples=5)
```



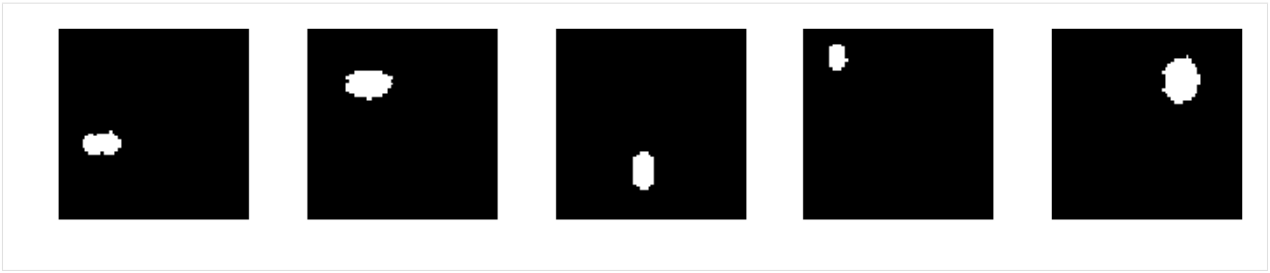
[7]:
```
showDatasetImages(pathDataset, 'datasetBinary_defNoise', nbSamples=5)
```



[9]:
```
showDatasetImages(pathDataset, 'datasetFuzzy_defNoise', nbSamples=5)
```



[10]:
```
showDatasetImages(pathDataset, 'dictionary', imPattern='pattern_*', nbSamples=5)
```



[12]:
```
pCoding = os.path.join(pathDataset, tl_utils.CSV_NAME_WEIGHTS) # 'combination.csv'
df = pandas.read_csv(pCoding, index_col=0)
print (df.head())
```

```
   combination          name
0  0;0;0;1;0;0  sample_00000
1  1;0;0;0;0;1  sample_00001
2  1;0;0;0;0;0  sample_00002
3  0;0;0;0;1;0  sample_00003
4  0;0;0;0;0;1  sample_00004
```

```
/usr/local/lib/python3.5/dist-packages/IPython/kernel/__main__.py:2: FutureWarning:␣
↪from_csv is deprecated. Please use read_csv(...) instead. Note that some of the␣
↪default arguments are different, so please refer to the documentation for from_csv␣
↪when changing your function calls
  from IPython.kernel.zmq import kernelapp as app
```

[ ]:
```
```

1.4.9 Training sigmoid funstion for gene activations

[1]:
```
% matplotlib inline
import os, glob
import numpy as np
import matplotlib.pylab as plt
```

[2]:
```
PATH_IMAGES = '/datagrid/Medical/microscopy/drosophila/TEMPORARY/annot-user-labels-
↪train/'
PATH_ACTIVE_IMGS = os.path.join(PATH_IMAGES, 'positive/*.png')
PATH_PASIVE_IMGS = os.path.join(PATH_IMAGES, 'negative/*.png')
PATH_SEGMS = PATH_IMAGES_SEGS = '/datagrid/Medical/microscopy/drosophila/RESULTS/
↪PIPELINE_ovary_all_images/1_init_tissue_segmentation'
lp_active = glob.glob(PATH_ACTIVE_IMGS)
lp_pasive = glob.glob(PATH_PASIVE_IMGS)
print('numbers of active: %i and pasive: %i example' % (len(lp_active), len(lp_
↪pasive)))
```

```
numbers of active: 185 and pasive: 47 example
```

Compute histograms

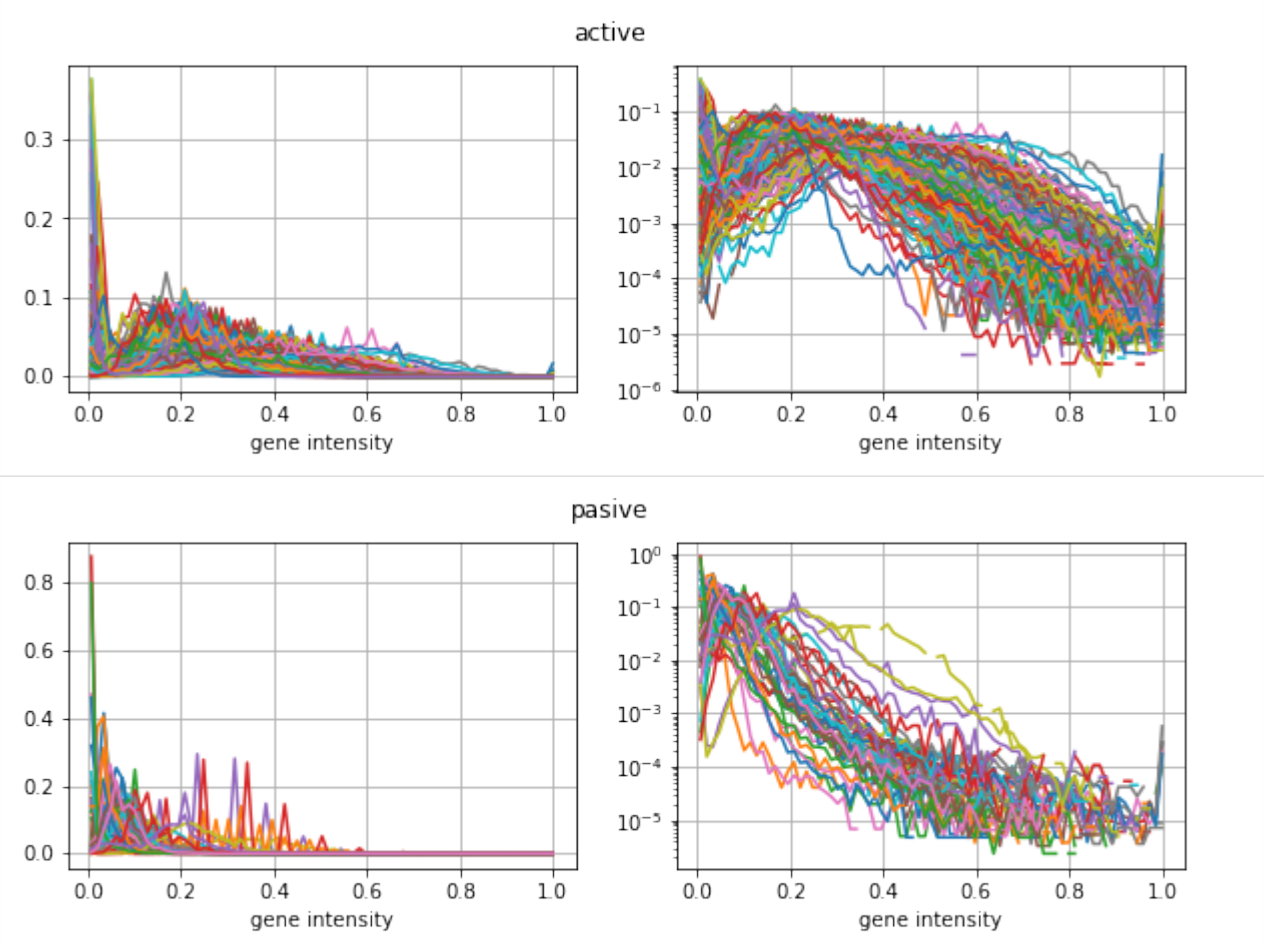
[3]:
```
def extract_images(lp_images, path_segms=None):
    imgs_px = []
    for p_img in lp_images:
        im = plt.imread(p_img)[:, :, 1]
        if path_segms is None:
            seg = np.ones(im.shape)
        else:
            p_seg = os.path.join(path_segms, os.path.splitext(os.path.basename(p_
↪img))[0] + '.png')
            if not os.path.isfile(p_seg):
                continue
            seg = plt.imread(p_seg) > 0
        imgs_px.append(im.ravel()[seg.ravel() > 0])
    return imgs_px
```

[4]:
```
def compute_histogram(imgs_px):
    hists = []
    for ip in imgs_px:
        hg, b = np.histogram(ip, bins=75)
        hg = (hg / float(np.sum(hg)))
        hists.append(hg)
    bins = (b[1:] + b[:-1]) / 2.
    return hists, bins
```

[5]:
```
def show_histogram(name, hists, bins):
    if len(hists) == 0:
        return
    plt.figure(figsize=(10, 3))
    plt.subplot(1, 2, 1), plt.plot(bins / np.max(bins), np.array(hists).T)
    plt.grid(), plt.xlabel('gene intensity'), plt.suptitle(name)
    plt.subplot(1, 2, 2), plt.semilogy(bins / np.max(bins), np.array(hists).T)
    plt.grid(), plt.xlabel('gene intensity'), plt.suptitle(name)
    # plt.xticks(range(75)[::5], [str(round(b * 255)) for b in bins[::5]])
```

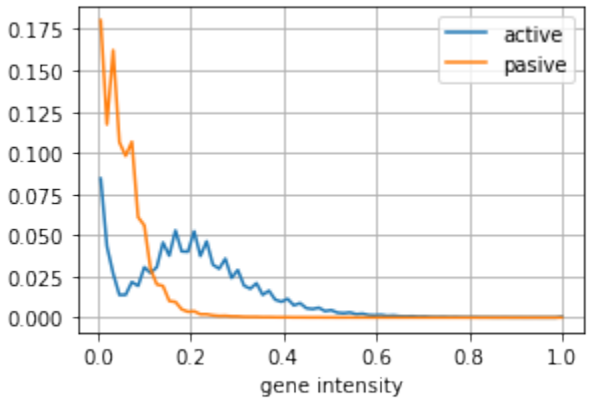
histograms over all images with filtered interior

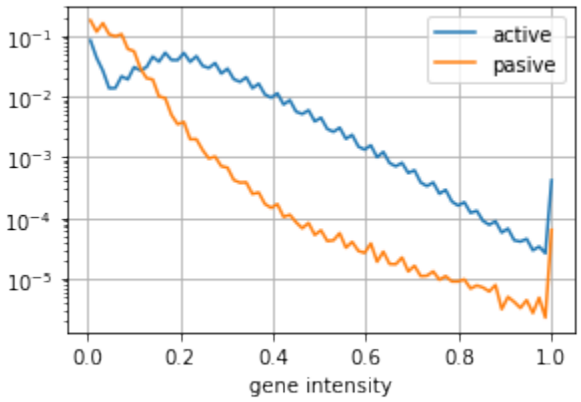
[6]:
```
for name, lp_images in [('active', lp_active), ('pasive', lp_pasive)]:
    imgs_px = extract_images(lp_images, path_segms=PATH_SEGMS)
    hists, bins = compute_histogram(imgs_px)
    show_histogram(name, hists, bins)
```



grouped histogram

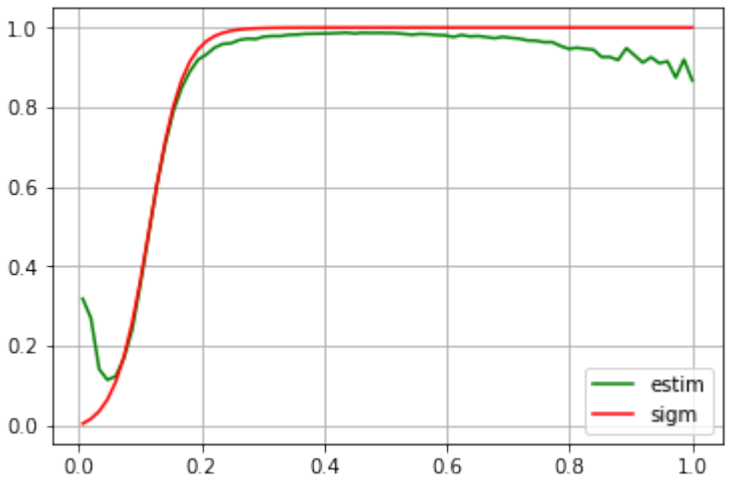
[7]:
```
d_hist = {}
for name, lp_images in [('active', lp_active), ('pasive', lp_pasive)]:
    imgs_px = extract_images(lp_images, path_segms=PATH_SEGMS)
    imgs_px = [np.hstack(imgs_px)]
    hists, bins = compute_histogram(imgs_px)
    d_hist[name] = hists[0]

bins_norm = bins / np.max(bins)
plt.figure(figsize=(10, 3))
plt.subplot(1, 2, 1)
for k in d_hist:
    plt.plot(bins_norm, d_hist[k], label=k)
_= plt.grid(), plt.xlabel('gene intensity'), plt.legend()
plt.subplot(1, 2, 2)
for k in d_hist:
    plt.semilogy(bins_norm, d_hist[k], label=k)
_= plt.grid(), plt.xlabel('gene intensity'), plt.legend()
```



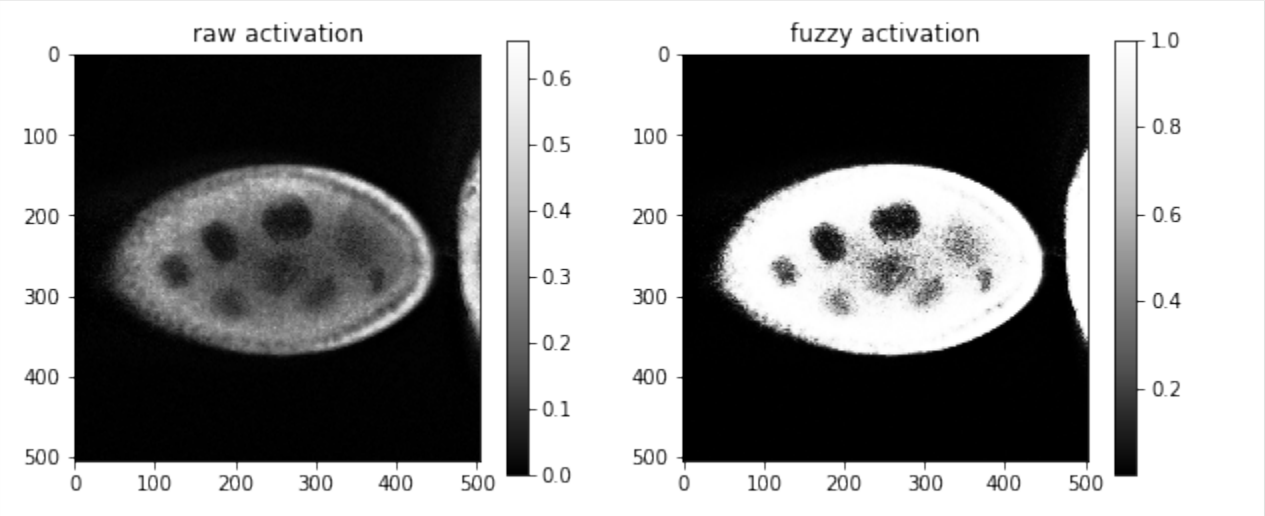


Fuzzy activations

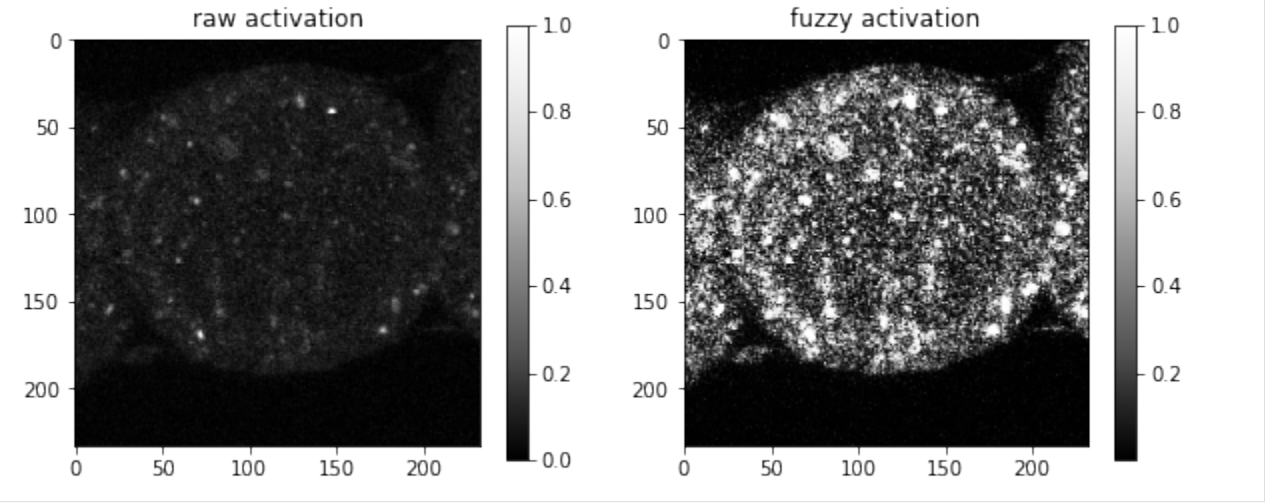
[30]:
```
dec = d_hist['active'] / (d_hist['active'] + d_hist['pasive'])

def activate(x, shift=0.12, slope=35.):
    sigm = lambda x, a, b: 1. / (1 + np.exp(b * (- x + a)))
    sigm_0, sigm_inf = sigm(0, shift, slope), sigm(1e3, shift, slope)
    val = (sigm(x, shift, slope) - sigm_0) / (sigm_inf - sigm_0)
    return val

plt.plot(bins_norm, dec, 'g', label='estim')
plt.plot(bins_norm, activate(bins_norm, 0.115, 36.), 'r', label='sigm')
_= plt.grid(), plt.legend()
```



[33]:
```
img = plt.imread(lp_active[np.random.randint(0, len(lp_active))])[:, :, 1]
img_fuzzy = activate(img, 0.12, 35.)
plt.figure(figsize=(10, 4))
_= plt.subplot(1, 2, 1), plt.title('raw activation'), plt.imshow(img, cmap=plt.cm.
↪Greys_r), plt.colorbar()
_= plt.subplot(1, 2, 2), plt.title('fuzzy activation'), plt.imshow(img_fuzzy,␣
↪cmap=plt.cm.Greys_r), plt.colorbar()
```



[35]:
```
img = plt.imread(lp_pasive[np.random.randint(0, len(lp_pasive))])[:, :, 1]
img_fuzzy = activate(img, 0.12, 35.)
plt.figure(figsize=(10, 4))
_= plt.subplot(1, 2, 1), plt.title('raw activation'), plt.imshow(img, cmap=plt.cm.
↪Greys_r), plt.colorbar()
_= plt.subplot(1, 2, 2), plt.title('fuzzy activation'), plt.imshow(img_fuzzy,␣
↪cmap=plt.cm.Greys_r), plt.colorbar()
```



[ ]:

1.4.10 Sample registration of an image to patterns - Deamons

Since thre is still an deformation among input images and its representation we look for a descriptive transformations...

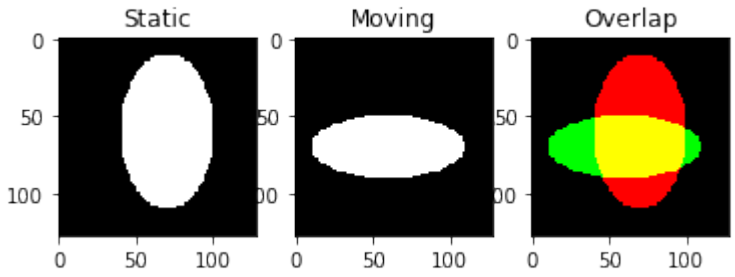
[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys
import time
import numpy as np
import matplotlib.pylab as plt

sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
from bpdl import data_utils as tl_data
import notebooks.notebook_utils as nb_utils
```

```
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

Input images

[2]:
```
# STATIC IMAGE - from atlas
img_static, img_moving = nb_utils.generate_synth_image_pair_simple()
img_static_fuzzy = tl_data.image_transform_binary2fuzzy(img_static, coef=0.2)
img_moving_fuzzy = tl_data.image_transform_binary2fuzzy(img_moving, coef=0.2)
```

[3]:
```
plt.subplot(1, 3, 1), plt.imshow(img_static, cmap=plt.cm.Greys_r), plt.title('Static')
plt.subplot(1, 3, 2), plt.imshow(img_moving, cmap=plt.cm.Greys_r), plt.title('Moving')
im_overlap = np.rollaxis(np.array([img_static, img_moving, np.zeros(img_static.
↪shape)]), 0, 3)
_= plt.subplot(1, 3, 3), plt.imshow(im_overlap), plt.title('Overlap')
```



Symmetric Diffeomorphic Registration

This example explains how to register 2D images using the Symmetric Normalization (SyN) algorithm proposed by Avants et al. Avants09 (also implemented in the ANTS software Avants11)

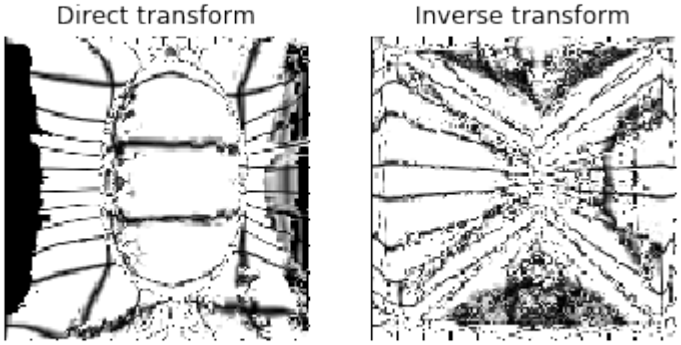
[4]:
```
# from dipy.data import get_data
from dipy.align.imwarp import SymmetricDiffeomorphicRegistration
from dipy.align.metrics import SSDMetric #, CCMetric, EMMetric
# import dipy.align.imwarp as imwarp
from dipy.viz import regtools

# regtools.overlay_images(img_static, img_moving, 'Static', 'Overlay', 'Moving', '')
```

```
/mnt/datagrid/personal/borovec/Applications/vEnv2/lib/python2.7/site-packages/h5py/__
↪init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from␣
↪`float` to `np.floating` is deprecated. In future, it will be treated as `np.
↪float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

[5]:
```
from dipy.align.imwarp import RegistrationStages as sStages

def print_deform(obj, stage):
    if stage not in [sStages.INIT_END, sStages.ITER_END, sStages.OPT_END, sStages.
↪ITER_END]:
        return
    nb_pxls = np.product(obj.moving_to_ref.forward.shape)
    deform_f = obj.moving_to_ref.forward
    deform_b = obj.static_to_ref.forward
    print('stage:', stage, '...',
          'deform B:', np.sum(abs(deform_b)) / nb_pxls,
          'deform F:', np.sum(abs(deform_f)) / nb_pxls)
```

[6]:
```
def register_dipy(img_static, img_moving, callback=None):
    sdr = SymmetricDiffeomorphicRegistration(metric=SSDMetric(img_static.ndim),
                                             step_length=1.,
                                             level_iters=[50, 100],
                                             inv_iter=50,
                                             ss_sigma_factor=1.,
                                             opt_tol=1.e-3,
                                             callback=callback)

    t = time.time()
    mapping = sdr.optimize(img_static.astype(float), img_moving.astype(float))
    print ('optimize took:', time.time() - t)
    return mapping

mapping = register_dipy(img_static, img_static_fuzzy)
_= regtools.plot_2d_diffeomorphic_map(mapping, 10)
```

```
Creating scale space from the moving image. Levels: 2. Sigma factor: 1.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 1.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 2.193876028060913)
```

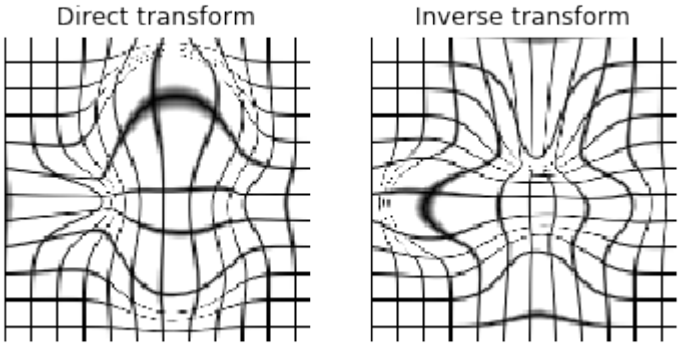


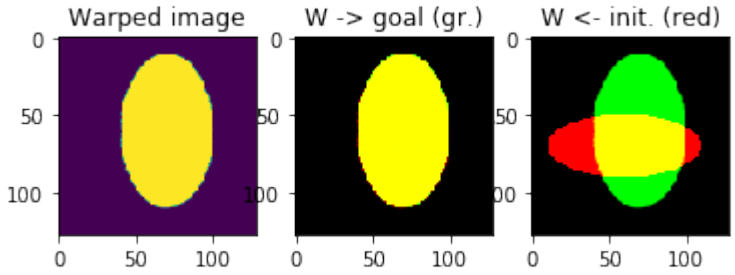
Experiments

Register: **static <- moving** image

[7]:
```
mapping = register_dipy(img_static, img_moving, None)
_= regtools.plot_2d_diffeomorphic_map(mapping, 10)
img_warped = mapping.transform(img_moving, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_moving)
```

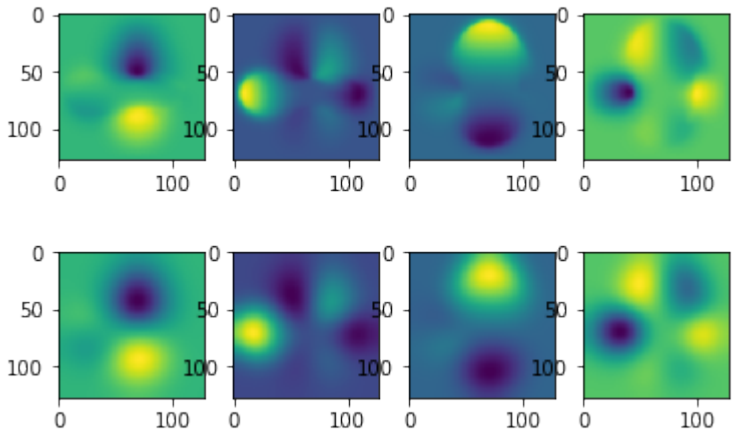
```
Creating scale space from the moving image. Levels: 2. Sigma factor: 1.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 1.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 0.6593570709228516)
```





[8]:
```
from scipy.ndimage.filters import gaussian_filter

plt.subplot(2, 4, 1), plt.imshow(mapping.forward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 2), plt.imshow(mapping.forward[..., 1], interpolation='nearest')
plt.subplot(2, 4, 3), plt.imshow(mapping.backward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 4), plt.imshow(mapping.backward[..., 1], interpolation='nearest')

m_sigma = [10, 10, 0]
mapping.forward = gaussian_filter(mapping.forward, sigma=m_sigma)
mapping.backward = gaussian_filter(mapping.backward, sigma=m_sigma)

plt.subplot(2, 4, 5), plt.imshow(mapping.forward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 6), plt.imshow(mapping.forward[..., 1], interpolation='nearest')
plt.subplot(2, 4, 7), plt.imshow(mapping.backward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 8), plt.imshow(mapping.backward[..., 1], interpolation='nearest')
```

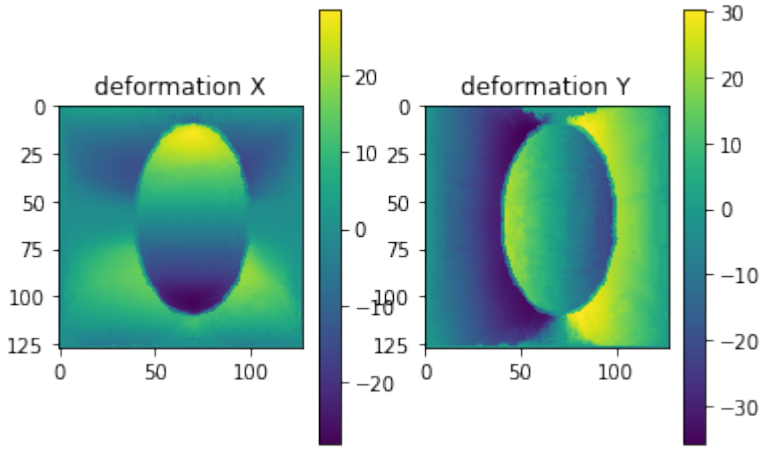
[8]:
```
(<matplotlib.axes._subplots.AxesSubplot at 0x7f451b9cd350>,
 <matplotlib.image.AxesImage at 0x7f451b9468d0>)
```

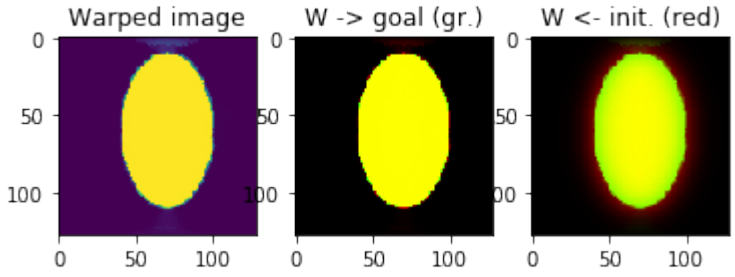


Register: **static <- static FUZZY** image

[9]:
```
mapping = register_dipy(img_static.astype(float), img_static_fuzzy, None)
# _= regtools.plot_2d_diffeomorphic_map(mapping, 10)
nb_utils.show_registered_deformation(mapping.backward)
img_warped = mapping.transform(img_static_fuzzy, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_static_fuzzy)
```

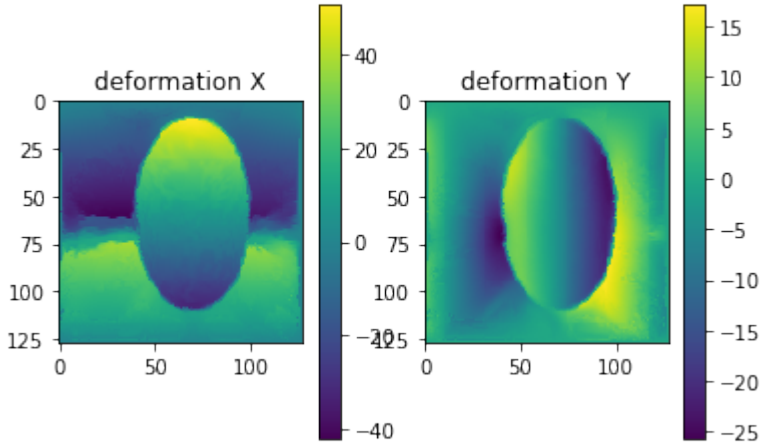
```
Creating scale space from the moving image. Levels: 2. Sigma factor: 1.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 1.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 2.107191801071167)
```

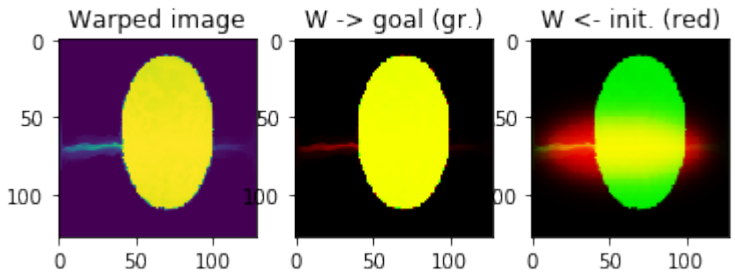




[10]:
```
mapping = register_dipy(img_static, img_moving_fuzzy, None)
# _= regtools.plot_2d_diffeomorphic_map(mapping, 10)
nb_utils.show_registered_deformation(mapping.backward)
img_warped = mapping.transform(img_moving_fuzzy, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_moving_fuzzy)
```

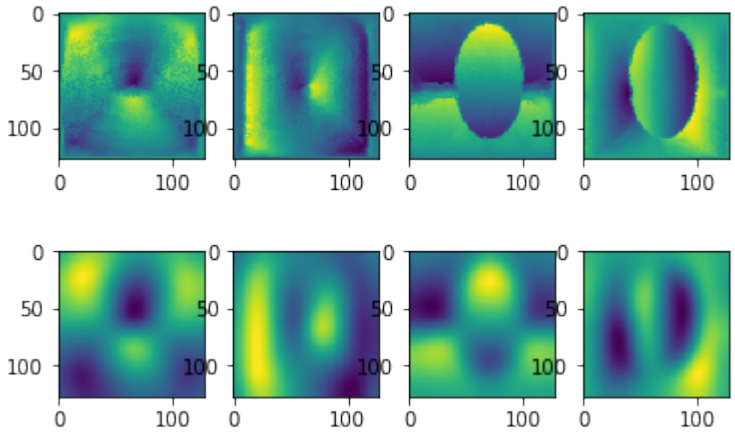
```
Creating scale space from the moving image. Levels: 2. Sigma factor: 1.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 1.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 2.1672451496124268)
```



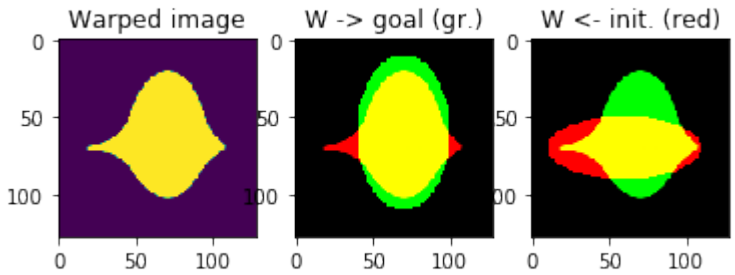


[11]:
```
from scipy.ndimage.filters import gaussian_filter

plt.subplot(2, 4, 1), plt.imshow(mapping.forward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 2), plt.imshow(mapping.forward[..., 1], interpolation='nearest')
plt.subplot(2, 4, 3), plt.imshow(mapping.backward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 4), plt.imshow(mapping.backward[..., 1], interpolation='nearest')

m_sigma = [10, 10, 0]
mapping.forward = gaussian_filter(mapping.forward, sigma=m_sigma)
mapping.backward = gaussian_filter(mapping.backward, sigma=m_sigma)

plt.subplot(2, 4, 5), plt.imshow(mapping.forward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 6), plt.imshow(mapping.forward[..., 1], interpolation='nearest')
plt.subplot(2, 4, 7), plt.imshow(mapping.backward[..., 0], interpolation='nearest')
plt.subplot(2, 4, 8), plt.imshow(mapping.backward[..., 1], interpolation='nearest')
```

[11]:
```
(<matplotlib.axes._subplots.AxesSubplot at 0x7f451b045510>,
 <matplotlib.image.AxesImage at 0x7f451af3ca90>)
```



[12]:
```
img_warped = mapping.transform(img_moving, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_moving)
```



Smooth Symmetric Diffeomorphic Registration

[13]:
```
from bpdl.registration import SmoothSymmetricDiffeomorphicRegistration
from dipy.align.imwarp import DiffeomorphicMap

def register_dipy_smooth(img_static, img_moving, smooth_sigma):
    sdr = SmoothSymmetricDiffeomorphicRegistration(metric=SSDMetric(img_static.ndim),
                                                   smooth_sigma=smooth_sigma,
                                                   step_length=1.,
                                                   level_iters=[5, 10],
                                                   inv_iter=5,
                                                   ss_sigma_factor=10.,
                                                   opt_tol=1.e-3)

    t = time.time()
    mapping = sdr.optimize(img_static.astype(float), img_moving.astype(float))
    print ('optimize took:', time.time() - t)

    # mapping_inv = sdr.moving_to_ref
    mapping_inv = DiffeomorphicMap(img_static.ndim,
                                   img_static.shape, None,
                                   img_static.shape, None,
                                   img_static.shape, None,
                                   None)
    mapping_inv.forward = np.array(sdr.moving_to_ref.forward)
```

(continues on next page)

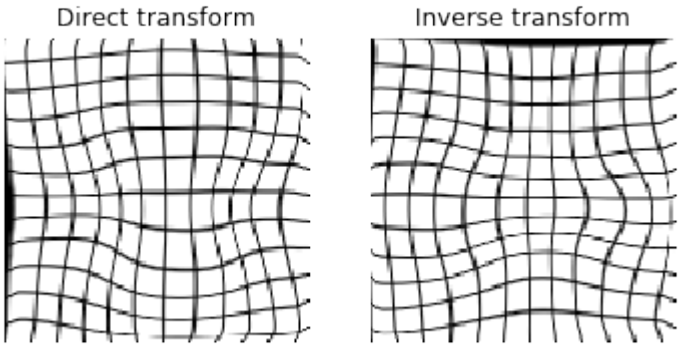
(continued from previous page)

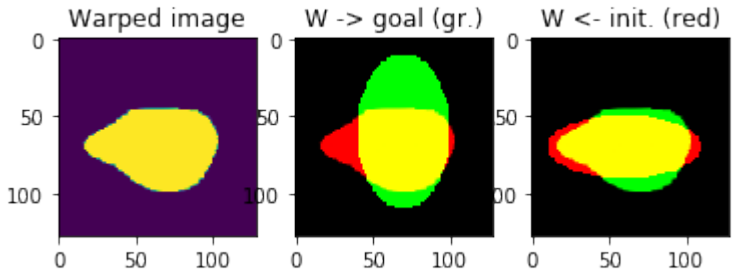
```
    mapping_inv.backward = np.array(sdr.moving_to_ref.backward)
    return mapping, mapping_inv
```

Register: **static <- moving** image

[14]:
```
mapping, mapping_inv = register_dipy_smooth(img_static, img_moving, 10.)
_= regtools.plot_2d_diffeomorphic_map(mapping, 10)
img_warped = mapping.transform(img_moving, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_moving)
```

```
Creating scale space from the moving image. Levels: 2. Sigma factor: 10.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 10.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 0.34917306900024414)
```

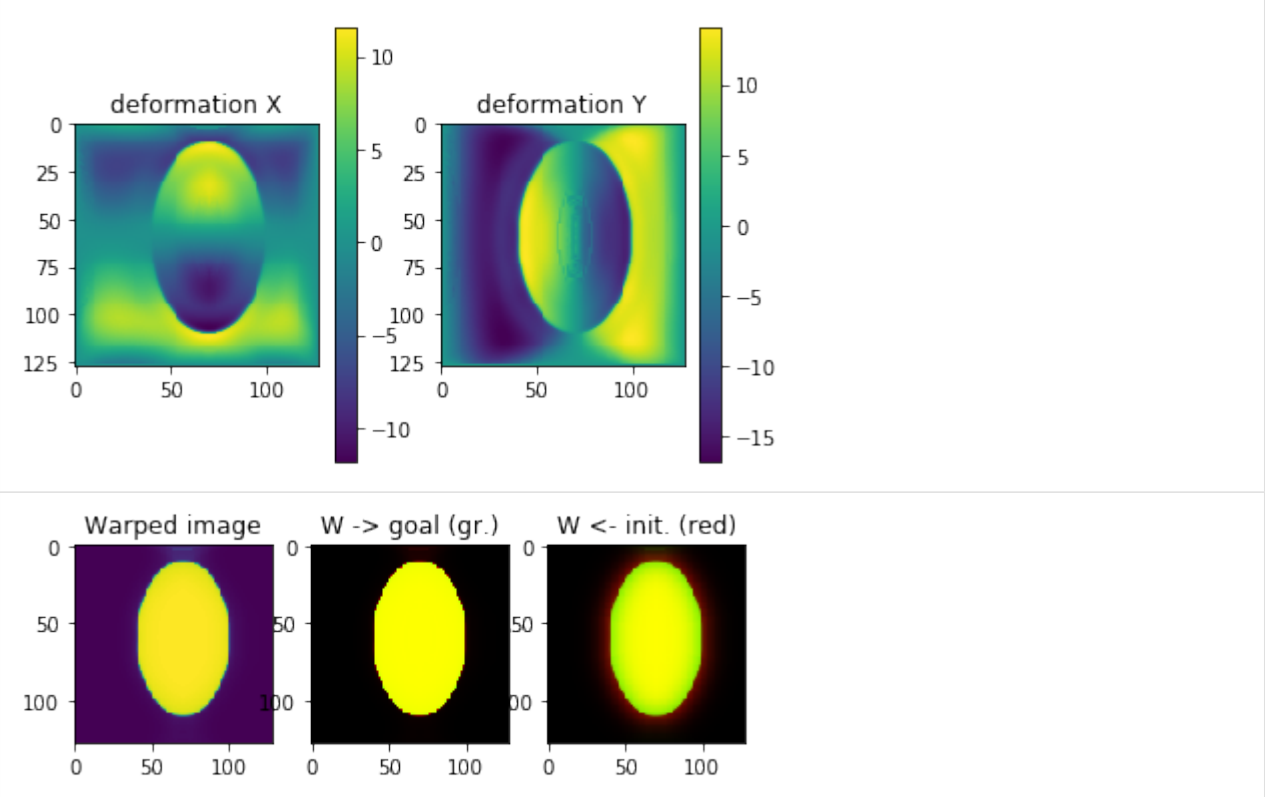




Register: **static <- static FUZZY** image

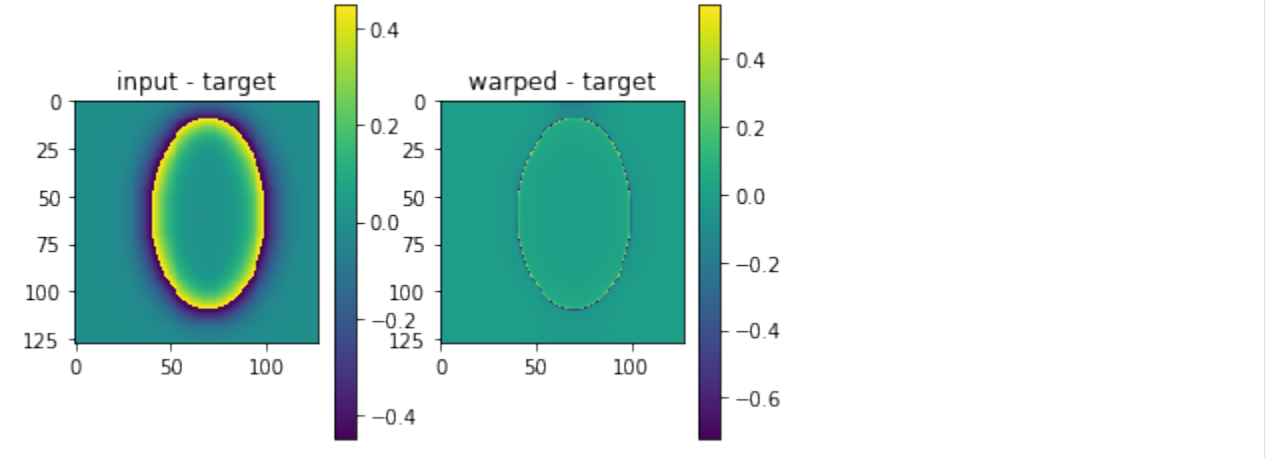
[15]:
```
mapping, mapping_inv = register_dipy_smooth(img_static, img_static_fuzzy, 0.)

# _= regtools.plot_2d_diffeomorphic_map(mapping, 10)
nb_utils.show_registered_deformation(mapping.backward)
img_warped = mapping.transform(img_static_fuzzy, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_static_fuzzy)
```

```
Creating scale space from the moving image. Levels: 2. Sigma factor: 10.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 10.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 0.3360099792480469)
```



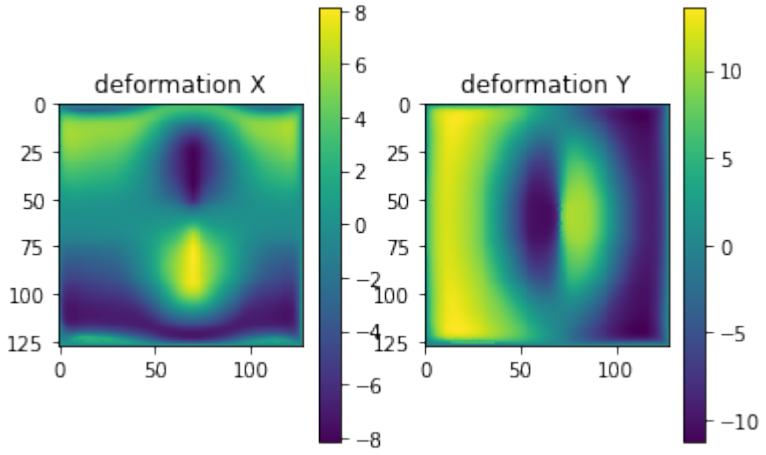
[16]:
```
_= plt.subplot(121), plt.imshow(img_static - img_static_fuzzy), plt.title('input -␣
↪target'), plt.colorbar()
print('SSD input-target:', np.sum((img_static - img_static_fuzzy) ** 2))
_= plt.subplot(122), plt.imshow(img_static - img_warped), plt.title('warped - target
↪'), plt.colorbar()
print('SSD warped-target:', np.sum((img_static - img_warped) ** 2))
```

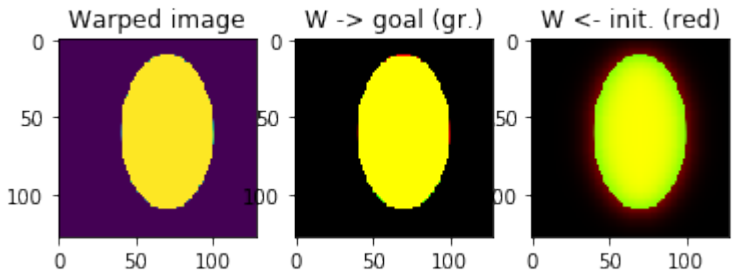
```
('SSD input-target:', 446.38491288535107)
('SSD warped-target:', 39.534977233930519)
```



Invert transformation: **static -> static FUZZY** image

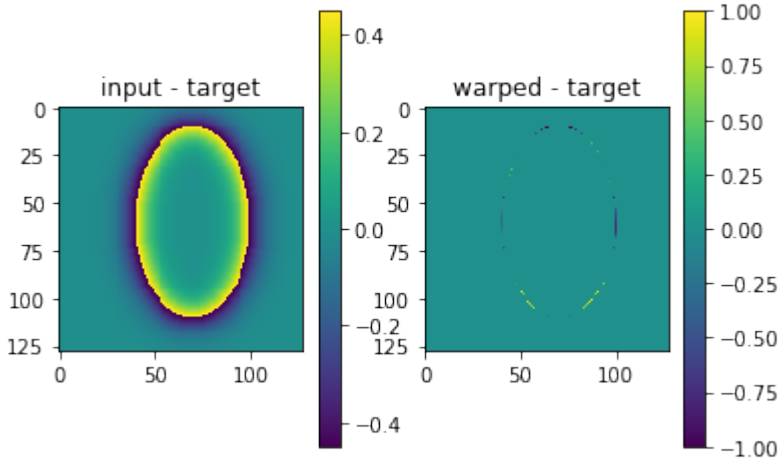
[17]:
```
nb_utils.show_registered_deformation(mapping_inv.forward)
img_warped = mapping_inv.transform_inverse(img_static, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_static_fuzzy)
```





[18]:
```
_= plt.subplot(121), plt.imshow(img_static - img_static_fuzzy), plt.title('input -␣
↪target'), plt.colorbar()
print('SSD input-target:', np.sum((img_static - img_static_fuzzy) ** 2))
_= plt.subplot(122), plt.imshow(img_static - img_warped), plt.title('warped - target
↪'), plt.colorbar()
print('SSD warped-target:', np.sum((img_static - img_warped) ** 2))
```

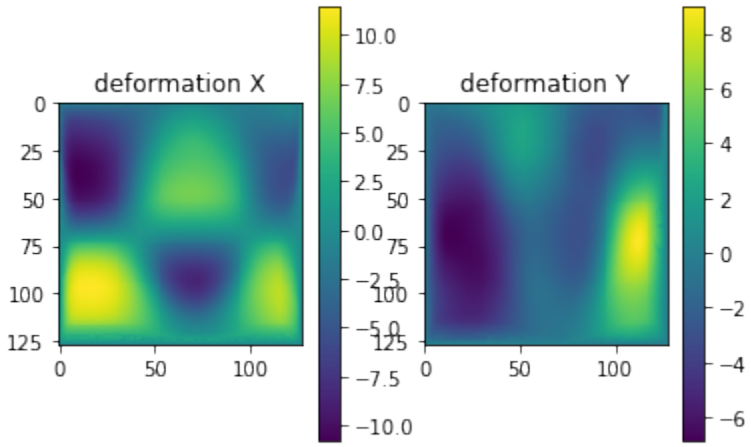
```
('SSD input-target:', 446.38491288535107)
('SSD warped-target:', 34.501153207744082)
```

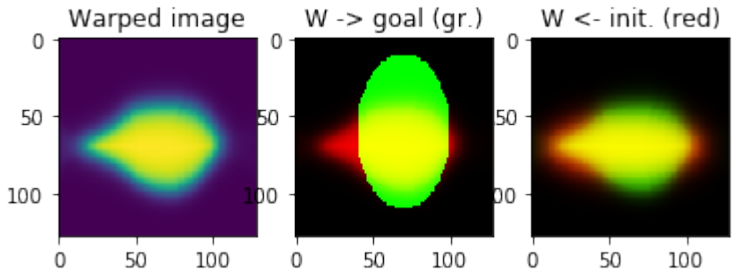


Register: **static <- moving FUZZY** image

[19]:
```
mapping, mapping_inv = register_dipy_smooth(img_static, img_moving_fuzzy, 10.)
# _= regtools.plot_2d_diffeomorphic_map(mapping, 10)
nb_utils.show_registered_deformation(mapping.backward)
img_warped = mapping.transform(img_moving_fuzzy, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_moving_fuzzy)
```

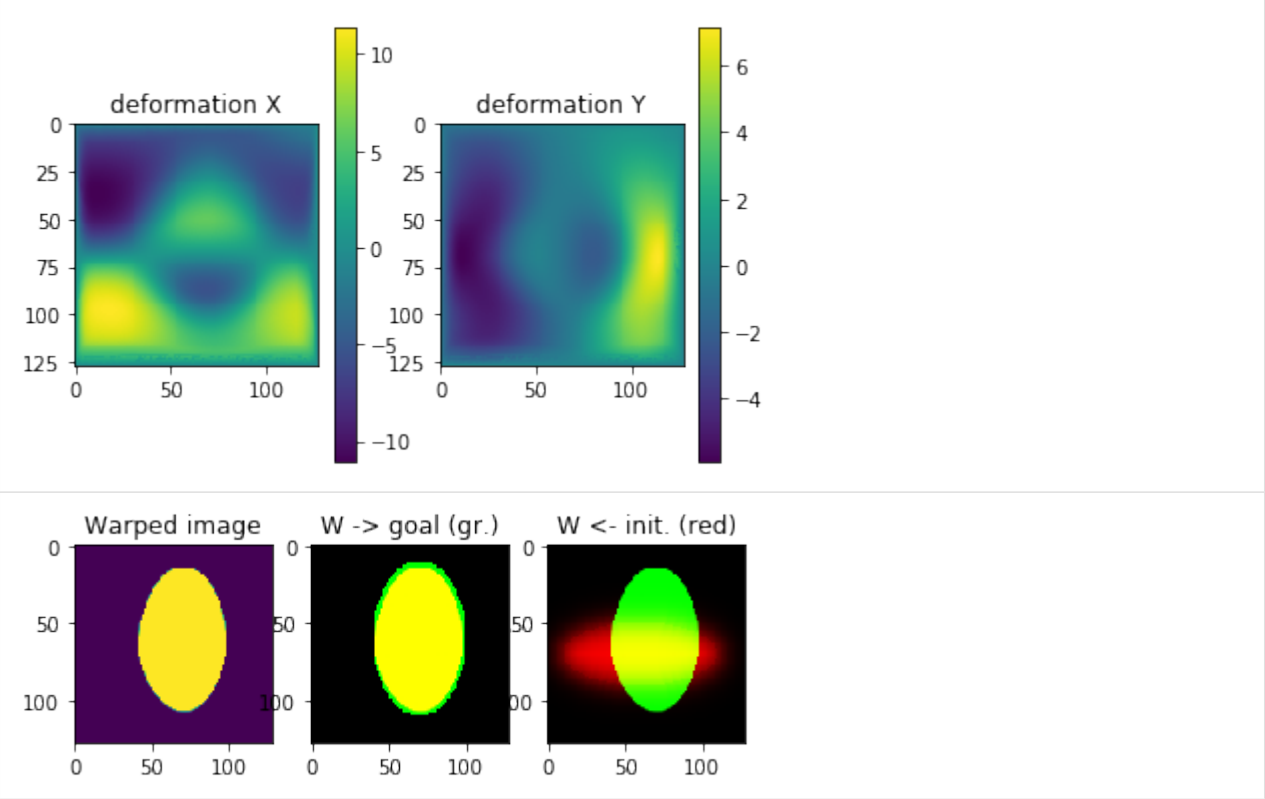
```
Creating scale space from the moving image. Levels: 2. Sigma factor: 10.000000.
Creating scale space from the static image. Levels: 2. Sigma factor: 10.000000.
Optimizing level 1
Optimizing level 0
('optimize took:', 0.3405900001525879)
```





Invert transformation: **static -> moving FUZZY** image

[20]:
```
nb_utils.show_registered_deformation(mapping_inv.backward)
img_warped = mapping_inv.transform_inverse(img_static, 'linear')
nb_utils.show_registered_overlap(img_warped, img_static, img_moving_fuzzy)
```



[ ]:

1.4.11 All results on Synthetic datasets - Binary images

Presenting results all state-of-the-art methods togehter with our APDL method

[2]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys, glob
import pandas, numpy
from skimage import io
import matplotlib.pylab as plt
from matplotlib import gridspec
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import bpdl.utilities as utils
import bpdl.data_utils as tl_data
```

```
/usr/local/lib/python2.7/dist-packages/matplotlib/__init__.py:1405: UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

  warnings.warn(_use_error_msg)
```

[5]:
```
p_results = utils.update_path('results')
p_csv = os.path.join(p_results, 'experiments_synth_APD_binary_overall.csv')
print(os.path.exists(p_csv), '<-', p_csv)
```

(continues on next page)

(continued from previous page)

```
p_data = '/mnt/F464B42264B3E590/TEMP'
DATASET = 'atomicPatternDictionary_v0'
print(os.path.exists(p_data), '<-', p_data)
```

```
True <- results/experiments_synth_APD_binary_overall.csv
True <- /mnt/F464B42264B3E590/TEMP
```

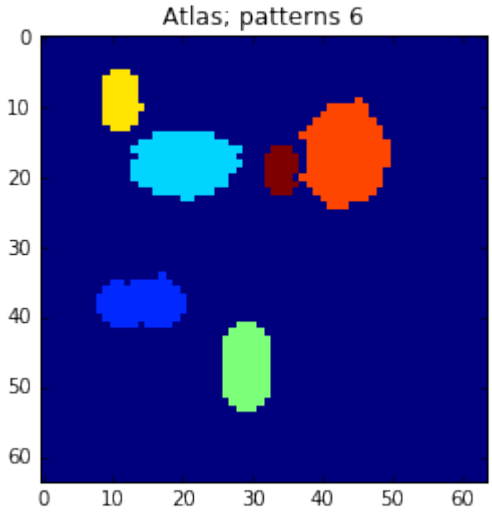
Loading data

[3]:
```
df_all = pandas.read_csv(p_csv, index_col=None)
print('-> loaded DF with', len(df_all), 'items and columns:\n', df_all.columns.
↪tolist())
d_unique = {col: len(df_all[col].unique()) for col in df_all.columns}
d_unique = {k: d_unique[k] for k in d_unique if d_unique[k] > 1}
df_all.sort('nb_labels', inplace=True)
print('-> unique:', d_unique)
```

```
-> loaded DF with 915 items and columns:
['nb_labels', 'atlas_ARS', 'reconstruct_diff', 'time', 'folders', 'overlap_major',
↪'name', 'nb_labels.1', 'nb_workers', 'nb_runs', 'max_iter', 'path_out', 'nb_samples
↪', 'dataset', 'method', 'ptn_split', 'computer', 'path_exp', 'gc_regul', 'path_in',
↪'subfiles', 'type', 'class', 'init_tp', 'gc_reinit']
-> unique: {'name': 16, 'nb_labels': 28, 'class': 5, 'dataset': 4, 'atlas_ARS': 784,
↪'path_exp': 65, 'time': 915, 'reconstruct_diff': 719, 'path_in': 4, 'method': 5}
```

```
/usr/local/lib/python2.7/dist-packages/IPython/kernel/__main__.py:5: FutureWarning:␣
↪sort(columns=...) is deprecated, use sort_values(by=...)
```

[4]:
```
atlas = tl_data.dataset_compose_atlas(os.path.join(p_data, DATASET))
plt.imshow(atlas, interpolation='nearest')
_ = plt.title('Atlas; patterns {}'.format(numpy.unique(atlas).shape[0] - 1))
```



Dependency in number of used patterns

take out the series with various param combination

[5]:
```
df_select = df_all[df_all['path_in'].str.endswith(DATASET)]
# df_select = df_select[df_select['nb_samples'] == None]
print ('number of selected', len(df_select))
df_res = pandas.DataFrame()
for v, df_gr0 in df_select.groupby('dataset'):
    for v1, df_gr1 in df_gr0.groupby('class'):
            d = {'dataset': v, 'class': v1}
            cols = ['nb_labels', 'atlas_ARS', 'reconstruct_diff', 'time']
            d.update({col: df_gr1[col].tolist() for col in cols})
            df_res = df_res.append(d, ignore_index=True)
# df_res = df_res.set_index('class')
print ('number of rows:', len(df_res), 'columns:', df_res.columns.tolist())
```

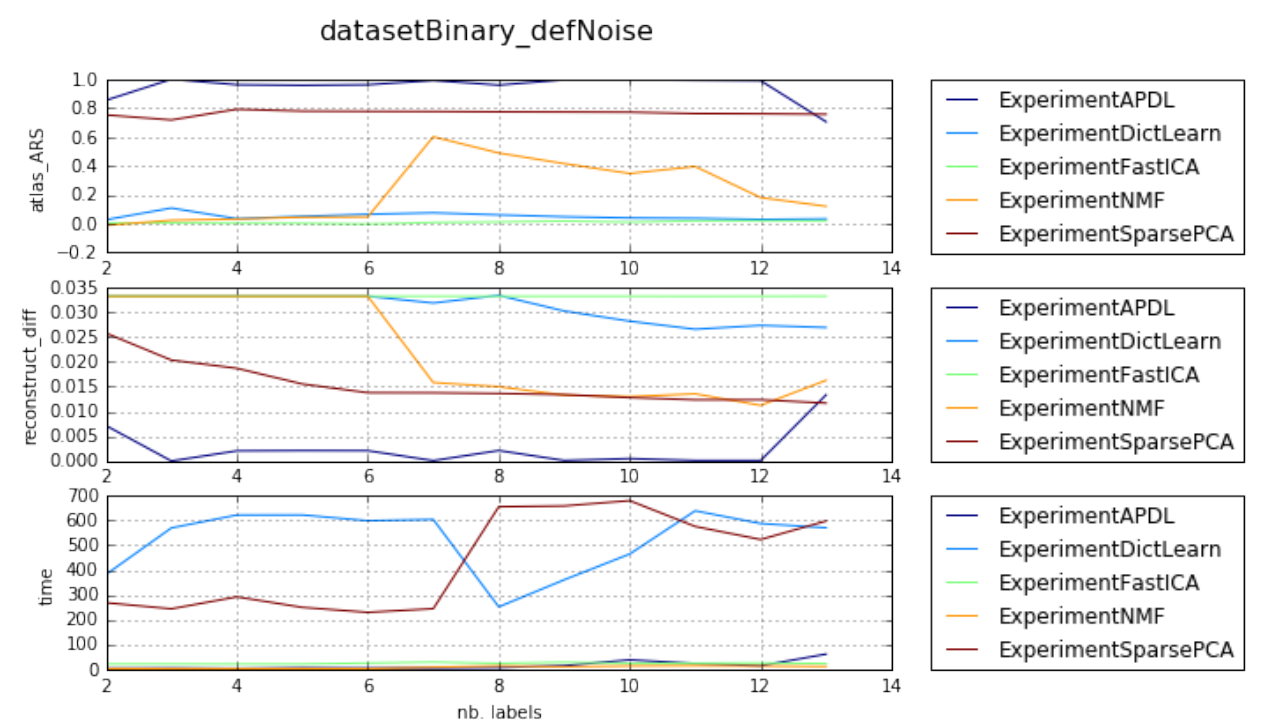
```
number of selected 240
number of rows: 20 columns: ['atlas_ARS', 'class', 'dataset', 'nb_labels',
↪'reconstruct_diff', 'time']
```

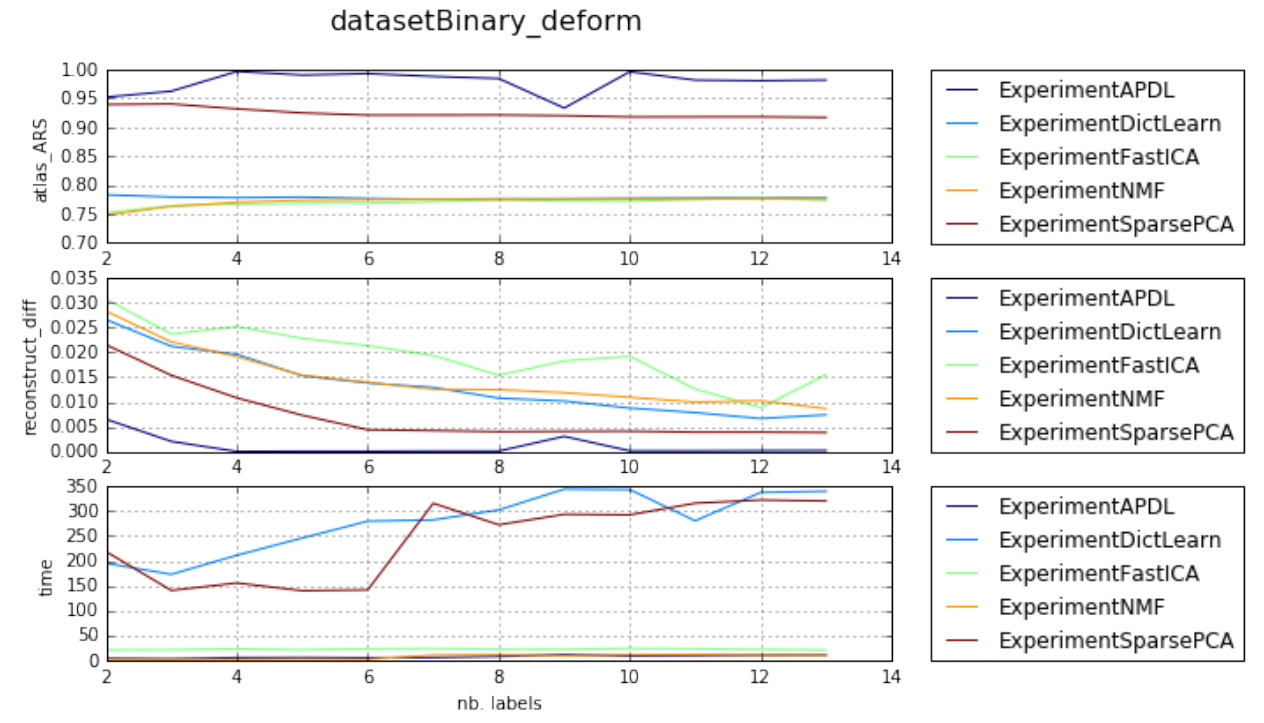
[8]:
```
def plot_results_graph(df_res, n_group, n_curve, l_graphs=('atlas_ARS', 'reconstruct_
↪diff', 'time')):
    for v, df_group in df_res.groupby(n_group):
        clrs = plt.cm.jet(numpy.linspace(0, 1, len(df_group)))
        fig, axarr = plt.subplots(len(l_graphs), 1, figsize=(8, 6))
        fig.suptitle('{}'.format(v), fontsize=16)
        for i, col in enumerate(l_graphs):
            for j, (idx, row) in enumerate(df_group.iterrows()):
                axarr[i].plot(row['nb_labels'], row[col], label=row[n_curve],␣
↪color=clrs[j])
            axarr[i].set_xlabel('nb. labels')
            axarr[i].set_ylabel(col)
            axarr[i].legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
            axarr[i].grid()
        # print v
```

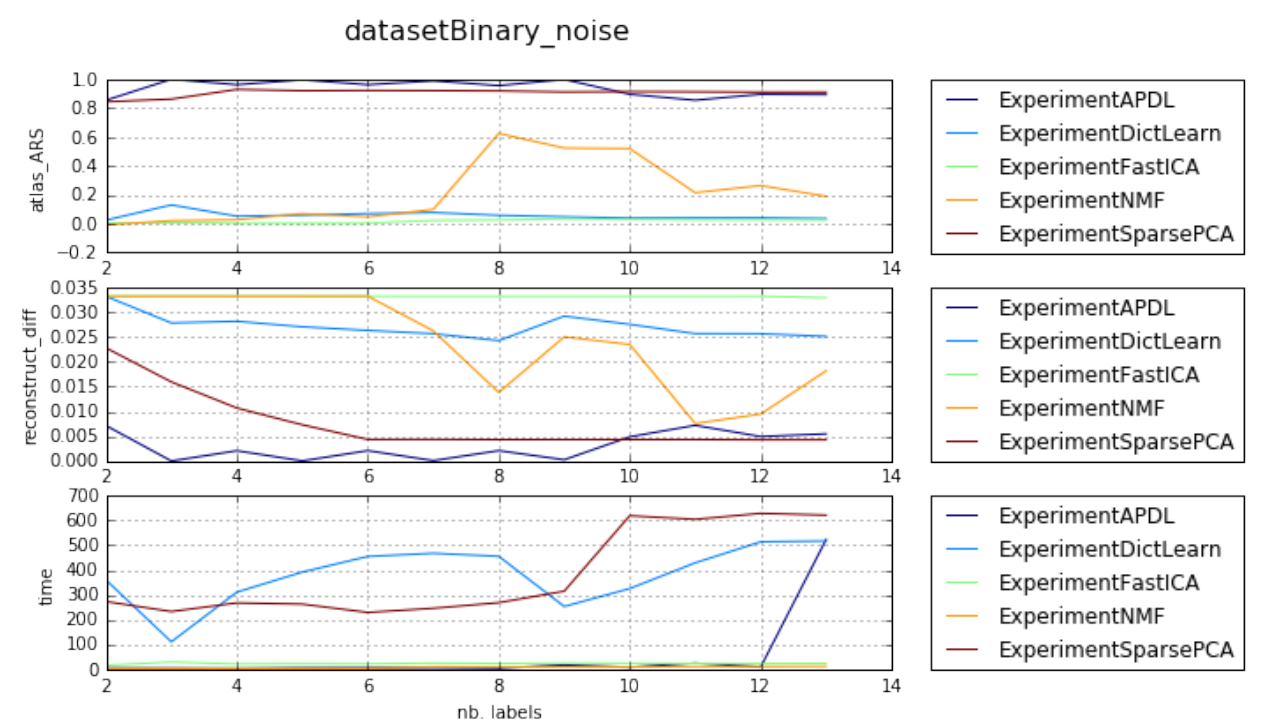
Plots by datasets

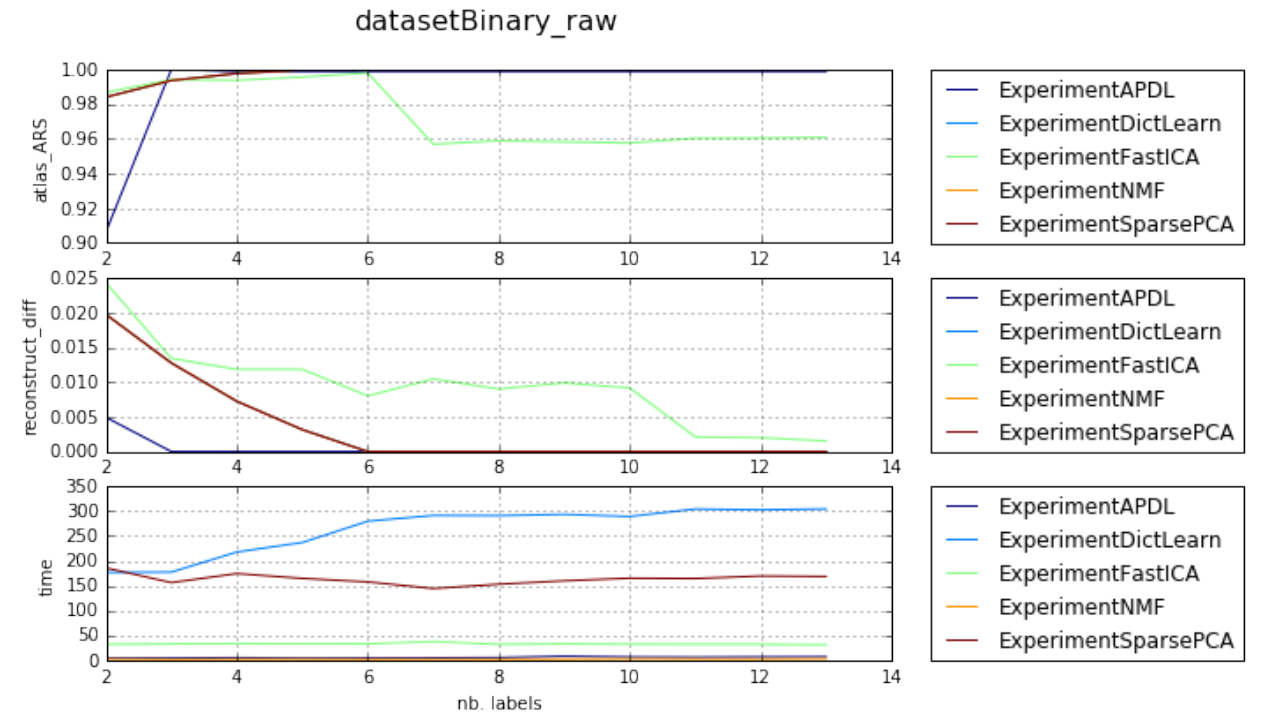
visualization per dataset (difficulty) and different param combination

[10]:
```
plot_results_graph(df_res, 'dataset', 'class')
```


datasetBinary_defNoise
atlas_ARS
reconstruct_diff
time
nb. labels
ExperimentAPDL
ExperimentDictLearn
ExperimentFastICA
ExperimentNMF
ExperimentSparsePCA

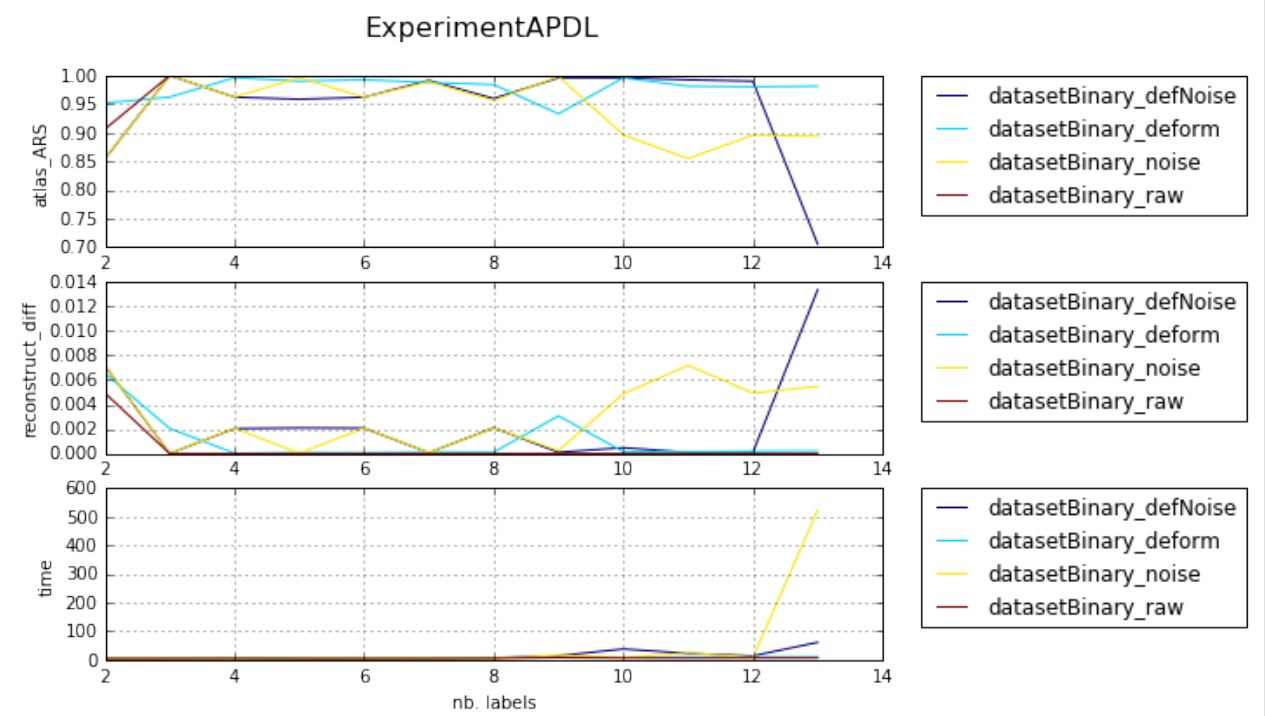

datasetBinary_deform
atlas_ARS
reconstruct_diff
time
nb. labels
ExperimentAPDL
ExperimentDictLearn
ExperimentFastICA
ExperimentNMF
ExperimentSparsePCA

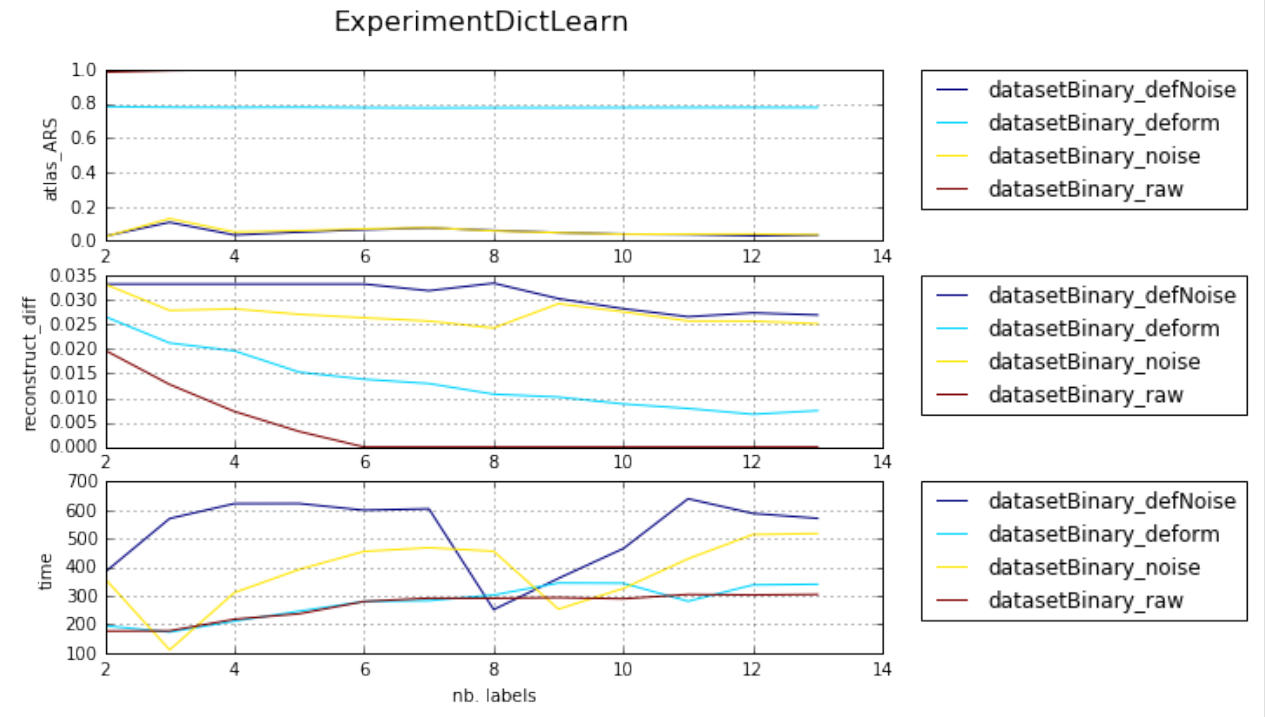

datasetBinary_noise
atlas_ARS
reconstruct_diff
time
nb. labels
ExperimentAPDL
ExperimentDictLearn
ExperimentFastICA
ExperimentNMF
ExperimentSparsePCA

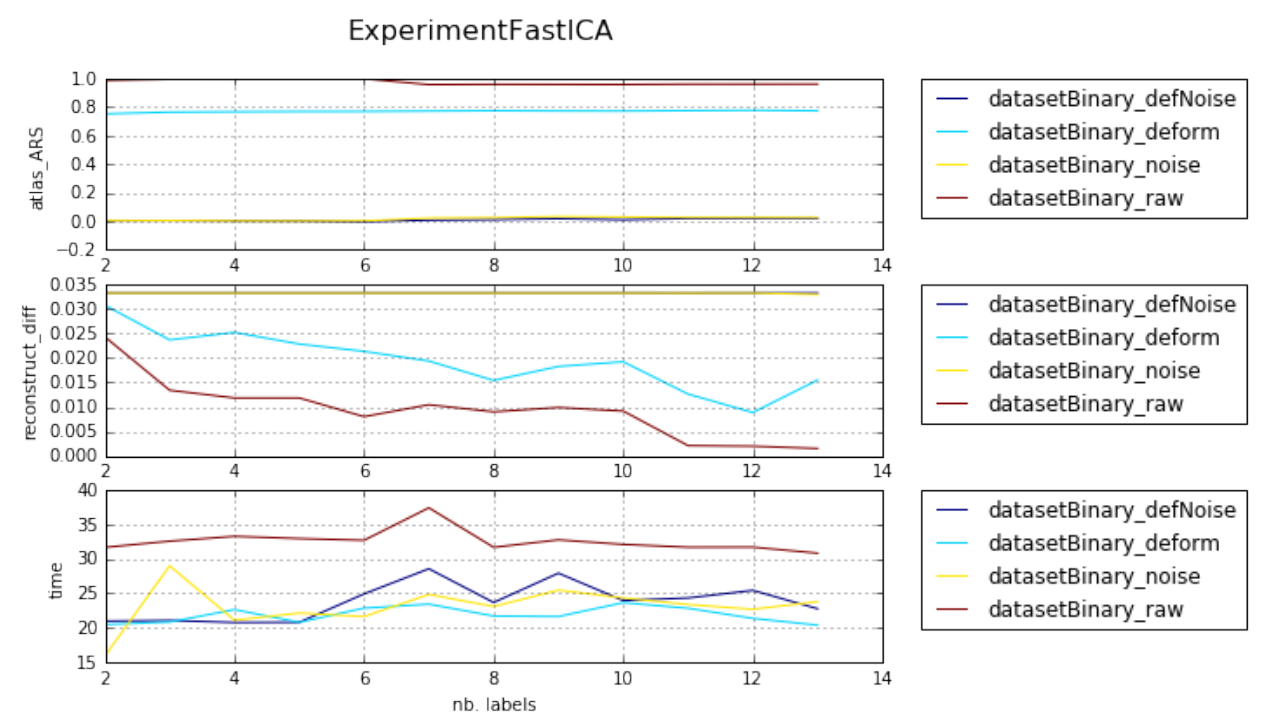

datasetBinary_raw
atlas_ARS
reconstruct_diff
time
nb. labels
ExperimentAPDL
ExperimentDictLearn
ExperimentFastICA
ExperimentNMF
ExperimentSparsePCA

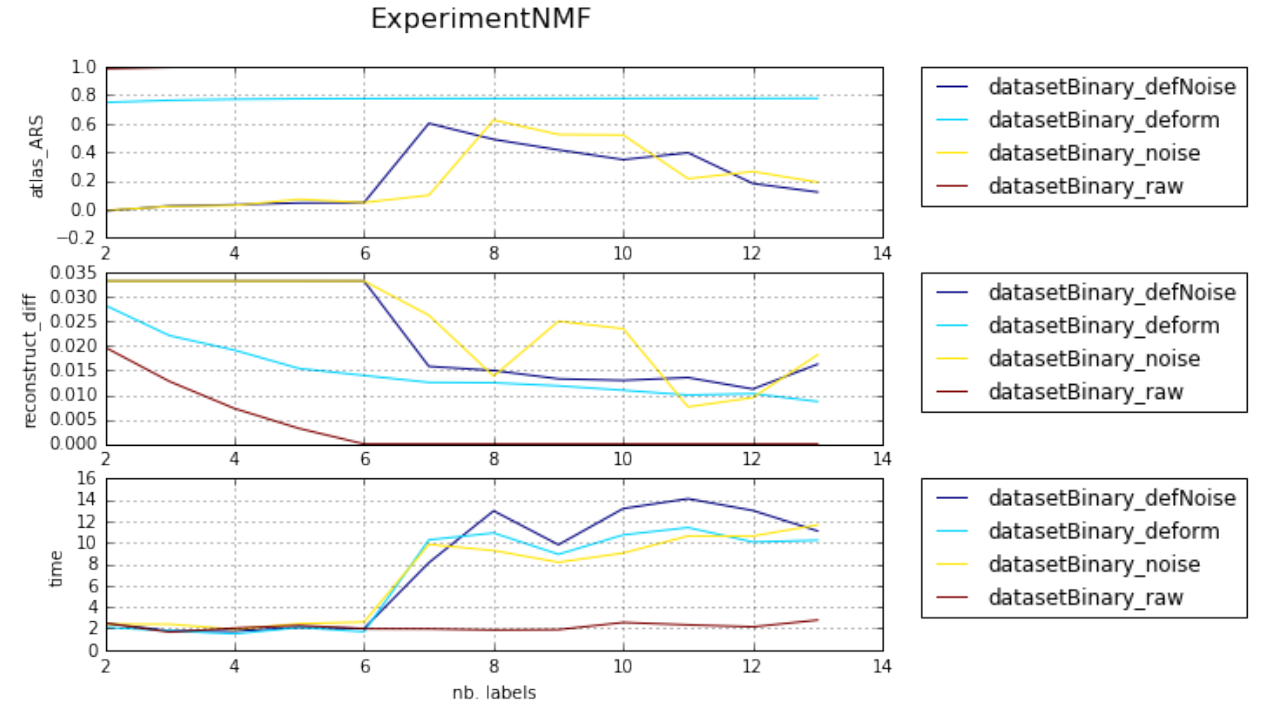
Plots by methods

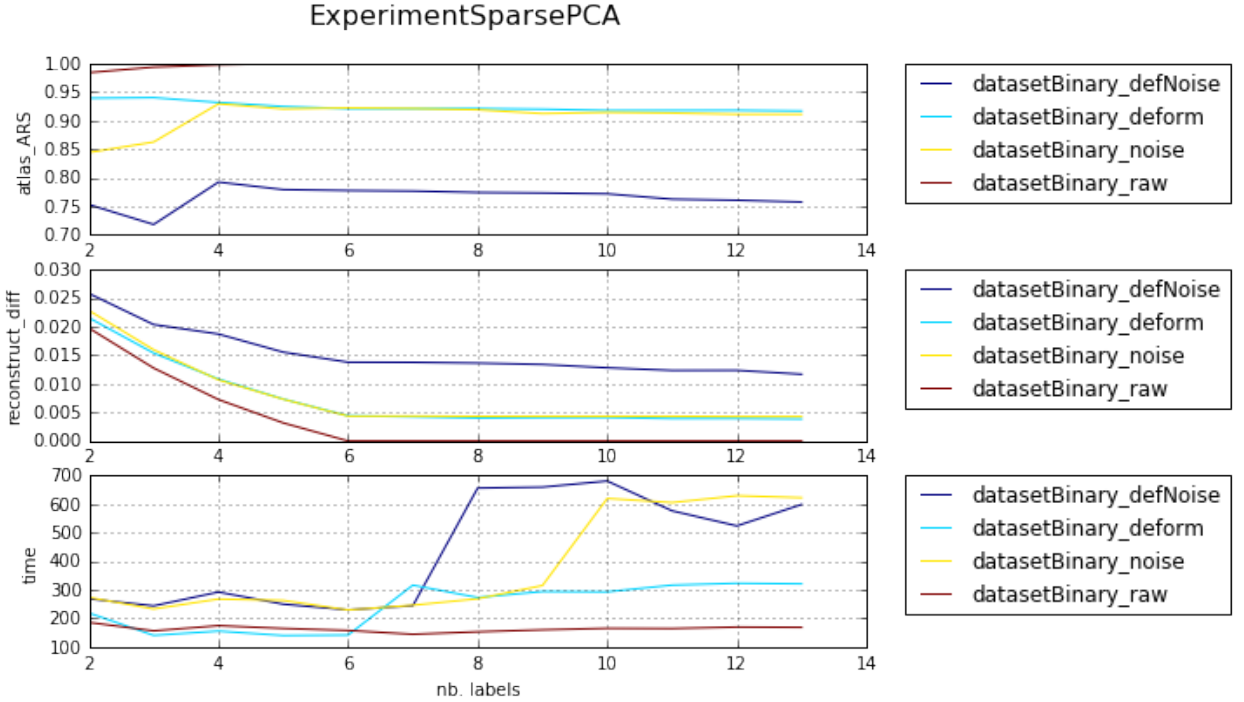
[11]:
```
plot_results_graph(df_res, 'class', 'dataset')
```






ExperimentFastICA
atlas_ARS
reconstruct_diff
time
nb. labels
datasetBinary_defNoise
datasetBinary_deform
datasetBinary_noise
datasetBinary_raw


ExperimentNMF
atlas_ARS
reconstruct_diff
time
nb. labels
datasetBinary_defNoise
datasetBinary_deform
datasetBinary_noise
datasetBinary_raw



[ ]:

1.4.12 BPDL results on Synthetic datasets - binary images

Presenting results from starting version of BPDL

[1]:
```
%matplotlib inline
import os, sys, glob
import pandas, numpy
from skimage import io
import matplotlib.pylab as plt
from matplotlib import gridspec
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import bpdl.utilities as utils
import bpdl.data_utils as tl_data
```

```
/usr/local/lib/python2.7/dist-packages/matplotlib/__init__.py:1405: UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

  warnings.warn(_use_error_msg)
```

[2]:
```
p_results = utils.update_path('results')
p_csv = os.path.join(p_results, 'experiments_synth_APDL_binary_overall.csv')
print (os.path.exists(p_csv), '<-', p_csv)

p_data = '/mnt/F464B42264B3E590/TEMP'
DATASET = 'atomicPatternDictionary_v0'
print (os.path.exists(p_data), '<-', p_data)
```

```
True <- results/experiments_synth_APDL_binary_overall.csv
True <- /mnt/F464B42264B3E590/TEMP
```

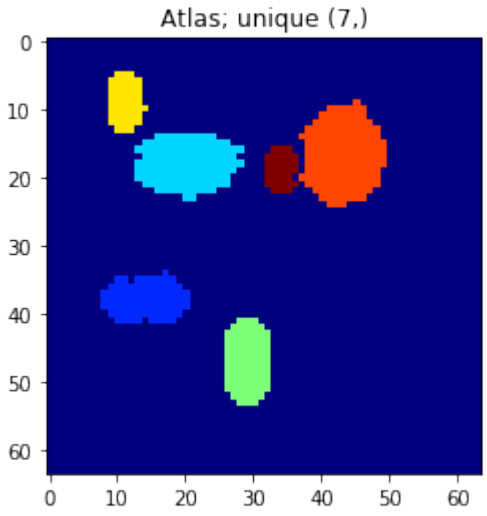
Loading data

[3]:
```
df_all = pandas.read_csv(p_csv, index_col=None)
df_all.sort(['nb_labels'], inplace=True)
print ('-> loaded DF with', len(df_all), 'items and columns:\n', df_all.columns.
↪tolist())
d_unique = {col: df_all[col].unique().tolist()
            for col in ['dataset', 'gc_regul', 'gc_reinit', 'init_tp', 'ptn_split']}
print ('-> unique:', d_unique)
```

```
-> loaded DF with 11683 items and columns:
['atlas_ARS', 'case', 'class', 'computer', 'dataset', 'folders', 'gc_regul', 'gc_
↪reinit', 'init_tp', 'max_iter', 'name', 'nb_workers', 'nb_labels', 'nb_res', 'nb_
↪runs', 'nb_samples', 'overlap_major', 'path_exp', 'path_in', 'path_out', 'ptn_
↪compact', 'ptn_split', 'reconstruct_diff', 'subfiles', 'time', 'tol', 'type']
-> unique: {'init_tp': ['msc2', 'GTd', 'msc', 'rnd', 'msc1', 'GT', 'OWSr', 'OWS', 'GWS
↪'], 'ptn_split': [False, True], 'gc_reinit': [True, nan], 'gc_regul': [0.001, 0.0,␣
↪9.999999999999999e-16, 1e-09], 'dataset': ['datasetBinary_noise', 'datasetBinary_raw
↪', 'datasetBinary_defNoise', 'datasetBinary_deform']}
```

```
/usr/local/lib/python2.7/dist-packages/IPython/kernel/__main__.py:2: FutureWarning:␣
↪sort(columns=...) is deprecated, use sort_values(by=...)
  from IPython.kernel.zmq import kernelapp as app
```

[5]:
```
atlas = tl_data.dataset_compose_atlas(os.path.join(p_data, DATASET))
plt.imshow(atlas, interpolation='nearest', cmap=plt.cm.jet)
_ = plt.title('Atlas; unique {}'.format(numpy.unique(atlas).shape))
```



[6]:
```
print('path_in:', df_all['path_in'].unique())
print('gc_regul:', df_all['gc_regul'].unique())
```

```
path_in: [ '/datagrid/Medical/microscopy/drosophila/synthetic_data/
↪atomicPatternDictionary_v0'
 '/datagrid/Medical/microscopy/drosophila/synthetic_data/atomicPatternDictionary_v1'
```

(continues on next page)

(continued from previous page)

```
 '/datagrid/Medical/microscopy/drosophila/synthetic_data/atomicPatternDictionary_v2'
 '/datagrid/Medical/microscopy/drosophila/synthetic_data/atomicPatternDictionary_v3']
gc_regul: [  1.00000000e-03   0.00000000e+00   1.00000000e-15   1.00000000e-09]
```

Dependency on number of used patterns

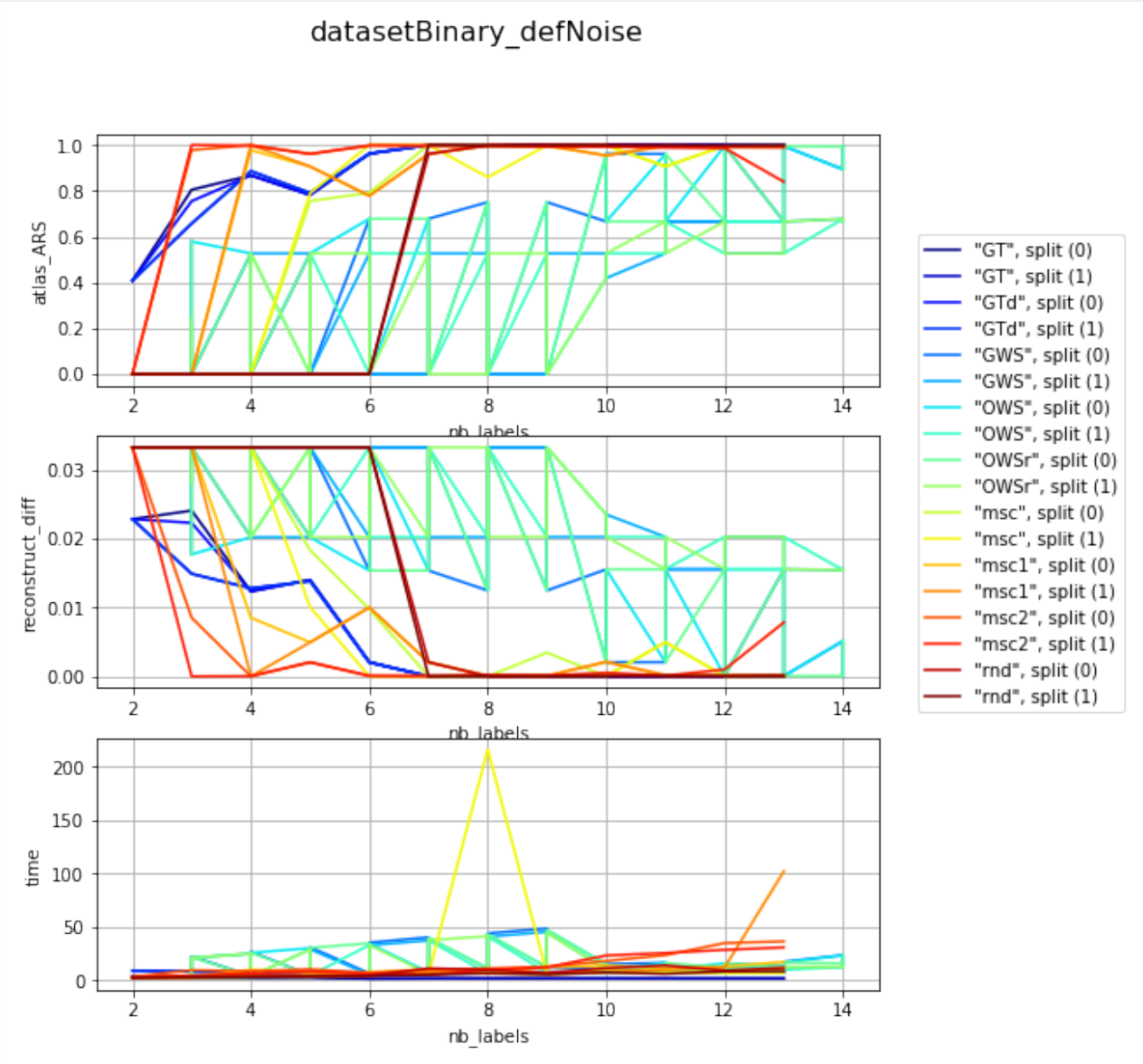
take out the series with various param combination

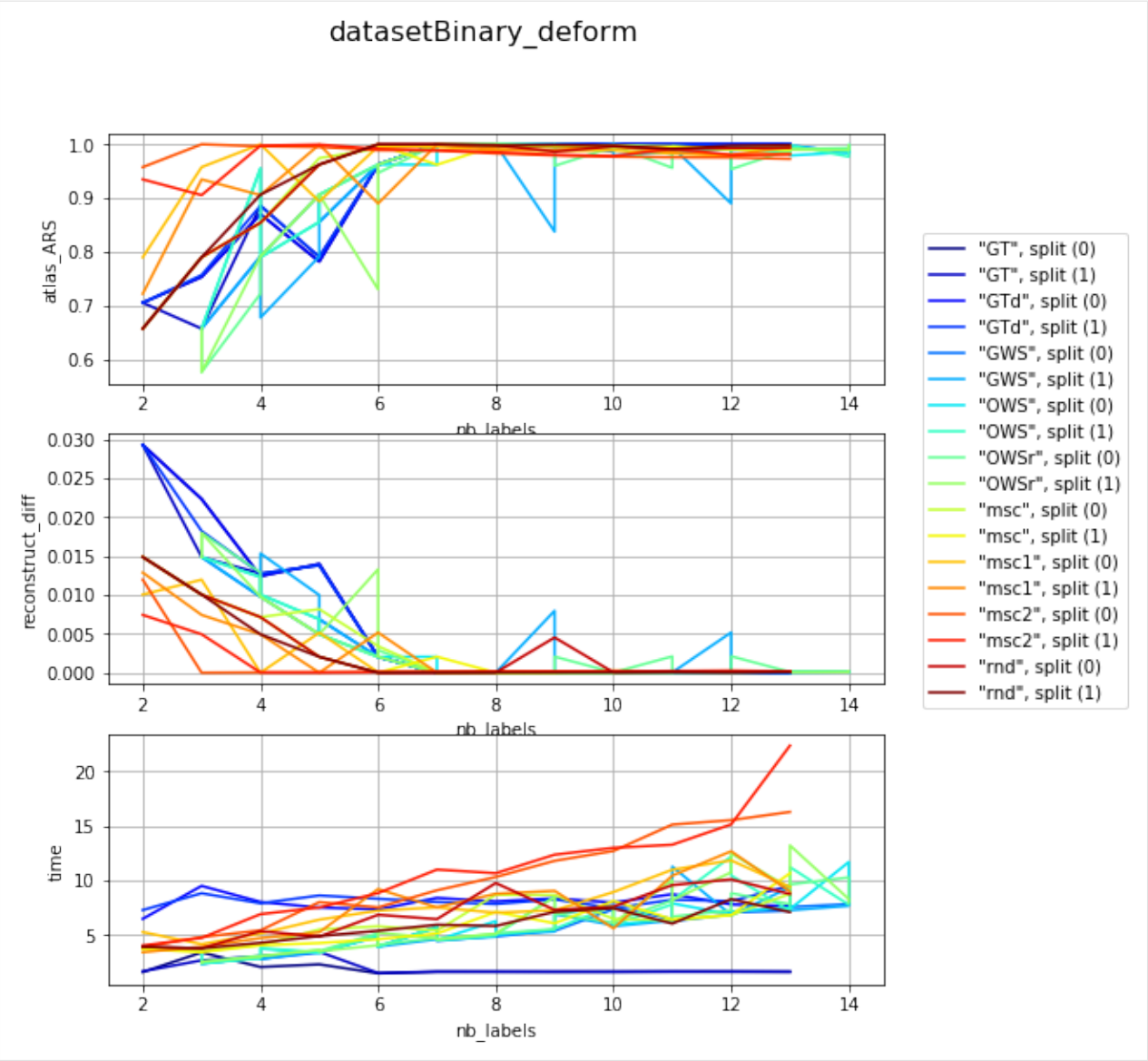
[7]:
```
GC_REGUL = 0.0
df_select = df_all[df_all['path_in'].str.endswith(DATASET)]
df_select = df_select[df_select['gc_regul'] == GC_REGUL]
# df_select = df_select[df_select['ptn_split'] == False]
print ('selected records:', len(df_select))
df_res = pandas.DataFrame()
for v, df_gr0 in df_select.groupby('dataset'):
    for v1, df_gr1 in df_gr0.groupby('init_tp'):
        for v2, df_gr2 in df_gr1.groupby('ptn_split'):
            # for v1, df_gr1 in df_gr0.groupby('overlap_major'):
            name = '"{}", split ({})'.format(v1, int(v2))
            d = {'dataset': v, 'name': name}
            cols = ['nb_labels', 'atlas_ARS', 'reconstruct_diff', 'time', 'subfiles']
            d.update({col: df_gr2[col].tolist() for col in cols})
            df_res = df_res.append(d, ignore_index=True)
df_res = df_res.set_index('name')
# df_res.to_csv(os.path.join(os.path.dirname(p_csv), 'synth_APDL_%s_gc_%f.csv' %␣
↪(DATASET, GC_REGUL)))
print ('number of rows:', len(df_res), 'columns:', df_res.columns.tolist())
```

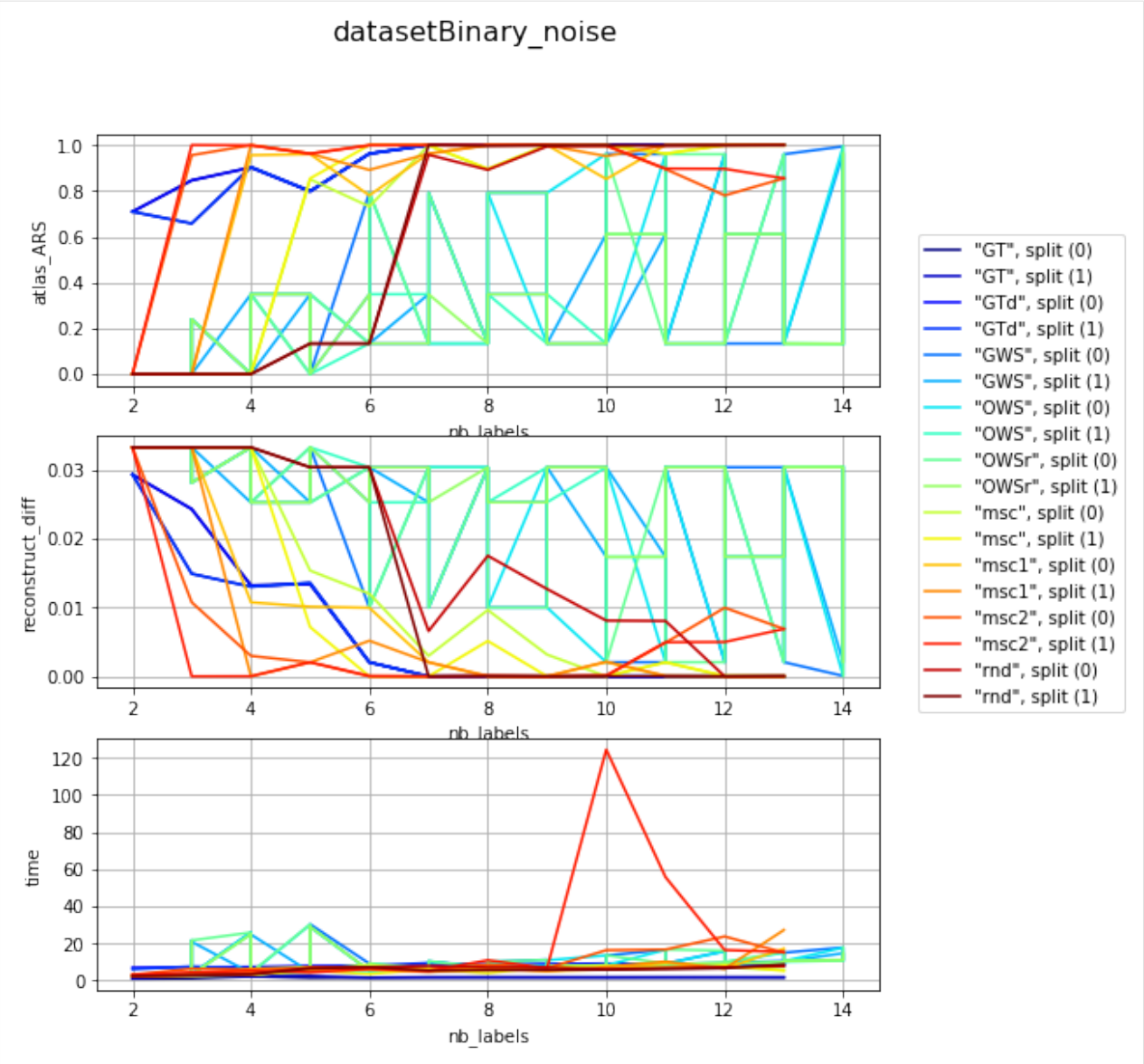
```
selected records: 1152
number of rows: 72 columns: ['atlas_ARS', 'dataset', 'nb_labels', 'reconstruct_diff',
↪'subfiles', 'time']
```

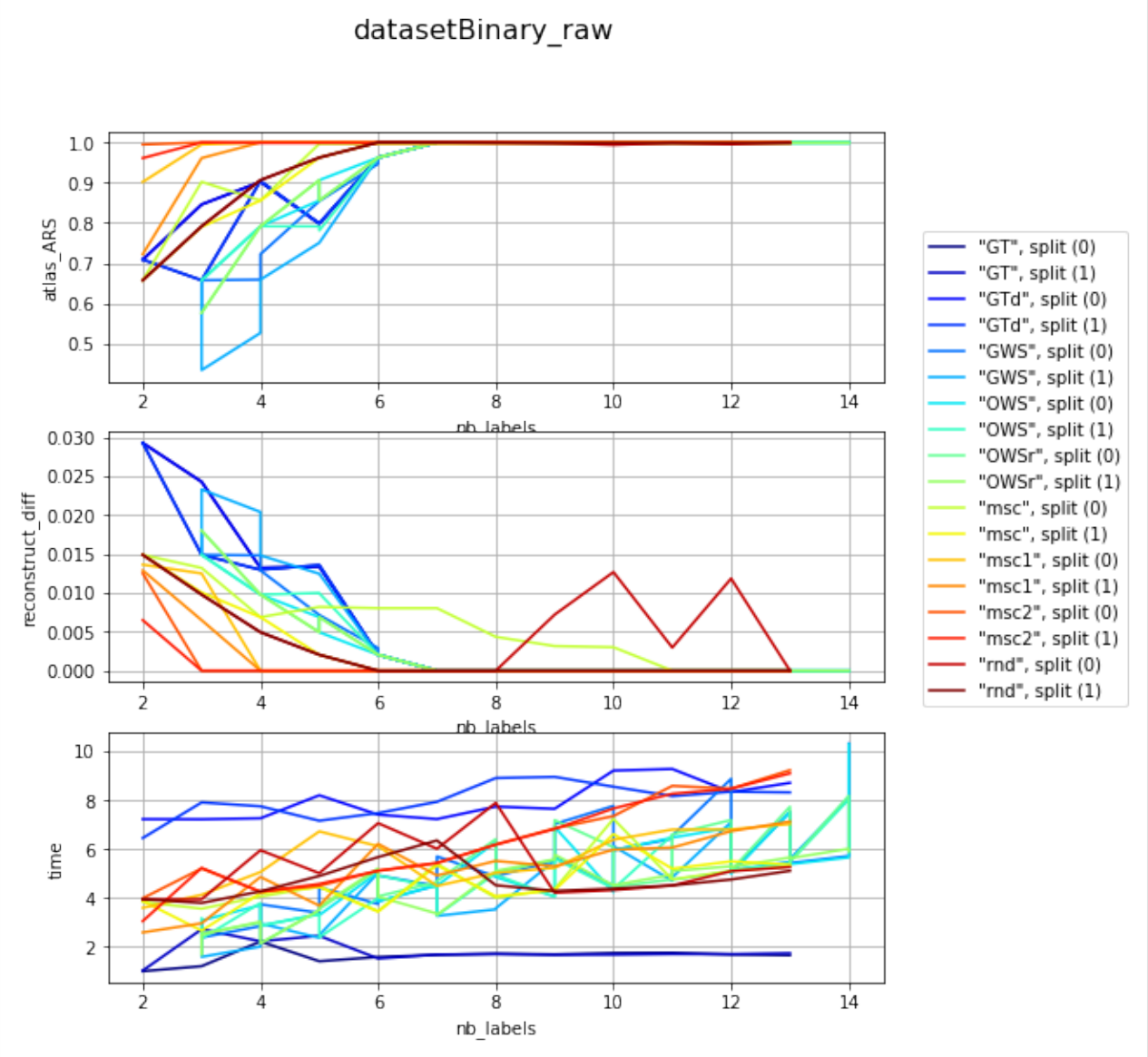
visualization per dataset (difficulty) and different param combination; sub fules corelate with mean number of iterartions ~ (subfiles / 2)

[13]:
```
# ADD graph ['nb_lbs', 'atlas_ARS', 'reconstruct_diff', 'subfiles'] and name as curve
l_graphs = ['atlas_ARS', 'reconstruct_diff', 'time']
for v, df_group in df_res.groupby('dataset'):
    clrs = plt.cm.jet(numpy.linspace(0, 1, len(df_group)))
    fig, axarr = plt.subplots(len(l_graphs), 1, figsize=(8, len(l_graphs) * 3))
    fig.suptitle('{}'.format(v), fontsize=16)
    for i, col in enumerate(l_graphs):
        for j, (idx, row) in enumerate(df_group.iterrows()):
            axarr[i].plot(row['nb_labels'], row[col], label=idx, color=clrs[j])
        axarr[i].set_xlabel('nb_labels'), axarr[i].set_ylabel(col), axarr[i].grid()
    plt.legend(bbox_to_anchor=(1.05, len(l_graphs)), loc=2, borderaxespad=0.)
```


datasetBinary_defNoise
atlas_ARS
reconstruct_diff
time
nb_labels
"GT", split (0)
"GT", split (1)
"GTd", split (0)
"GTd", split (1)
"GWS", split (0)
"GWS", split (1)
"OWS", split (0)
"OWS", split (1)
"OWSr", split (0)
"OWSr", split (1)
"msc", split (0)
"msc", split (1)
"msc1", split (0)
"msc1", split (1)
"msc2", split (0)
"msc2", split (1)
"rnd", split (0)
"rnd", split (1)


datasetBinary_deform
atlas_ARS
reconstruct_diff
time
nb_labels
"GT", split (0)
"GT", split (1)
"GTd", split (0)
"GTd", split (1)
"GWS", split (0)
"GWS", split (1)
"OWS", split (0)
"OWS", split (1)
"OWSr", split (0)
"OWSr", split (1)
"msc", split (0)
"msc", split (1)
"msc1", split (0)
"msc1", split (1)
"msc2", split (0)
"msc2", split (1)
"rnd", split (0)
"rnd", split (1)


datasetBinary_noise
atlas_ARS
reconstruct_diff
time
nb_labels
"GT", split (0)
"GT", split (1)
"GTd", split (0)
"GTd", split (1)
"GWS", split (0)
"GWS", split (1)
"OWS", split (0)
"OWS", split (1)
"OWSr", split (0)
"OWSr", split (1)
"msc", split (0)
"msc", split (1)
"msc1", split (0)
"msc1", split (1)
"msc2", split (0)
"msc2", split (1)
"rnd", split (0)
"rnd", split (1)



[ ]:

1.4.13 All results on Synthetic datasets - Prob. images

Presenting results all state-of-the-art methods togehter with our APDL method

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys, glob
import pandas, numpy
from skimage import io
import matplotlib.pylab as plt
from matplotlib import gridspec
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
```

(continues on next page)

(continued from previous page)

```
import bpdl.data_utils as tl_data
from notebooks.notebook_utils import filter_df_results_4_plotting, plot_bpdl_graph_
↪results
```

```
/usr/local/lib/python2.7/dist-packages/matplotlib/__init__.py:1405: UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

  warnings.warn(_use_error_msg)
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

[2]:
```
# p_csv = os.path.expanduser('~/Dropbox/Documents/lab_CMP-BIA/paper_2017_drosophila_
↪APDL/data/experiments_synth_APD_prob_results_NEW_OVERALL.csv')
p_csv = os.path.expanduser(os.path.join('results', 'experiments_synth_APD_prob_
↪results_NEW_OVERALL.csv'))
print os.path.exists(p_csv), '<-', p_csv
# DATASET = 'datasetFuzzy_raw'
# p_data = '/mnt/F464B42264B3E590/TEMP/atomicPatternDictionary_v0'
```

```
True <- results/experiments_synth_APD_prob_results_NEW_OVERALL.csv
```

Loading data

[5]:
```
df_all = pandas.read_csv(p_csv, index_col=None)
df_all.dropna(subset=['nb_labels', 'atlas_ARS', 'reconstruct_diff'], inplace=True)
print('-> loaded DF with', len(df_all), 'items and columns:\n', df_all.columns.
↪tolist())
d_unique = {col: df_all[col].unique() for col in df_all.columns}
df_all.sort_values('nb_labels', inplace=True)
print('-> unique:', {k: len(d_unique[k]) for k in d_unique if len(d_unique[k]) > 10})
```

```
-> loaded DF with 24357 items and columns:
['nb_labels', 'atlas ARS', 'atlas accuracy', 'atlas f1_macro', 'atlas f1_weighted',
↪'atlas precision_macro', 'atlas precision_weighted', 'atlas recall_macro', 'atlas␣
↪recall_weighted', 'atlas support_macro', 'atlas support_weighted', 'atlas_ARS',
↪'reconstruct_diff', 'time', 'folders', 'max_iter', 'path_out', 'dataset', 'computer
↪', 'path_exp', 'files @dir', 'class', 'nb_runs', 'name', 'overlap_major', 'gc_reinit
↪', 'ptn_compact', 'nb_workers', 'nb_samples', 'ptn_split', 'gc_regul', 'tol', 'path_
↪in', 'type', 'method', 'init_tp']
-> unique: {'atlas ARS': 14682, 'atlas f1_weighted': 13931, 'atlas accuracy': 6005,
↪'atlas recall_macro': 14688, 'name': 56, 'nb_labels': 29, 'nb_workers': 4, 'atlas_
↪ARS': 14682, 'class': 10, 'atlas precision_weighted': 6005, 'dataset': 14, 'atlas␣
↪precision_macro': 13411, 'computer': 3, 'path_exp': 5811, 'time': 24356,
↪'reconstruct_diff': 9149, 'method': 5, 'path_in': 4, 'atlas recall_weighted': 14685,
↪ 'nb_runs': 2, 'atlas f1_macro': 13972}
```

```
/usr/local/lib/python2.7/dist-packages/IPython/kernel/__main__.py:1: FutureWarning:␣
↪from_csv is deprecated. Please use read_csv(...) instead. Note that some of the␣
↪default arguments are different, so please refer to the documentation for from_csv␣
↪when changing your function calls
  if __name__ == '__main__':
```

Parse name and noise level

[6]:
```
df_all['version'] = map(os.path.basename, df_all['path_in'])
print('Versions:', df_all['version'].unique().tolist())
```

```
Versions: ['atomicPatternDictionary_00', 'atomicPatternDictionary_v0',
↪'atomicPatternDictionary_v1', 'atomicPatternDictionary_v2']
```

[7]:
```
print('Datasets:', df_all['dataset'].unique().tolist())
```

```
Datasets: ['datasetFuzzy_noise', 'datasetFuzzy_defNoise', 'datasetFuzzy_deform',
↪'datasetFuzzy_raw', 'datasetFuzzy_raw_gauss-0.100', 'datasetFuzzy_raw_gauss-0.001',
↪'datasetFuzzy_raw_gauss-0.200', 'datasetFuzzy_raw_gauss-0.150', 'datasetFuzzy_raw_
↪gauss-0.075', 'datasetFuzzy_raw_gauss-0.025', 'datasetFuzzy_raw_gauss-0.050',
↪'datasetFuzzy_raw_gauss-0.010', 'datasetFuzzy_raw_gauss-0.005', 'datasetFuzzy_raw_
↪gauss-0.125']
```

[8]:
```
noise, dataset_name = [], []
for d in df_all['dataset'].values.tolist():
    if '-' in d:
        noise.append(float(d.split('-')[-1]))
        dataset_name.append(d.split('-')[0])
    else:
        noise.append(None)
        dataset_name.append(d)
df_all['dataset'] = dataset_name
df_all['noise'] = noise
print('Datasets:', df_all['dataset'].unique().tolist())
print('Noise levels:', df_all['noise'].unique().tolist())
```

```
Datasets: ['datasetFuzzy_noise', 'datasetFuzzy_defNoise', 'datasetFuzzy_deform',
↪'datasetFuzzy_raw', 'datasetFuzzy_raw_gauss']
Noise levels: [nan, 0.1, 0.001, 0.2, 0.15, 0.075, 0.025, 0.05, 0.01, 0.005, 0.125]
```

Dependency on iter. parameter

take out the series with various param combination

[9]:
```
# LIST_GRAPHS = ['atlas ARS', 'atlas accuracy', 'atlas f1_weighted', 'atlas precision_
↪weighted', 'atlas recall_weighted', 'reconstruct_diff', 'time']
LIST_GRAPHS = ['atlas ARS', 'atlas f1_weighted', 'reconstruct_diff', 'time']
```

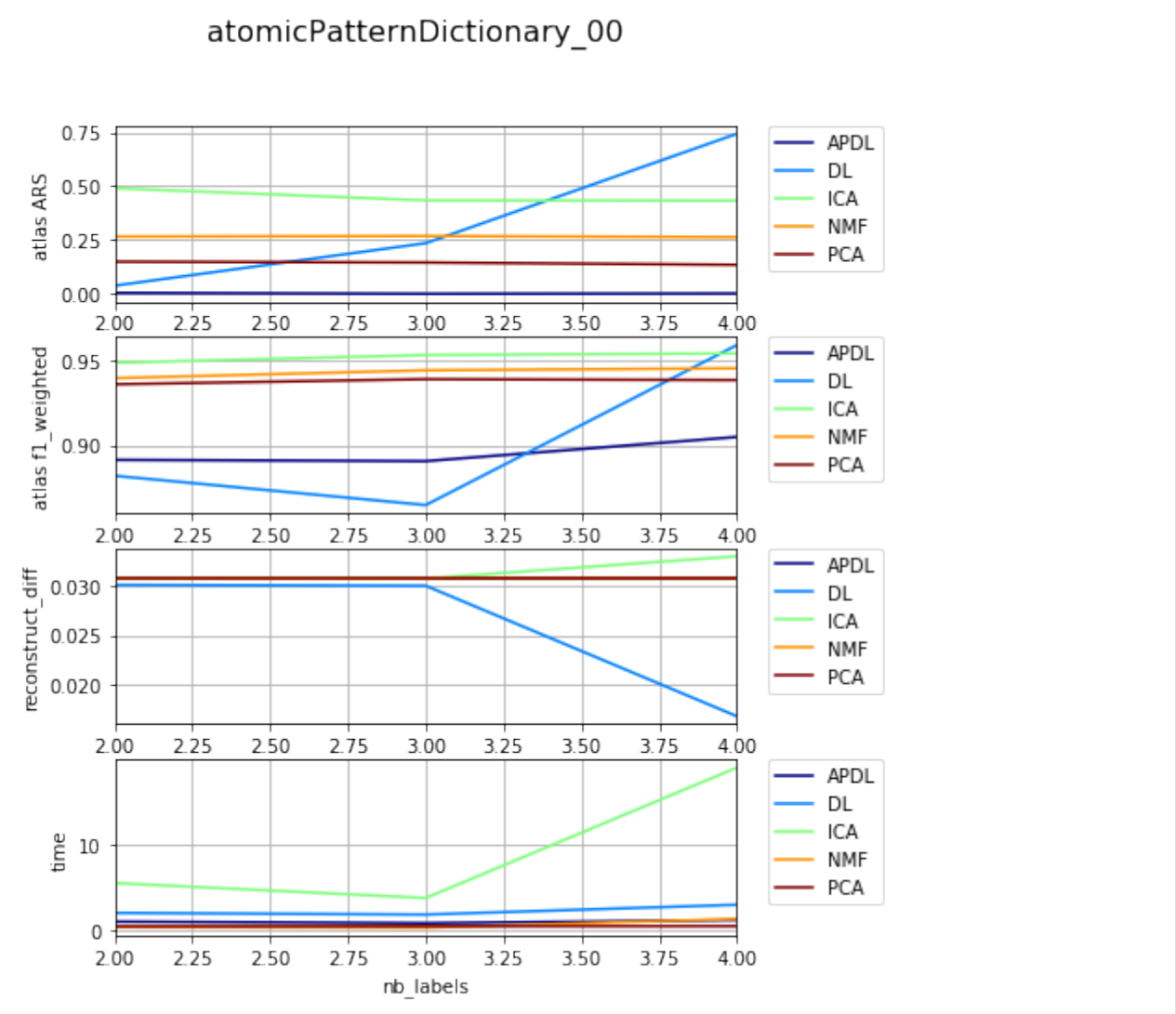
Dependency on number of samples

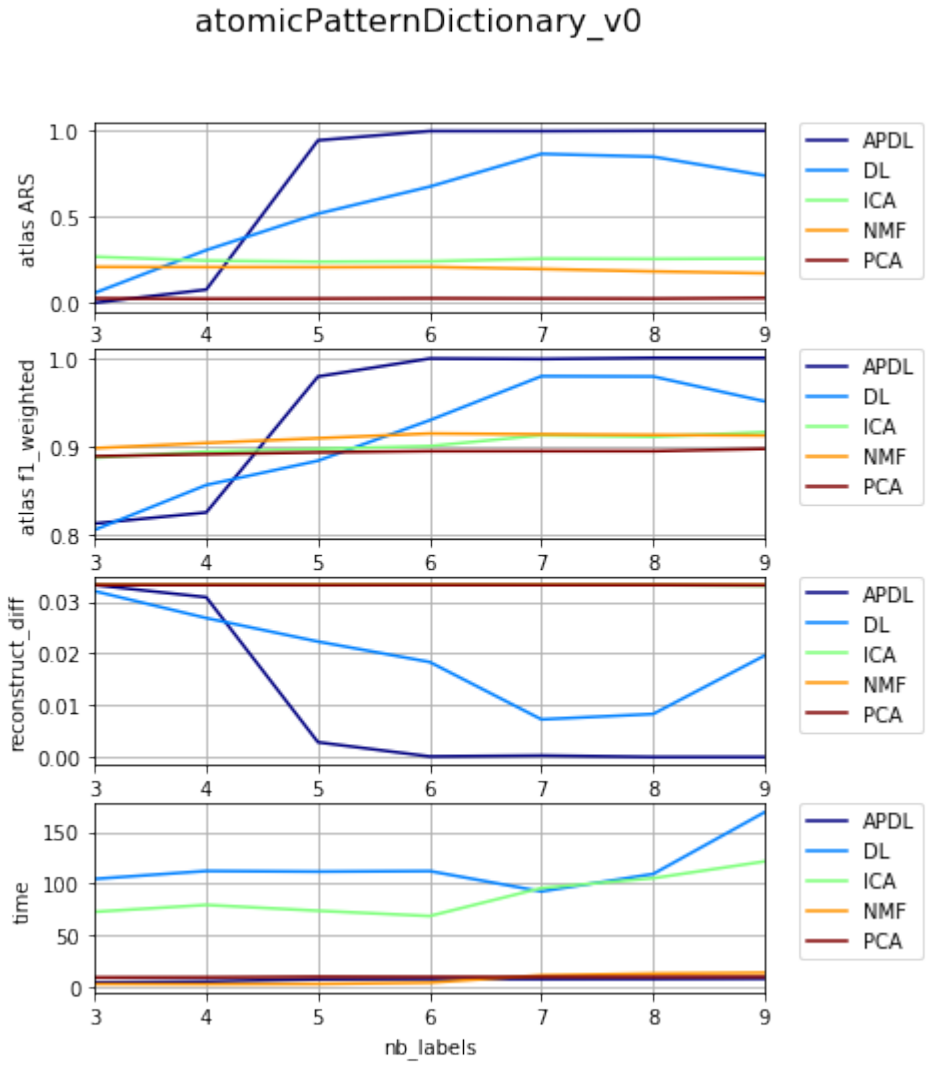
[13]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw']
df_res, dict_samples = filter_df_results_4_plotting(df_select, n_group='version', n_
↪class='method', iter_var='nb_labels', cols=LIST_GRAPHS)
print (dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'method', l_graphs=LIST_GRAPHS, iter_var=
↪'nb_labels', figsize=(6, 2))
```

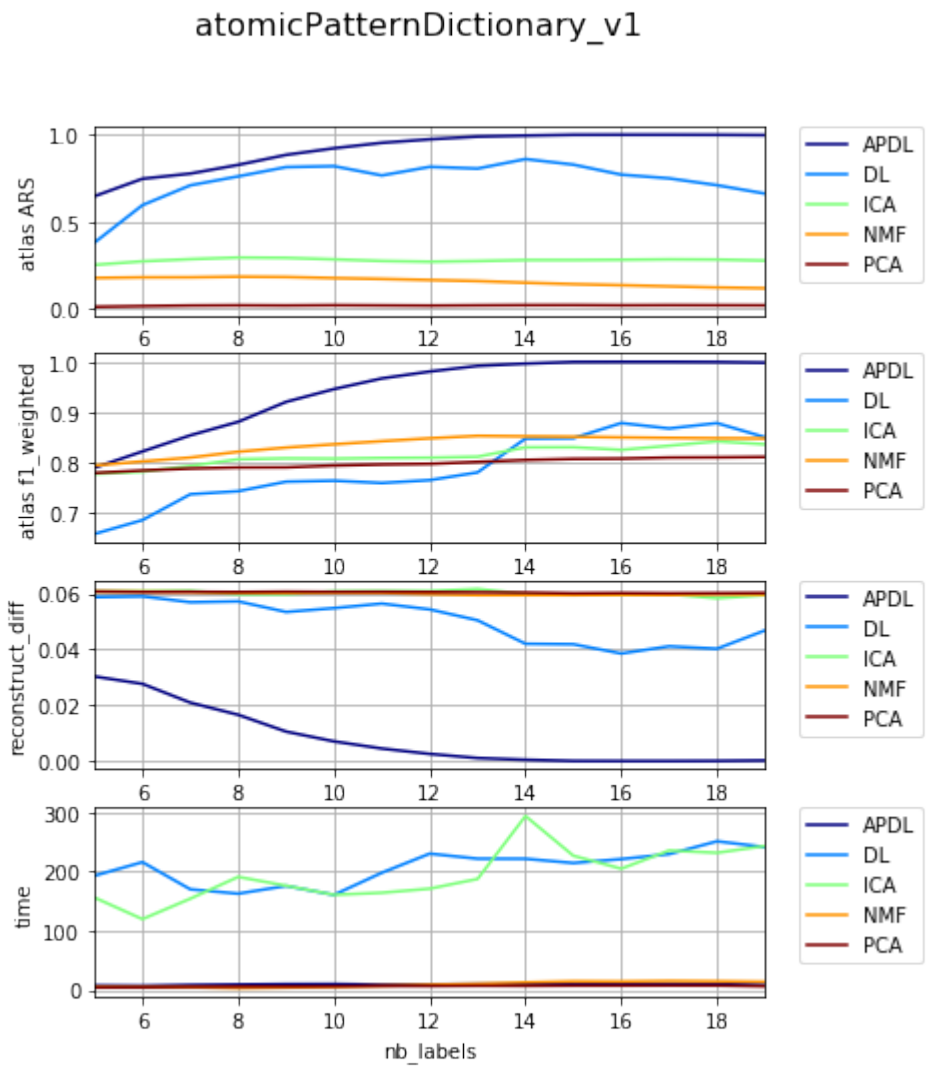
```
{'atomicPatternDictionary_00': {'NMF': [1, 2, 1], 'DL': [1, 2, 1], 'PCA': [1, 2, 1],
↪'ICA': [1, 2, 1], 'BPDL': [1, 2, 1]}, 'atomicPatternDictionary_v2': {'NMF': [78, 78,
↪ 78, 78, 78, 78, 78, 80, 80, 78, 78, 78, 78, 78, 78, 78], 'DL': [75, 75, 75, 75, 75,
↪ 75, 75, 76, 76, 75, 75, 75, 75, 75, 75, 75], 'PCA': [76, 76, 76, 76, 76, 76, 76,␣
↪78, 78, 76, 76, 76, 76, 76, 76, 76], 'ICA': [76, 76, 76, 76, 76, 76, 76, 78, 78, 76,
↪ 76, 76, 76, 76, 76, 76], 'BPDL': [75, 75, 75, 75, 75, 75, 75, 76, 76, 75, 75, 75,␣
↪75, 75, 75, 75]}, 'atomicPatternDictionary_v0': {'NMF': [78, 78, 78, 80, 80, 78,␣
↪78], 'DL': [78, 78, 78, 80, 80, 78, 78], 'PCA': [78, 78, 78, 80, 80, 78, 78], 'ICA':
↪[78, 78, 78, 80, 80, 78, 78], 'BPDL': [78, 78, 78, 80, 80, 78, 78]},
↪'atomicPatternDictionary_v1': {'NMF': [78, 78, 78, 78, 78, 78, 78, 78, 80, 80, 78,␣
↪78, 78, 78, 78], 'DL': [78, 78, 78, 78, 78, 78, 78, 78, 80, 80, 78, 78, 78, 78, 78],
↪ 'PCA': [78, 78, 78, 78, 78, 78, 78, 78, 80, 80, 78, 78, 78, 78, 78], 'ICA': [78,␣
```

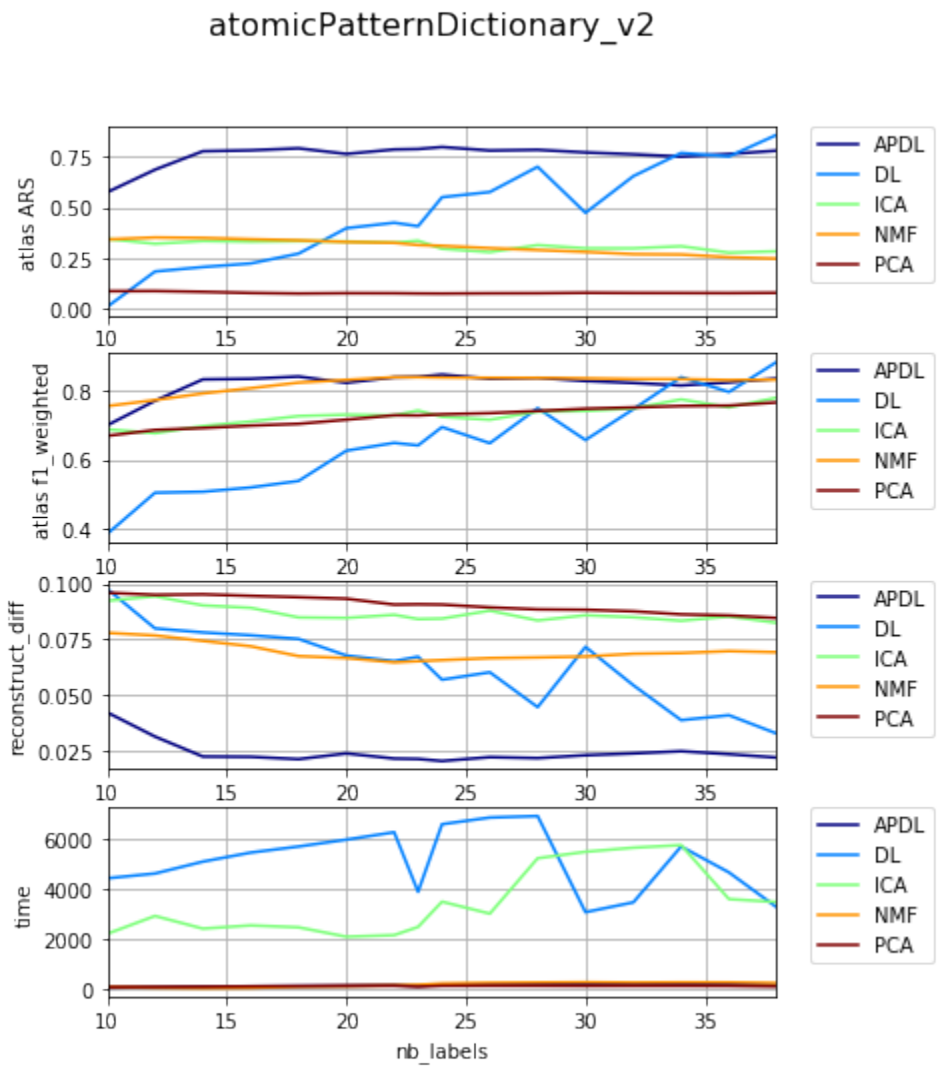
(continues on next page)

(continued from previous page)




atomicPatternDictionary_v0
atlas ARS
atlas f1_weighted
reconstruct_diff
time
nb_labels
APDL
DL
ICA
NMF
PCA

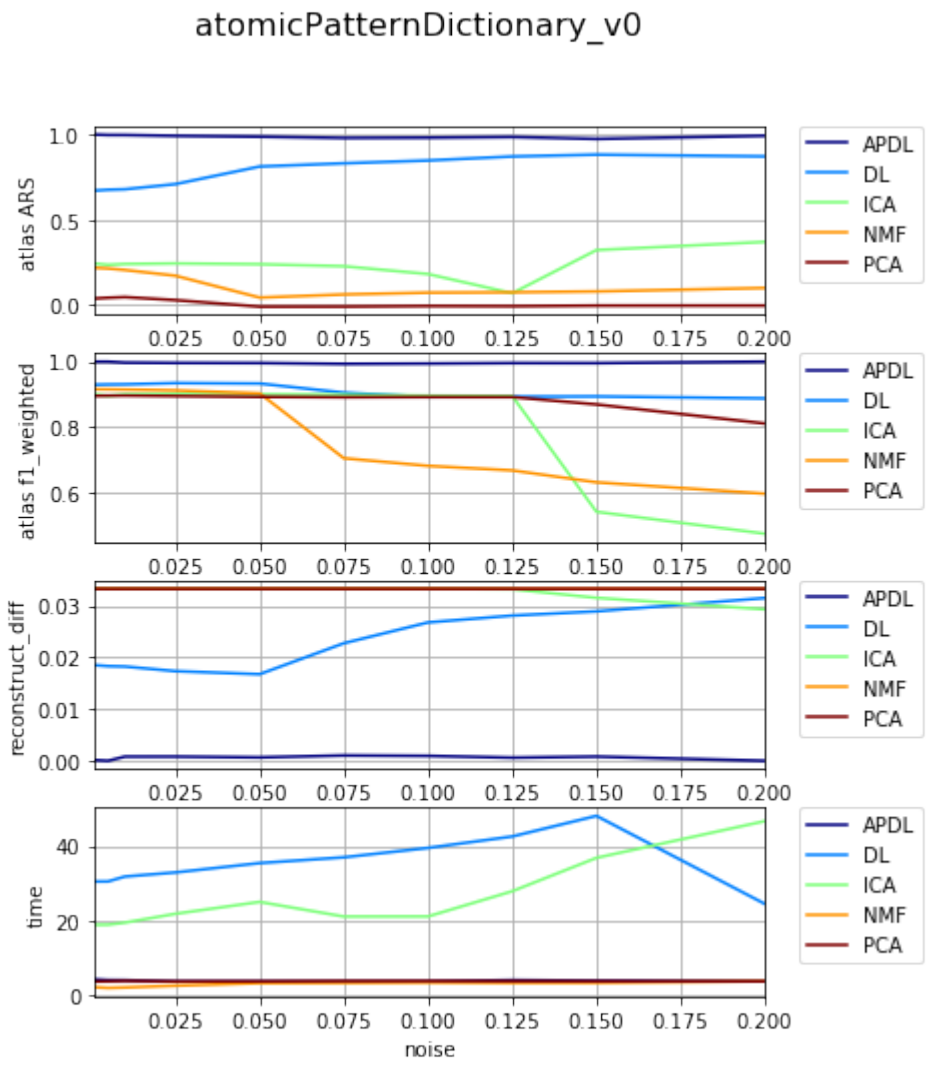

atomicPatternDictionary_v1
atlas ARS
atlas f1_weighted
reconstruct_diff
time
nb_labels
APDL
DL
ICA
NMF
PCA



Dependency on level of noise

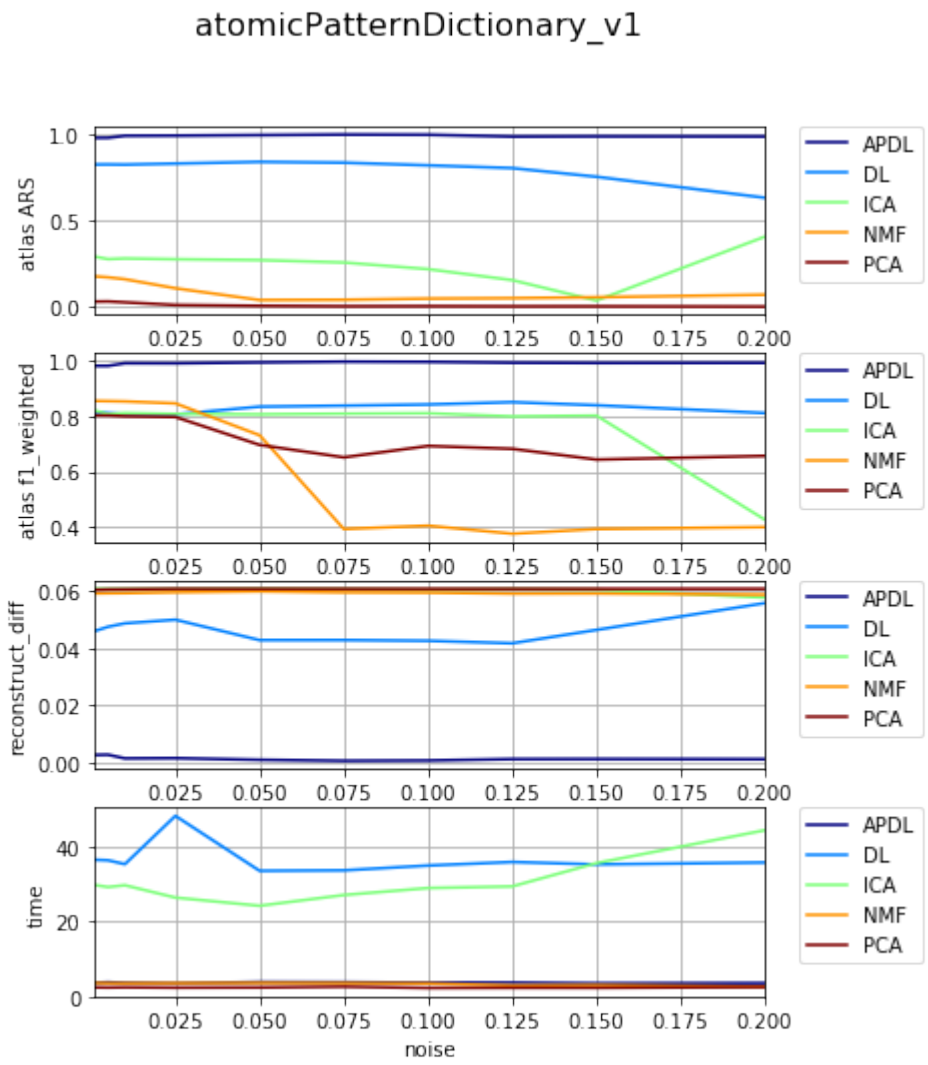
[14]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw_gauss']
df_select = df_select[df_select['version'] == 'atomicPatternDictionary_v0']
df_select = df_select[df_select['nb_labels'] == 6]
df_res, dict_samples = filter_df_results_4_plotting(df_select, n_group='version', n_
↪class='method', iter_var='noise', cols=LIST_GRAPHS)
print (dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'method', l_graphs=LIST_GRAPHS, iter_var=
↪'noise', figsize=(6, 2))
```

```
{'atomicPatternDictionary_v0': {'NMF': [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'DL':
↪ [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'PCA': [30, 30, 30, 30, 30, 30, 30, 30,
↪30, 30], 'ICA': [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'BPDL': [30, 30, 30, 30,
↪30, 30, 30, 30, 30, 30]}}
```



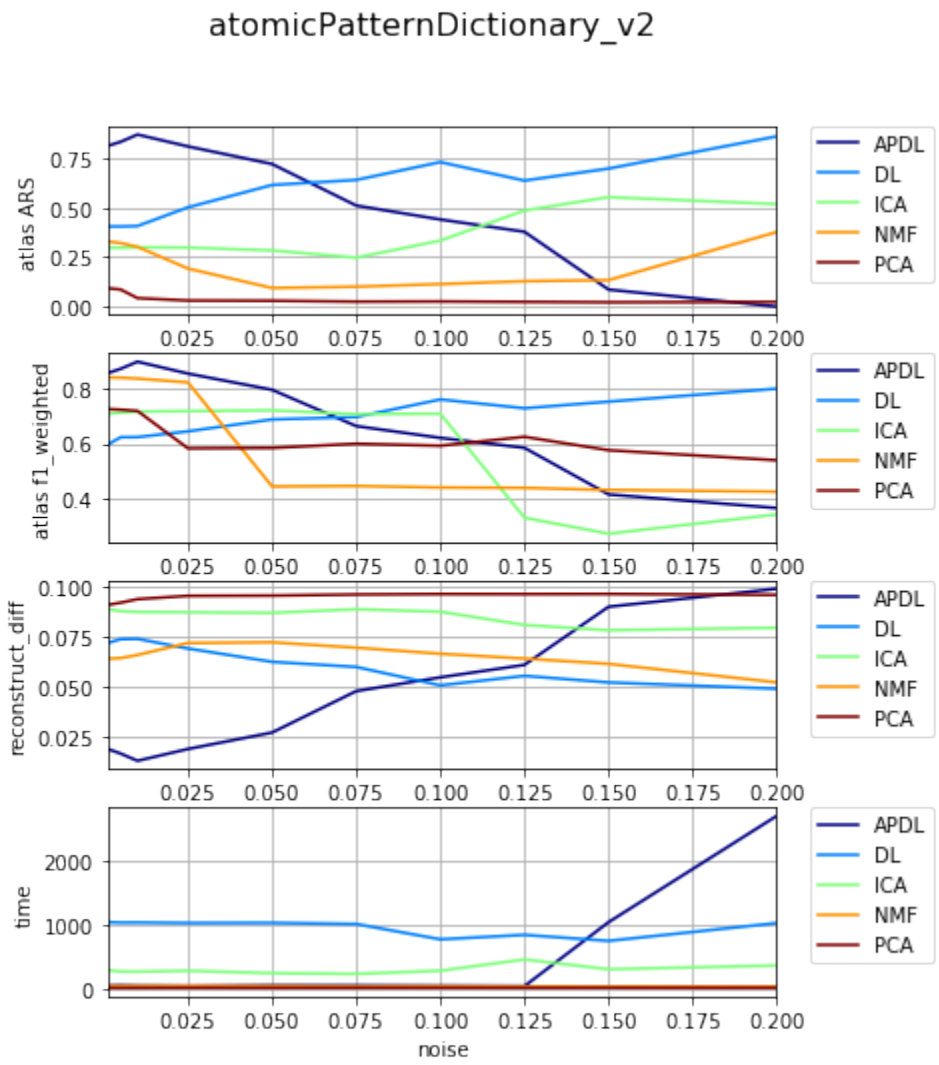
[15]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw_gauss']
df_select = df_select[df_select['version'] == 'atomicPatternDictionary_v1']
df_select = df_select[df_select['nb_labels'] == 13]
df_res, dict_samples = filter_df_results_4_plotting(df_select, n_group='version', n_
↪class='method', iter_var='noise', cols=LIST_GRAPHS)
print (dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'method', l_graphs=LIST_GRAPHS, iter_var=
↪'noise', figsize=(6, 2))
```

```
{'atomicPatternDictionary_v1': {'NMF': [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'DL':
↪ [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'PCA': [30, 30, 30, 30, 30, 30, 30, 30,␣
↪30, 30], 'ICA': [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'BPDL': [30, 30, 30, 30,␣
↪30, 30, 30, 30, 30, 30]}}
```



[17]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw_gauss']
df_select = df_select[df_select['version'] == 'atomicPatternDictionary_v2']
df_select = df_select[df_select['nb_labels'] == 23]
df_res, dict_samples = filter_df_results_4_plotting(df_select, n_group='version', n_
↪class='method', iter_var='noise', cols=LIST_GRAPHS)
print (dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'method', l_graphs=LIST_GRAPHS, iter_var=
↪'noise', figsize=(6, 2))
```

```
{'atomicPatternDictionary_v2': {'NMF': [30, 30, 30, 30, 30, 30, 30, 30, 30, 30], 'DL':
↪ [29, 29, 29, 29, 29, 29, 29, 29, 29, 29], 'PCA': [29, 29, 29, 29, 29, 29, 29, 29,␣
↪29, 29], 'ICA': [30, 30, 30, 29, 29, 29, 29, 29, 29, 29], 'BPDL': [28, 28, 28, 28,␣
↪28, 28, 28, 28, 28, 28]}}
```



[ ]:

1.4.14 Results on Synthetic datasets - Prob. images

Presenting results APDL method with respect to diffrent initilisation methods

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os, sys, glob
import pandas, numpy
from skimage import io
import matplotlib.pylab as plt
from matplotlib import gridspec
sys.path += [os.path.abspath('.'), os.path.abspath('..')]  # Add path to root
import bpdl.data_utils as tl_data
from notebooks.notebook_utils import filter_df_results_4_plotting, plot_bpdl_graph_
↪results
```

Loading data

[3]:
```
# BASE_PATH = '~/Dropbox/Documents/lab_CMP-BIA/paper_2017_drosophila_APDL/data/'
BASE_PATH = 'results'
paths_csv = [# os.path.expanduser(BASE_PATH + 'experiments_synth_APD_prob_results_NEW_
↪OVERALL.csv'),
             os.path.expanduser(os.path.join(BASE_PATH, 'experiments_synth_APDL_prob_
↪results_NEW_OVERALL.csv'))]
# p_csv = os.path.expanduser(os.path.join('results', 'experiments_synth_APD_overall.
↪csv'))
df_all = pandas.DataFrame()
for p_csv in paths_csv:
    print('%s <- %s' % (os.path.exists(p_csv), p_csv))
    df_all = df_all.append(pandas.read_csv(p_csv, index_col=None), ignore_index=True)
    print('nb lines: %i' % len(df_all))
```

```
True <- results/experiments_synth_APDL_prob_results_NEW_OVERALL.csv
nb lines: 55440
```

[4]:
```
df_all.dropna(subset=['nb_labels', 'atlas_ARS', 'reconstruct_diff'], inplace=True)
print('-> loaded DF with', len(df_all), 'items and columns:\n', df_all.columns.
↪tolist())
d_unique = {col: df_all[col].unique() for col in df_all.columns}
df_all.sort_values('nb_labels', inplace=True)
print('-> unique:', {k: len(d_unique[k]) for k in d_unique if len(d_unique[k]) > 1})
```

```
-> loaded DF with 55440 items and columns:
['atlas ARS', 'atlas accuracy', 'atlas f1_macro', 'atlas f1_weighted', 'atlas␣
↪precision_macro', 'atlas precision_weighted', 'atlas recall_macro', 'atlas recall_
↪weighted', 'atlas support_macro', 'atlas support_weighted', 'atlas_ARS', 'class',
↪'computer', 'dataset', 'files @dir', 'folders', 'gc_regul', 'gc_reinit', 'init_tp',
↪'max_iter', 'method', 'name', 'nb_workers', 'nb_labels', 'nb_runs', 'nb_samples',
↪'overlap_major', 'path_exp', 'path_in', 'path_out', 'ptn_compact', 'ptn_split',
↪'reconstruct_diff', 'time', 'tol', 'type']
-> unique: {'atlas ARS': 16230, 'atlas accuracy': 3419, 'nb_labels': 28, 'dataset':␣
↪14, 'atlas precision_macro': 5628, 'computer': 2, 'reconstruct_diff': 12217, 'atlas␣
↪recall_weighted': 15978, 'nb_runs': 2, 'nb_workers': 2, 'atlas_ARS': 16230, 'folders
↪': 4, 'atlas precision_weighted': 3419, 'files @dir': 177, 'path_exp': 3203, 'atlas␣
↪f1_weighted': 11721, 'name': 42, 'atlas recall_macro': 15966, 'time': 55408, 'path_
↪in': 3, 'init_tp': 18, 'atlas f1_macro': 11783}
```

[5]:
```
df_all = df_all[df_all['method'] == 'BPDL']
print('dropped, now lines:', len(df_all))
```

```
dropped, now lines: 55440
```

Parse name and noise level

[6]:
```
df_all['version'] = map(os.path.basename, df_all['path_in'])
print('Versions:', df_all['version'].unique().tolist())
```

```
Versions: ['atomicPatternDictionary_v0', 'atomicPatternDictionary_v1',
↪'atomicPatternDictionary_v2']
```

[7]:
```
print('Datasets: %s' % repr(df_all['dataset'].unique().tolist()))
```

```
Datasets: ['datasetFuzzy_raw', 'datasetFuzzy_defNoise', 'datasetFuzzy_noise',
↪'datasetFuzzy_deform', 'datasetFuzzy_raw_gauss-0.100', 'datasetFuzzy_raw_gauss-0.050
↪', 'datasetFuzzy_raw_gauss-0.200', 'datasetFuzzy_raw_gauss-0.150', 'datasetFuzzy_
↪raw_gauss-0.001', 'datasetFuzzy_raw_gauss-0.010', 'datasetFuzzy_raw_gauss-0.125',
↪'datasetFuzzy_raw_gauss-0.075', 'datasetFuzzy_raw_gauss-0.025', 'datasetFuzzy_raw_
↪gauss-0.005']
```

[8]:
```
noise, dataset_name = [], []
for d in df_all['dataset'].values.tolist():
    if '-' in d:
        noise.append(float(d.split('-')[-1]))
        dataset_name.append(d.split('-')[0])
    else:
        noise.append(None)
        dataset_name.append(d)
df_all['dataset'] = dataset_name
df_all['noise'] = noise
print('Datasets: %s' % repr(df_all['dataset'].unique().tolist()))
print('Noise levels: %s' % repr(df_all['noise'].unique().tolist()))
```

```
Datasets: ['datasetFuzzy_raw', 'datasetFuzzy_defNoise', 'datasetFuzzy_noise',
↪'datasetFuzzy_deform', 'datasetFuzzy_raw_gauss']
Noise levels: [nan, 0.1, 0.05, 0.2, 0.15, 0.001, 0.01, 0.125, 0.075, 0.025, 0.005]
```

[9]:
```
df_all = df_all[df_all['init_tp'].isin([
    'GT', 'GT-deform', 'random', 'random-grid', 'random-mosaic',
    'soa-init-DL', 'soa-init-ICA', 'soa-init-PCA', 'soa-init-NFM',
    'soa-tune-NFM', 'soa-tune-ICA', 'soa-tune-PCA', 'soa-tune-DL'])]
```

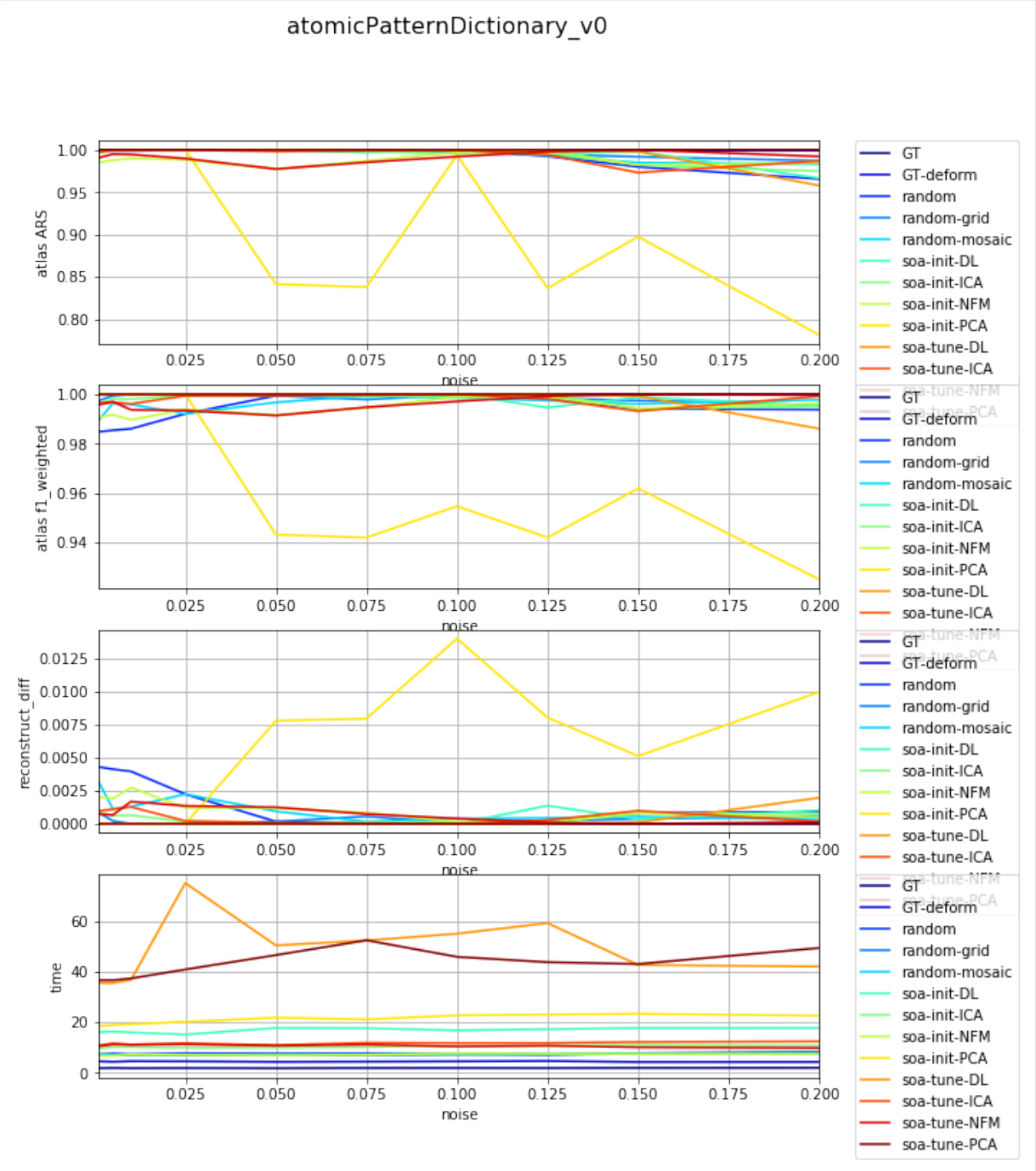
Dependency on level of noise

take out the series with various param combination

[10]:
```
# LIST_GRAPHS = ['atlas ARS', 'atlas accuracy', 'atlas f1_weighted', 'atlas precision_
↪weighted', 'atlas recall_weighted', 'reconstruct_diff', 'time']
LIST_GRAPHS = ['atlas ARS', 'atlas f1_weighted', 'reconstruct_diff', 'time']
```

[15]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw_gauss']
df_select = df_select[df_select['version'] == 'atomicPatternDictionary_v0']
df_select = df_select[df_select['nb_labels'] == 7]
df_res, dict_samples = filter_df_results_4_plotting(df_select, iter_var='noise',␣
↪cols=LIST_GRAPHS)
print(dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'init_tp', l_graphs=LIST_GRAPHS, iter_var=
↪'noise', figsize=(9, 3))
```

```
{'atomicPatternDictionary_v0': {'GT': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-
↪tune-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-init-PCA': [47, 47, 47,␣
↪47, 47, 47, 47, 47, 47, 47], 'soa-init-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 47,␣
↪47], 'random-grid': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'random': [47, 47, 47,
↪ 47, 47, 47, 47, 47, 47, 47], 'GT-deform': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47],
↪ 'soa-init-DL': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-ICA': [47, 47,␣
↪47, 47, 47, 47, 47, 47, 47, 47], 'random-mosaic': [47, 47, 47, 47, 47, 47, 47, 47,␣
↪47, 47], 'soa-init-ICA': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-PCA':␣
↪[47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-DL': [47, 47, 47, 47, 47, 47,␣
↪47, 47, 47, 47]}}
```

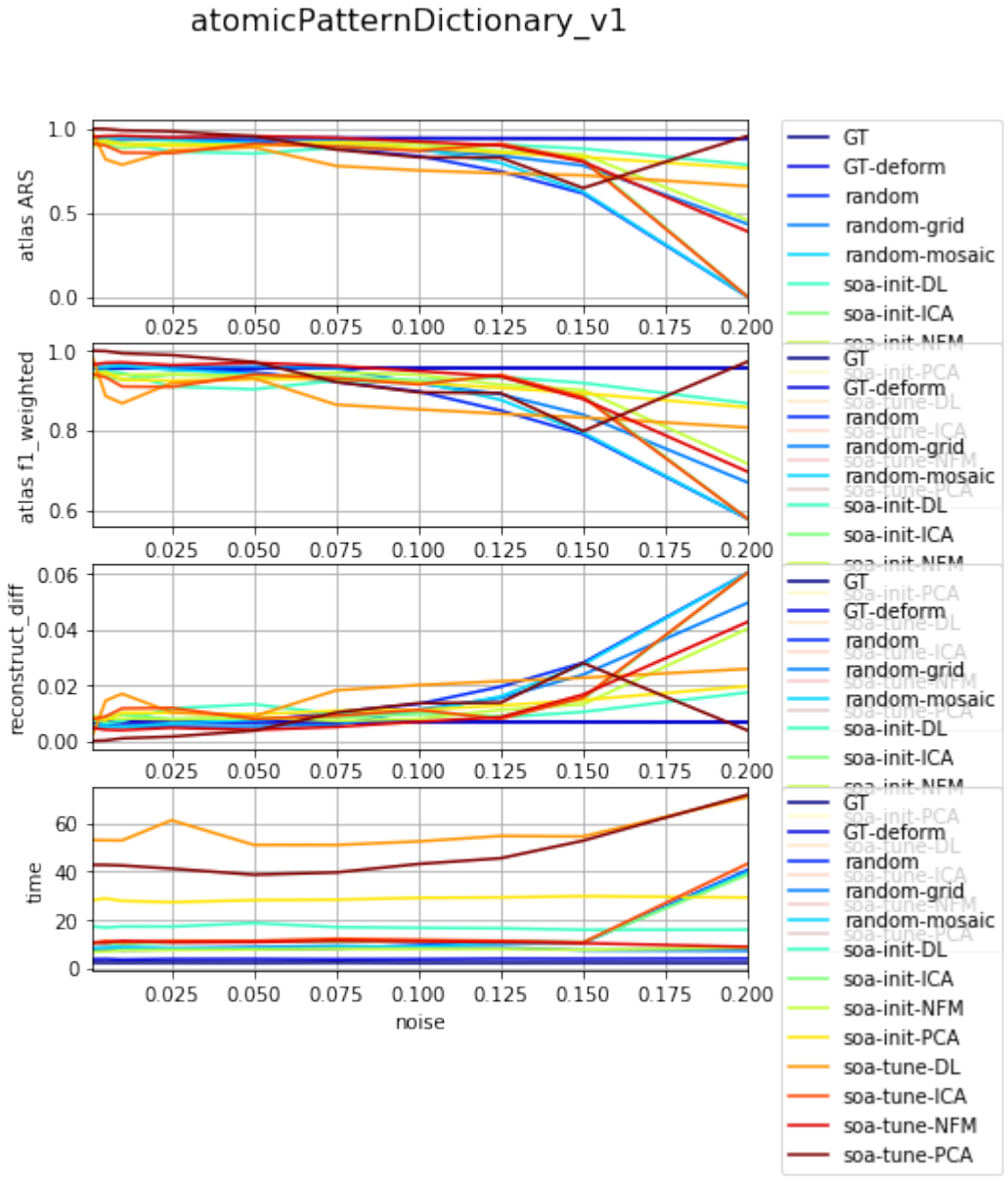


[16]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw_gauss']
df_select = df_select[df_select['version'] == 'atomicPatternDictionary_v1']
df_select = df_select[df_select['nb_labels'] == 13]
df_res, dict_samples = filter_df_results_4_plotting(df_select, iter_var='noise',␣
↪cols=LIST_GRAPHS)
print(dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'init_tp', l_graphs=LIST_GRAPHS, iter_var=
↪'noise', figsize=(9, 3))
```

(continues on next page)

(continued from previous page)

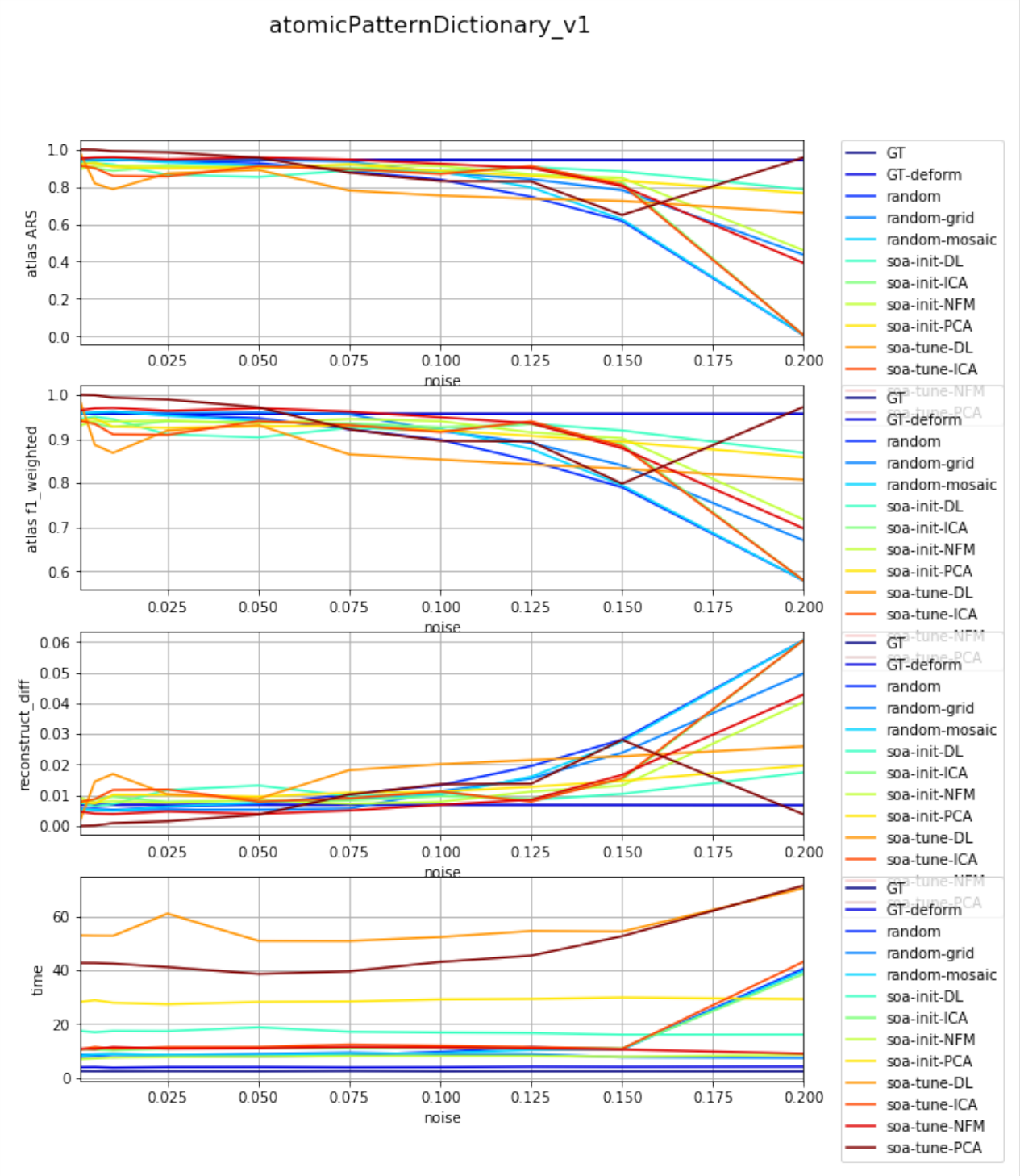

```
{'atomicPatternDictionary_v1': {'GT': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-
↪tune-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-init-PCA': [47, 47, 47,␣
↪47, 47, 47, 47, 47, 47, 47], 'soa-init-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 47,␣
↪47], 'random-grid': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'random': [47, 47, 47,
↪ 47, 47, 47, 47, 47, 47, 47], 'GT-deform': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47],
↪ 'soa-init-DL': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-ICA': [47, 47,␣
↪47, 47, 47, 47, 47, 47, 47, 47], 'random-mosaic': [47, 47, 47, 47, 47, 47, 47, 47,␣
↪47, 47], 'soa-init-ICA': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-PCA':␣
↪[47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-DL': [47, 47, 47, 47, 47, 47,␣
↪47, 47, 47, 47]}}
```


```
{'atomicPatternDictionary_v1': {'GT': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-
↪tune-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-init-PCA': [47, 47, 47,␣
↪47, 47, 47, 47, 47, 47, 47], 'soa-init-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 47,␣
↪47], 'random-grid': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'random': [47, 47, 47,
↪ 47, 47, 47, 47, 47, 47, 47], 'GT-deform': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47],
↪ 'soa-init-DL': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-ICA': [47, 47,␣
↪47, 47, 47, 47, 47, 47, 47, 47], 'random-mosaic': [47, 47, 47, 47, 47, 47, 47, 47,␣
↪47, 47], 'soa-init-ICA': [47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-PCA':␣
↪[47, 47, 47, 47, 47, 47, 47, 47, 47, 47], 'soa-tune-DL': [47, 47, 47, 47, 47, 47,␣
↪47, 47, 47, 47]}}
```


(continues on next page)

(continued from previous page)



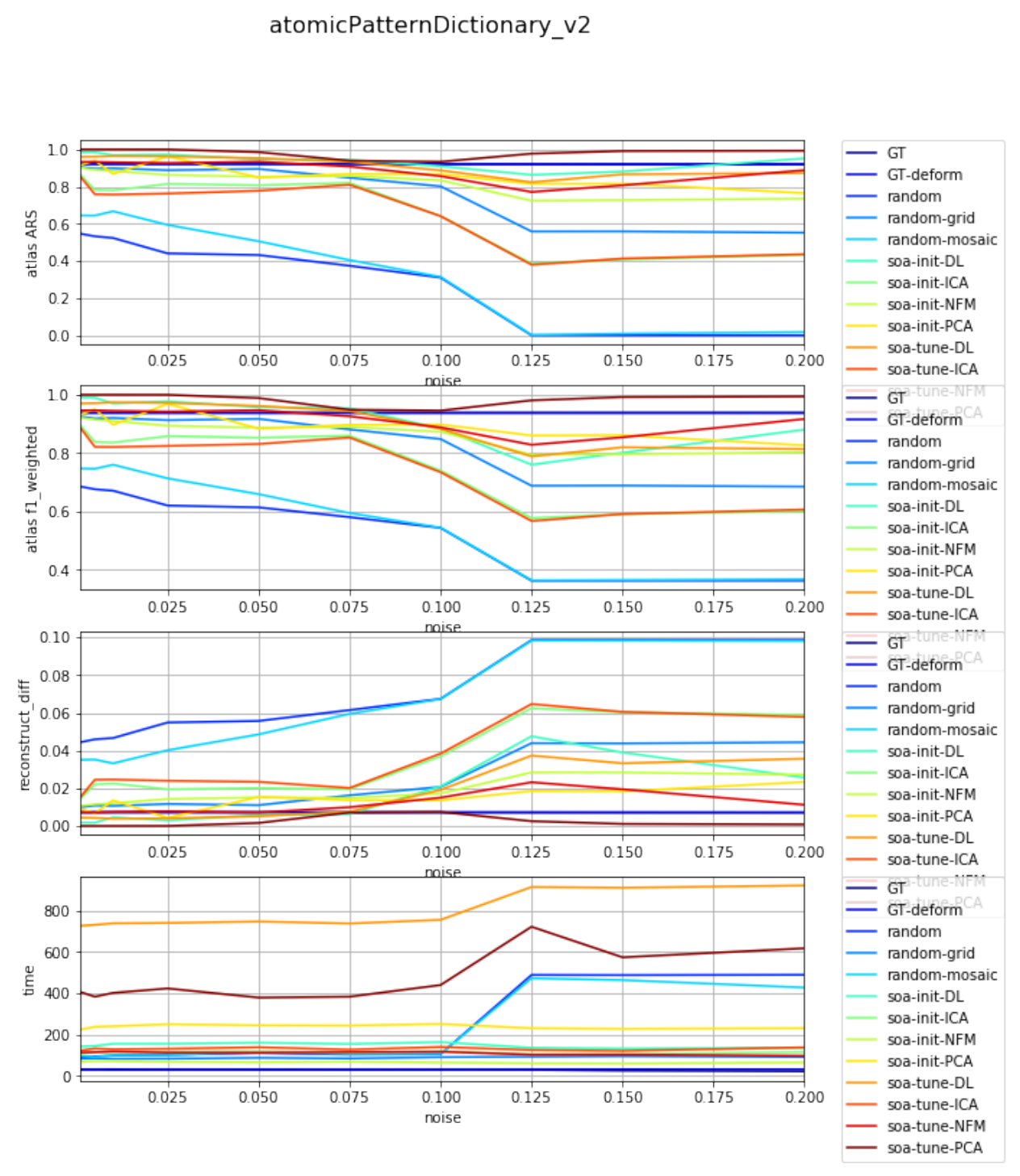
[18]:
```
df_select = df_all[df_all['dataset'] == 'datasetFuzzy_raw_gauss']
df_select = df_select[df_select['version'] == 'atomicPatternDictionary_v2']
df_select = df_select[df_select['nb_labels'] == 23]
df_res, dict_samples = filter_df_results_4_plotting(df_select, iter_var='noise',␣
↪cols=LIST_GRAPHS)
```

(continues on next page)

(continued from previous page)

```
print(dict_samples)
plot_bpdl_graph_results(df_res, 'version', 'init_tp', l_graphs=LIST_GRAPHS, iter_var=
↪'noise', figsize=(9, 3))
```

```
{'atomicPatternDictionary_v2': {'GT': [47, 47, 47, 47, 47, 47, 47, 47, 46, 46], 'soa-
↪tune-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 46, 46], 'soa-init-PCA': [47, 47, 47,␣
↪47, 47, 47, 47, 47, 46, 46], 'soa-init-NFM': [47, 47, 47, 47, 47, 47, 47, 47, 46,␣
↪46], 'random-grid': [47, 47, 47, 47, 47, 47, 47, 47, 46, 46], 'random': [47, 47, 47,
↪ 47, 47, 47, 47, 47, 46, 46], 'GT-deform': [47, 47, 47, 47, 47, 47, 47, 47, 46, 46],
↪ 'soa-init-DL': [47, 47, 47, 47, 47, 47, 47, 47, 46, 46], 'soa-tune-ICA': [47, 47,␣
↪47, 47, 47, 47, 47, 47, 47, 46, 46], 'random-mosaic': [47, 47, 47, 47, 47, 47, 47, 47,␣
↪46, 46], 'soa-init-ICA': [47, 47, 47, 47, 47, 47, 47, 47, 46, 46], 'soa-tune-PCA':␣
↪[47, 47, 47, 47, 47, 47, 47, 47, 46, 46], 'soa-tune-DL': [47, 47, 47, 47, 47, 47,␣
↪47, 47, 46, 46]}}
```


atomicPatternDictionary_v2
atlas ARS
atlas f1_weighted
reconstruct_diff
time
noise
GT
GT-deform
random
random-grid
random-mosaic
soa-init-DL
soa-init-ICA
soa-init-NFM
soa-init-PCA
soa-tune-DL
soa-tune-ICA
soa-tune-NFM
soa-tune-PCA

[ ]:

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

THREE

BPDL - BINARY PATTERN DICTIONARY LEARNING

The package contain Binary pattern Dictionary Learning (BPDL) which is image processing toolfor unsupervised pattern extraction and atlas estimation. Moreover the project/repositorycontains comparisons with State-of-the-Art decomposition methods applied to image domain.The package also includes useful tools for dataset handling and around real microscopy images.

3.1 Main features

- implementation of BPDL package
- using fuzzy segmentation as inputs
- deformation via daemon registration
- experimental setting & synthetic dataset
- comparison with NMF, SPCS, ICA, CanICA, MSDL, etc.
- visualisations and notebook samples

3.2 References

Borovec J., Kybic J. (2016) Binary Pattern Dictionary Learning for Gene Expression Representationin Drosophila Imaginal Discs. In: Computer Vision - ACCV 2016 Workshops. Lecture Notes in ComputerScience, vol 10117, Springer. DOI: 10.1007/978-3-319-54427-4_40.

PYTHON MODULE INDEX

b

e

INDEX