fMRI data for visual image reconstruction (beta version)

Yoichi Miyawaki (original: 2009/08/05, last updated: 2012/05/23)
E-mail: yoichi.miyawaki@uec.ac.jp

This document describes MATLAB codes and the data structure used for visual image reconstruction (Miyawaki, et al. (2008)). The data corresponds to subject S1 used in Miyawaki et al. (2008). The data is prepared for MATLAB v.6 compatible format.

Previously we released only the data for visual image reconstruction, but this version also contains MATLAB codes to process the data. To run the codes, additional library programs are necessary. To keep compatibility with those library programs, the data structure has been largely modified. This document explains all the necessary processes to make a new library-compatible MAT file from ANALYZE files, train local decoders, learn combination coefficients of local decoders, reconstruct visual images, and display the reconstructed images.

NOTE: We recommend testing the following analyses under Linux environment since we have not tested other environment such as Windows and Mac.

Table of contents

1 Download of libraries

In addition to the main MATLAB codes for visual reconstruction, the following packages are necessary.

- Brain Decoder Toolbox

  Japanese page http://www.cns.atr.jp/dni/download/brain-decoder-toolbox/

  English page http://www.cns.atr.jp/dni/en/downloads/brain-decoder-toolbox/

  Note: use version 1.2.2

- SPM5

  http://www.fil.ion.ucl.ac.uk/spm/software/spm5/

- Sparse Logistic Regression toolbox

  http://www.cns.atr.jp/~oyamashi/SLR_WEB.html

  Note: use version 1.2.1alpha

After you download the above packages, unzip and place all library directories as Figure 1. [visReconRoot] can be any name, which corresponds to your root directory to work with this. Directories under `main/` is made by unzipping `Data_code_public.zip` file. Make `lib/` directory at the same level as `main/` and place the above three additional libraries under the `lib/` directory.

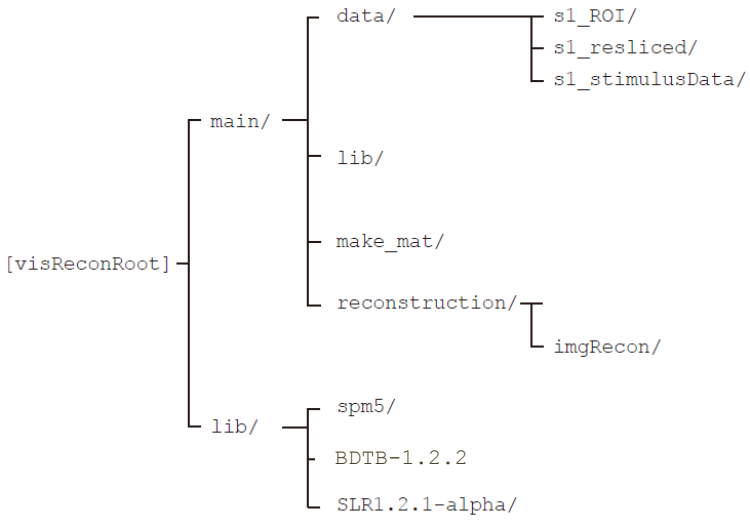

Figure 1: Directory structure for demo of visual image reconstruction.

2 Path setup

The main MATLAB codes for visual image reconstruction are placed in `make_mat/`, `reconstruction/`, and `reconstruction/imgRecon/` directories. To run these programs properly, the following directories including subdirectories should be added to search path:

- `[visReconRoot]/lib/spm5/`
- `[visReconRoot]/lib/BDTB-1.2.2/`
- `[visReconRoot]/lib/SLR1.2.1-alpha/`
- `[visReconRoot]/main/lib/`

At the beginning of each m-file, `addpath` and `genpath` commands are used to include the directories to search path automatically. However, if the programs do not work properly, please check these directories are included to search path.

3 Making a MAT file from ANALYZE files and experimental protocol files

Visual image reconstruction needs a data set in which fMRI signal patterns and visual images are associated properly. The first step is to create such a data set as a MAT file using fMRI data with experimental protocol data that preserves information about presented visual images.

Here we use fMRI data in ANALYZE format, which were already motion-corrected and resliced[1]. The fMRI data are located in `data/s1_reliced/`. The experimental protocol data are located in `data/s1_stimulusData/`.

The MATLAB program that makes a MAT file from these data is `make_fmri_mat_visRecon.m`, which is located in `make_mat/`. To run this program, move to `make_mat/` directory and type like this:

```
>> make_fmri_mat_visRecon
```

This program automatically loads fMRI data and experimental protocol information, and shapes the data so that the data format is compatible with our decoding toolbox library

[1] If you are interested in fMRI data before the preprocessing, please contact us.

"BrainDecoderToolbox (BDTB)." If all processing completes properly, you will find a new directory `data/s1_fmri_mat/`. In that directory,

`s1_fmri_roi-1to2mm_Th1_fromAna_s1071119ROI_resol10_v6.mat`

will be created.

If you use 64bit-Linux version MATLAB and encounter an error "`bad image handle dimensions`," please visit this web site

`http://en.wikibooks.org/wiki/SPM/Installation_on_64bit_Linux`

and then fix compatibility problems of SPM MEX files.

Loading the MAT file, you will find the following data fields:

```
>> load s1_fmri_roi-1to2mm_Th1_fromAna_s1071119ROI_resol10_v6.mat
>> D =

        label: [4460x101 double]
    label_type: {1x101 cell}
     label_def: {1x101 cell}
       design: [4460x3 double]
   design_type: {'run'  'block'  'figure_type'}
          roi: [64x6046 double]
     roi_name: {1x64 cell}
          xyz: [3x6046 double]
         stat: [64x6046 double]
     stat_type: {'tval'}
         data: [4460x6046 double]
```

All the data fields have the same meaning as defined in BDTB except `D.label`, `D.stat`, and `D.stat_type`. `D.label` contains integer labels as an identifier for each visual image type (1st column) and binary labels (2nd to 101st columns) representing whether patches in presented images are high contrast (= 1) or uniform gray (= 0). `D.stat` and `D.stat_type` fields are prepared to keep compatibility with BDTB but their contents have no meaning,

not used in further analyses.

For this subject (S1), we performed 20 runs in the random image session and 12 runs in the figure image sessions. Thus the data contains a total of 20 + 12 = 32 runs of fMRI data and information about presented images. The order of presented images is as follows: runs #1 - #20, random images; runs #21 - #24, geometrical figures; #25 - #28, alphabet letters (layout 1); #29 - #32, alphabet letters (layout 2). Although we presented results of figure image reconstructions using runs #21 - #24 of geometrical figures and runs #29 - #32 of alphabet letters (layout 2) in Miyawaki et al. (2008), we also tested slightly different layout of alphabet letters, that is, "runs #21 - #24, alphabet letters (layout 1)". This data set contains a full set of these data.

For example, if you want to see a visual image presented as the first image of the first run of the random image session, type as follows:

```
>> s = reshape(D.label(11, 2:end),10,10); %% first 10 samples are skipped since
they are in the rest period (i.e., homogeneous gray images)
>> imagesc(s); colormap(gray); axis image;
```

The, you will see an image like Figure 2.

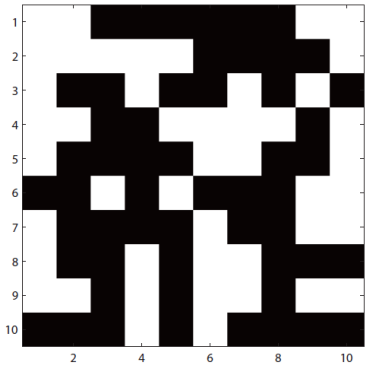

Figure 2: An example of a random image presented in the random image session (run #1, block #1).

For the other example, if you want to see a visual image presented as the first image of the first run of the figure image session, type as follows:

```
>> runIDs = D.design(:, ismember(D.design_type, 'run'));
>> offsetOfFigureRun = min(find(runIDs == 21));
```

```
>> s = reshape(D.label(offsetOfFigureRun + 10, 2:end),10,10); %% first 10 samples
are skipped since they are in the rest period (i.e., homogeneous gray images)
>> imagesc(s); colormap(gray); axis image;
```

The, you will see an image like Figure 3.

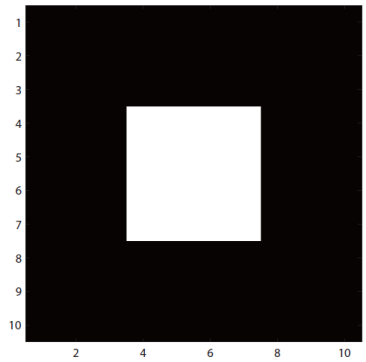

Figure 3: An example of a figure image presented in the figure image session (run #21, block #1).

4 Training of local decoders

In the next step, local decoders are trained using the MAT file. To determine combination coefficients of local decoders (see the next section and page 928 of Miyawaki et al. (2008)), ten-fold cross-validation should be performed. In addition, all runs of the random image session should be used as a training data set of local decoders for reconstructing independent images presented in the figure image session. Thus a total of 10 sets of training using 18 runs out of 20 runs ("leave1" mode) and 1 set of training using all 20 runs out of 20 runs ("leave0" mode) should be performed for reconstruction of visual images presented in the figure image session.

Furthermore, in Miyawaki et al. (2008), four types of spatial scales were used for basis functions. Thus, the local decoders should be prepared for these four scales separately.

`trainLocalDecoder.m` function is designed to perform local decoder training with the above two training modes and with four spatial scales of basis functions for a specified ROI. `trainLocalDecoder.m` is located in `reconstruction/` directory. Examples to call this function are written as BATCH file, which is `trainLocalDecoder_BATCH.m`, located in the same directory. In the `trainLocalDecoder_BATCH.m` file, you will see `trainLocalDecoder` function is called with various options like,

```
trainLocalDecoder('s1_s1071119',{'1x1'},'V1V2','leave0',0);
trainLocalDecoder('s1_s1071119',{'1x1'},'V1V2','leave1',0);

trainLocalDecoder('s1_s1071119',{'1x2'},'V1V2','leave0',0);
trainLocalDecoder('s1_s1071119',{'1x2'},'V1V2','leave1',0);
…
```

The first options is constant (not necessary to modify), describing subject ID for this data set. The second option describes the spatial scales of the basis function for which local decoders are trained. You can specify individual spatial scale separately as shown above, and you can also specify a list of spatial scales as cell array like,

```
trainLocalDecoder('s1_s1071119',{'1x1','1x2','2x2','2x2'},'V1V2','leave0',0);
```

In this case, all the scales are trained sequentially with this one command. The fourth option represents ROI name. In this demonstration, `'V1V2'` (a combination of V1 and V2), `'V1'` only, `'V2'` only, `'V3'` (a combination of V3 and VP), and `'AllArea'` (a combination of V1 – V4) are possible ROI names. `'AllArea'` needs a huge memory space to train decoders and a very long time to complete the training, so be careful to run the program with that option. The firth option indicates the training mode as described above.

To run the training of the local decoders, in `reconstruction/` directory, execute BATCH file as,

```
>> trainLocalDecoder_BATCH
```

or just directly call `trainLocalDecoder` while specifying options.

If you did not modify anything on `trainLocalDecoder_BATCH.m` file, the command will perform training of the local decoders for the spatial scale for `'1x1'` with `'V1V2'` ROI for `'leave0'` and `'leave1'` mode.

The model used in this demonstration is sparse logistic regression, the same one used in Miyawaki et al. (2008).

The trained results, including the model parameters, are automatically saved in

`de_s1_V1V2_Ecc1to11_baseByRestPre_smlr_s1071119ROI_resol10_leave0/`

or

`de_s1_V1V2_Ecc1to11_baseByRestPre_smlr_s1071119ROI_resol10_leave1/`

directory under `reconstruction/` directory. Within these directories, you will find subdirectories that are named with the spatial scales that you specified when you call `trainLocalDecoder`. Further, in each spatial scale subdirectory, you will find multiple directories named as `label001/`, `label002/`, … They correspond to basis identifier numbers. The basis identifier number begins with the left top and ends at the right bottom of the presented image (Figure 4). In case of 1x2, 2x1, and 2x2 spatial scales, there are one-patch overlaps with adjacent bases (Figure 4). Thus a total number of basis functions are 100 for 1x1 and 90 for 1x2 and 2x1, and 81 for 2x2 spatial scales.

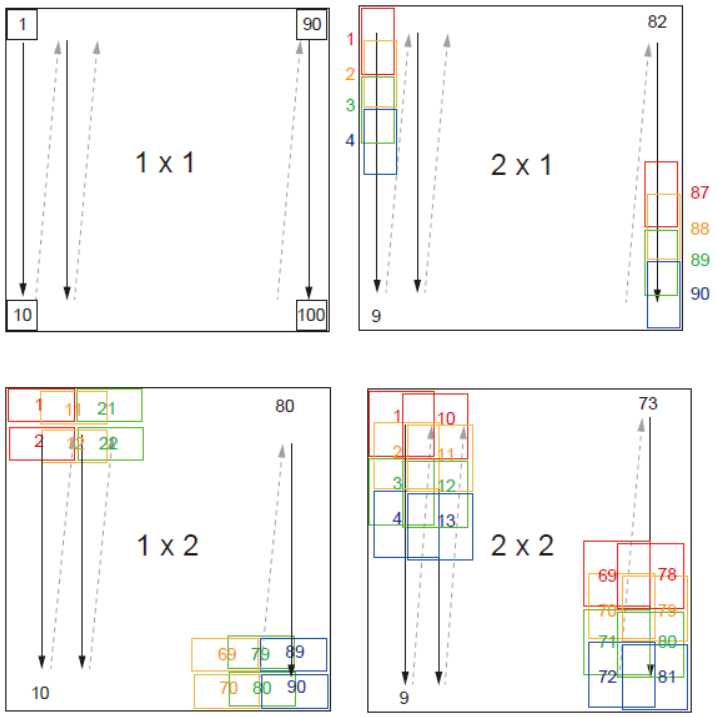

Figure 4: Assignment of basis function locations.

Thus, if you complete training of the local decoders, you will find directory structure like this:

```
>> cd de_s1_V1V2_Ecc1to11_baseByRestPre_smlr_s1071119ROI_resol10_leave0
>> ls -F
1x1/  1x2/
>> cd 1x1/
>> ls -F
label001/  label002/  label003/  ...
... label098/  label099/  label100/
>> cd ../1x2
>> ls -F
label001/  label002/  label003/  ...
... label088/  label089/  label090/
```

In each image basis directory, you will find MAT files that contain trained model parameters. In case of 'label001' for '1x1' scale with 'leave0' mode, the file name is as follows.

```
>> cd label001/
>> ls
1x1_label001_train1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
1x1_label001_train1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20_RUNNING_INFO.mat
```

The sequence of numbers in the file name represents run identifier numbers for the random image session with which the local decoder is trained. In this case, all the twenty runs are used to train the decoder since the training mode is 'leave0'. In case of 'leave1' mode, the file name is as follows.

```
>> pwd

ans =
[visReconRoot]/reconstruction/de_s1_V1V2_Ecc1to11_baseByRestPre_smlr_s1071119ROI_resol10
_leave1/1x1/label001/

>> cd label001/
>> ls -t
1x1_label001_train3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
1x1_label001_train3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20_RUNNING_INFO.mat
1x1_label001_train1-2-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
```

```
1x1_label001_train1-2-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20_RUNNING_INFO.mat
1x1_label001_train1-2-3-4-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
1x1_label001_train1-2-3-4-7-8-9-10-11-12-13-14-15-16-17-18-19-20_RUNNING_INFO.mat
...
1x1_label001_train1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18.mat
1x1_label001_train1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18_RUNNING_INFO.mat
```

The `*_RUNNING_INFO.mat` files are auxiliary files, which are used for parallel processing using multiple machines (or CPU cores). Each `RUNNING_INFO` file is created when calculation of the corresponding local decoder starts. If you run the same program (i.e., trainLocalDecoder_BATCH) on the same machine on a different CPU core, the second program first checks existence of `RUNNING_INFO` file for the corresponding local decoder. If there is already the corresponding `RUNNING_INFO` file, then the program automatically skips the calculation and goes to calculation for the next local decoder training. If you further issue the third command, it searches a local decoder that has not been trained by other CPU cores. In this way, tasks of the training of local decoders can be virtually parallel-processed with multiple CPU cores. If you have server-client systems, in which a file server is shared with multiple client machines, you can perform the same thing. The larger number of machines you use, the faster the training completes.

5 Reconstruction of visual images

5.1 Extraction of parameters of local decoders

If you complete the local decoder training successfully, the results for the trained model parameters are saved in multiple directories as described above. For the purpose of convenience for further analyses, here the model parameters are extracted from the multiple directories and are integrated into another intermediate files. To perform this operation, run the following code:

```
>> extractDecoder_BATCH
```

in `main/reconstruction/` directory. In `extractDecoder_BATCH.m`, `extractDecoder` function is called with options for spatial scales of basis functions, ROI names, and training modes. `extractDecoder` automatically creates a new directory, whose name ends with `*_decoder`, in which the integrated files are created. One integrated file contains trained

model parameters for all image bases for one training data set. Thus, a total of 10 MAT files will be created for `'leave1'` mode, whereas only 1 MAT file will be created for `'leave0'` mode. For example, suppose that you complete this extraction and integration process successfully for `'1x1'` image basis, `'V1V2'` ROI, and `'leave1'` mode, you will find the following 10 files in the corresponding directories like this:

```
>> pwd

ans =

[visReconRoot]/main/reconstruction

>> ls -F de_s1_V1V2_Ecc1to11_baseByRestPre_smlr_s1071119ROI_resol10_leave1_decoder
1x1_1-2-3-4-5-6-7-8-11-12-13-14-15-16-17-18-19-20.mat   1x1_1-2-3-4-5-6-7-8-9-10-13-14-15-16-17-18-19-20.mat
1x1_1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18.mat    1x1_1-2-3-4-5-6-9-10-11-12-13-14-15-16-17-18-19-20.mat
1x1_1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-19-20.mat    1x1_1-2-3-4-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
1x1_1-2-3-4-5-6-7-8-9-10-11-12-13-14-17-18-19-20.mat    1x1_1-2-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
1x1_1-2-3-4-5-6-7-8-9-10-11-12-15-16-17-18-19-20.mat    1x1_3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20.mat
```

The file name convention is [basis type]_[string of run identifier numbers used for the training].mat.

5.2 Learning of combination coefficients

Using the trained local decoders, combination coefficients can be optimized so that image reconstruction error is minimized for the training data set. Now we have 10 sets of local decoder parameters for different training data set, consisting of 18 runs out of 20 runs of the random image session. For each training data set two runs are reserved for testing reconstruction performance, and thus 10 sets of reconstruction error can be calculated. Specifically, combination coefficients are determined so as to minimize a sum of reconstruction errors of these 10 training-test data sets.

`calCombCoef_figRecon_smlr` is a function to perform this optimization, which is located in `reconstruction/imgRecon/` directory. `calCombCoef_figRecon_smlr` needs the following three options: scales of basis functions, optimization mode, and ROI name.

The scales of basis functions specify which scales of basis function are included to

reconstruct visual images. If you set it as `'1x1'`, only 1x1 basis functions are used. If you set it as `'1x1_1x2'`, 1x1 and 1x2 basis functions are combined. If you set it as `'1x1_1x2_2x1_2x2'`, all basis functions are combined. Here the basis scales should be concatenated with `'_'` mark.

The optimization mode specifies an objective function of error minimization. In the current demonstration, only two mode is enabled, `'no_opt'` or `'errFuncImageNonNegCon'`. `'no_opt'` can be used for only 1x1 basis functions, performing no optimization for the combination coefficients (i.e., the combination coefficients for all the basis functions are 1). `'errFuncImageNonNegCon'` minimizes square errors under non-negative constraints on the combination coefficients, using `fmincon` in Optimization toolbox function of MATLAB.

The ROI name specifies ROI name you used to train the local decoders. It should match what you specified when training the local decoders.

You will find several examples of optimization of the combination coefficients in `calCombCoef_figRecon_smlr_BATCH.m`. As an initial setup, `calCombCoef_figRecon_smlr` is called with `'1x1'`, `'errFuncImageNonNegCon'`, and `'V1V2'`, and other examples are commented out. So try this example first by just typing as,

```
>> calCombCoef_figRecon_smlr_BATCH
```

Then, the program will load trained decoders' parameters and minimization processes automatically start. If the minimization processes complete, a new directory named `result/` is automatically created. In the `result/` directory, hierarchical subdirectory structure is also created such as `result/s1/V1V2/smlr/`. At the deepest level of the directory, you will find MAT files containing optimization results. You will also find `*_toSkip.mat` file, which is paired with MAT file containing the optimization results. These files are created to implement virtually-parallel processing using multiple CPU cores (or machines) as described in section 4. Thus you can use a single BATCH file containing multiple commands to compute them in parallel using multiple CPU cores (or machines).

5.3 Reconstruction of visual images

After the combination coefficients are optimized, you can apply the parameter set and reconstruct visual images presented in the figure image session. To reconstruct the figure images, local decoders can be trained using all the data in the random image session because

they are independent. This decoder is trained by `'leave0'` mode, which uses all runs from the random image session (see also section 4). Visual image reconstruction is performed by applying these local decoders to the fMRI data in the figure image session and combining their predictions with the optimized combination coefficients.

`figRecon_smlr` is a function to perform the process of visual image reconstruction, which is located in `reconstruction/imgRecon/` directory. `figRecon_smlr` needs the following three options: scales of basis functions, optimization mode, and ROI name. These options can be specified in a similar way to `calCombCoef_figRecon_smlr` function. You will find several examples calling `figRecon_smlr` with different options in `figRecon_smlr_BATCH.m`. As an initial setup, `figRecon_smlr` is called with `'1x1'`, `'errFuncImageNonNegCon'`, and `'V1V2'`, and other examples are commented out. So try this example first by just typing as,

```
>> figRecon_smlr_BATCH
```

If the reconstruction process completes successfully, you will find MAT file containing results of visual image reconstruction in the same directory where the combination coefficient are saved (e.g., `result/s1/V1V2/smlr/`).

5.4 Presenting reconstructed images

The final step is to display the reconstructed visual images. `showReconImg` is a function to show reconstructed visual images as MATLAB figures with quantitative evaluation of reconstruction performance (mean square error, correlation, and p-value of correlation). The function is located in `reconstruction/imgRecon/` directory. `showReconImg` needs the following three options: scales of basis functions, optimization mode, and ROI name. These options can be specified in a similar way to `calCombCoef_figRecon_smlr` and `figRecon_smlr` function. You will find several examples calling `showReconImg` with different options in `showReconImg_BATCH.m`. As an initial setup, `showReconImg` is called with `'1x1_1x2_2x1_2x2'`, `'errFuncImageNonNegCon'`, and `'V1V2'`, and other examples are commented out. So try this example first by just typing as,

```
>> showReconImg_BATCH
```

Then, you fill find the results on the figure panels like Figure 5. The top row corresponds to the presented images. The second row corresponds to the reconstructed images. The third to

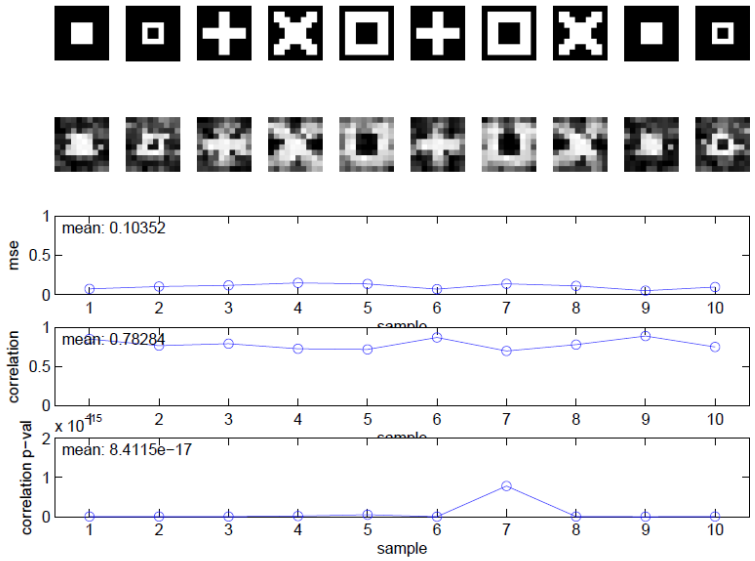

Figure 5: Reconstructed visual images and performance evaluation (run #21).

the fifth rows correspond to the mean square error, correlation, and p-value of correlation for each image, respectively. In each figure panel, results from one run will be shown. Thus the figure panel will appear as much as the number of the runs in the figure image session.

6 Copyright