**Deep learning on *cis*-decoding with cistrome datasets**

**STRUCTURE**

(i). Training of TFs recognition patterns from cistrome datasets

[on directory “**1stDL\_predict\_CREs**”]

(ii). Deep learning for binary classification of expression patterns

[on directory “**2ndDL\_predict\_expression**”]

(iii). Feature visualization

[on directory “**Backpropagation**”]

**(i). Training of TFs recognition patterns from DAP-Seq**

1. Extract fragments including significant peak from DAP-Seq (NOTE: the length should be identical), to convert into simple text files.

If you target 31-bp tiles, save as .txt like,

ATGCGTGCGTGCGTGGCTGCAATGTGCAAAT

GGGTACTAGCTTGTATATAGCAAATATAGCA

2. Training/validation by a fully-connected model

**$ python FC-cistrome-training.py [-p] [-n] [-o] [-e] [-l]**

"-p", help="File path to positive DNAs"

"-n", help="File path to negative DNAs"

"-o", help="output prefix"

"-e", help="epoch numbers"

"-l", help="DNA length"

* Output “.h5 file” with ROC-AUC data.

3. Detection of prediction (of each TF biding) in the target promoter sequences in bin sliding-windows.

Target promoter sequences should be in **fasta** format.

**$ python MultiSeq\_CREs\_prediction\_walking.py [-f] [-m] [-w] [-b] [-o]**

"-f", help="File path to fasta"

"-m", help="File path to HDF5 or H5 model"

"-w", help="walk bp size"

"-b", help="bin bp size" **(should be same as the tile set above)**

"-o", help="output file name"

Output “wOTU\_XXX(out)” with each target info as OTUs.

4. Conversion to binary CRE arrays in bins

$ python BinIntg2BinaryArray.py [-i] [-b] [-t] [-o]

"-i", help="File path to input"

"-b", help="bin size"

"-t", help="confidence threshold (0-1)"

"-o", help="output file name"

With 2-bp walking size in the previous step (MultiSeq\_CREs\_prediction\_walking.py), “-b 25” produces an array with 50-bp bins.

If input file is like

geneA 0.1 0.4 0.9 0.1 0.1 0.0 0.2 1.0 0.9

**-b 3 -t 0.8 produces**

geneA 1 0 1

**(ii). Deep learning for binary classification of expression patterns**

Use “**2ndDL\_predict\_expression**” directory.

1. Move all of the data including transitions of the predicted TF-binding sites for all target promoters, into “raw\_data” (as default).

2. **$ python make\_dataset.py [--length] [--raw\_data\_root] [--dataset\_root]**

--length, default='20', help='data length or bin numbers (int)'

--raw\_data\_root, default='raw\_data', help='path to gene data root.'

--dataset\_root, default='gene\_dataset', help='path to dataset root used for training.'

* Output compiled data (train\_00/, train\_01/…) into /gene\_dataset

6. Make target binary expression pattern” (with the identical OTU names).

target binary expression pattern file (with a specific name) is like (tab-delimited),

geneA 0

geneB 1

geneC 0

geneD 0

geneE 1

**Now, the file/code structure is like,**

/root

├ **1dCNN\_CisDecoding\_training\_basic.py**  
├ data\_utils/generator.py  
├ make\_dataset.py  
├ gene\_dataset/  
│     ├ train\_00/  
│     │   ├ >XXX.npy  
│     │   ├ >YYY.npy  
│     │   ⋮  
│     │   └ >ZZZ.npy  
│     ├ train\_01/  
│     ├ train\_02/  
│     ⋮  
│     └ train\_09/  
│     
├ binary\_expr\_pattern file (a specific name)  
├ cnn\_models/cnn\_model\_bisic.py

7. **$ python 1dCNN\_CisDecoding\_training\_basic.py [--n\_channel] [--data\_length] [--batch\_size] [--epochs] [--val\_rate] [--shuffle] [--class\_weight] [--target\_file] [--learning\_rate] [--out\_file] [--prediction\_file]**

--n\_channel, default=50, help='number of channels.'

--data\_length, default=20, help='length of sequence.'

--batch\_size, default=156, help='batch size for training.'

--epochs, default=10, help='number of epochs for training.'

--val\_rate, default=0.3, help='rate of validation data.'

--shuffle, default=True, help='phenotype data training shuffle'

--class\_weight, default=5, help='class-weight or positive sample imbalance rate'

--target\_file, default='BRup.txt', help='phenotype data file'

--learning\_rate, default=0.0001, help='learning rate'

--out\_file, default='model.h5', help='output model file name'

--prediction\_file, default='prediction.txt', help='output prediction confidence file name'

Output trained h5 file, list for prediction confidence in validation datasets, ROC-AUC value and curve, and confusion matrix.

**(iii) Feature visualization by Guided Backpropagation (other methods are also applicable)**

Use “**Backpropagation**” directory.

This step requires “**jupyter notebook**”, handling “ipynb” format

**(iii-1) Feature visualization in the 2nd DL framework: expr pattern -> CREs**

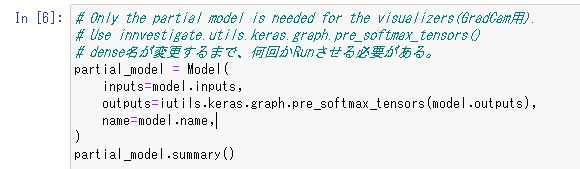
“**GuidedBackProp\_CisDecode\_batch.ipynb**”

(need “**visualizations\_forCisDecode.py**” and “**helper\_forCisDecode2**” in the same directory.)

Open jupyter, and run the ipynb file.

**NOTE**: At the third cell given below, we may have to repeat runs of this cell until the “dense” name is properly changed (expect 4-times).

This has been reported to be presumably due to flaw in the iNNvestigate library.



Need the trained prediction model “XXX.h5”, and “YYY.npy” for the objective genes, which have been made in the “make\_dataset.py” section in the section (ii) above.

The objective gene files (in npy format) need to be located on “**select\_GBP**” directory.

**(iii-2). Feature visualization in the 1st DL framework: high confidence CREs -> nucleotide residues**

“**GuidedBackprop\_CREs-prediction.ipynb**”

(need “**visualizations\_forCisDecode.py**” and “**helper\_forCisDecode2**” in the same directory.)

Need a prediction model “ZZZ.h5” for each TF channel, which has been made in the section (i), and “fragments list” for the objective tiles, which would be made from a promoter sequence.