Structure:

1. The folder ‘Data’ consists of 'LupusGeneExpressionCompendium\_AllGfeatures.csv' and 'PatientDx\_Labels.csv' files
2. Code is executed using ‘Reversible\_Net\_Tester.ipynb’
   1. Data and labels are read from the csv files
   2. Classifier is trained
   3. Tuned hyperparameters obtained from Bayes optimization of hyperparameters are used for training the classifier
   4. Classifier performance is evaluated using validation data
   5. Generator is trained
   6. Tuned hyperparameters obtained from Bayes optimization of hyperparameters are used for training the generator
   7. Generator performance is evaluated using validation data
   8. Input-like matrix is obtained using the matrix inversion method with pseudoinverse
   9. The following are used in ‘Reversible\_Net\_Tester.ipynb’ code:
      1. ‘Reversible\_Classifier.py’ class from ‘Models’ folder
         1. ‘Reversible\_Classifier.py’ is a subclass of ‘Net.py’
      2. ‘Classifier\_data.py’ class from ‘Data\_Handler’ folder
         1. ‘Classifier\_data.py’ is a subclass of ‘Data.py’
3. The hyperparameters are tuned using ‘BayesOpt.ipynb’
   1. The optimized values of the hyperparameters were obtained and manually fed in the model in ‘Reversible\_Net\_Tester.ipynb’

**Model:**

1. *Net:*
2. Super class inherited by all the other classes (Reversible\_classifier, classifier, GANs) in Models folder
3. Consists of functions to:
   1. Build graph
   2. Get default configurations
   3. Learn by feeding tensors into the model
   4. Train the model
   5. Save the model and config file
   6. Evaluate the model
4. Functions a, b, c and f need to be overridden in the subclass/child-class
5. *Reversible Classifier*
6. Subclass that inherits superclass Net
7. Code for classifier, generator and classifier network inversion
8. Splits data into train, validation and test sets
9. Functions:
   1. build\_graph:
      1. Defines a dataflow graph which gets executed when it is run in a session
      2. Number of layers and nonlinearity (activation function) are specified
      3. Builds graph for classifier, generator and classier-network inversion components
      4. Uses pseudoinverse for inversion of classifier layers
         1. The operations in the classifier layer (leaky\_relu(WX + B)) are inverted in the following sequence:
            1. Inverse leaky\_relu applied to the last layer
            2. Bias is subtracted from the result
            3. The resulting matrix is multiplied by pseudoinverse on the weight matrix to obtain X
      5. Also specifies the components for optimization of classifier and generator
   2. define\_ops (Optimizer for classifier):
      1. Softmax cross entropy loss
      2. L2 Regularization is added to avoid overfitting of training data and improve accuracy
      3. The optimizer hyperparameters are defined
      4. Regularized loss is minimized
      5. Adamoptimizer is used for minimizing the loss function
   3. define\_G\_ops (Optimizer for Generator):
      1. Uses two loss operations:
         1. Input loss: mean squared error for input and input-like matrix
         2. Discriminator loss: softmax cross entropy with logits for classification of the input-like matrix
      2. Generator loss = input loss + (alpha \* discriminator loss)
      3. Hyperparameters are defined
      4. Adamoptimizer is used for minimizing the loss function
   4. create\_weight\_bias:
      1. Weights and biases matrices are created and initialized
      2. Weight matrices initialized using Xavier initializer
      3. Bias matrices initialized using Glorot uniform initializer
   5. get\_default\_configs:
      1. Defines all default values for parameters and hyperparameters to be used for the program if no value is specified
   6. build\_network:
      1. Builds the neural network
      2. Specifies the slope of leaky relu to be used
      3. Determines the parameters of the model: number of layers, number of inputs and number of outputs
      4. Defines the network by:
         1. Creating weights (W) and biases (b)
         2. Implementing the operation: leaky\_relu(WX + b) where X is the input to any node in a layer
         3. Calculates the output values after the above operations and appends all the outputs in a list named ‘outputs’
   7. pinv:
      1. Calculates the pseudoinverse of the matrix passed in as argument
   8. inverse\_relu:
      1. Defines the inverse relu function used in network inversion
   9. learn (for classifier):
      1. Updates the hyperparameters using the Bayes Optimization results
      2. Specifies number of iterations
      3. Converts input data type to tensors to be fed in the Tensorflow model
      4. Batch\_x and batch\_y which are fed into the model for training are tensors
      5. Trains the classifier and calculates loss by running the components of the optimizer in a Tensorflow session
      6. Prints the training loss calculated after each epoch
   10. evaluate (for classifier):
       1. Function for model evaluation with validation data
       2. Calculates loss after each epoch and makes prediction on the validation data by running the loss and prediction functions in a Tensorflow session
       3. Compares the prediction made by the model on the validation data with the correct Y value
       4. Evaluates and prints the confusion matrix with the help of this comparison
       5. Prints the value of validation loss after each epoch
   11. train\_generator:
       1. Function for generator training
       2. Specifies the max\_epoch hyperparameter
       3. Trains the generator usinf the G\_learn function
       4. Saves and evaluates the model
   12. G\_learn:
       1. Function to get tensors for generator training
       2. Updates the hyperparameters
       3. Feeds the input tensors to the model
       4. Trains the generator and calculates the generator loss by running the generator optimization components in a Tensorflow session
       5. Evaluates and prints the training loss
   13. G\_evaluate (for generator):
       1. Function to evaluate generator performance using validation data
       2. Feeds the input validation tensors to the generator model
       3. Evaluates the generator loss by running the generator optimization components in a Tensorflow session
       4. Compares the predicted label with the correct label
       5. Uses this comparison to generate a confusion matrix
       6. Evaluates and prints the validation loss
   14. zipDicts:
       1. static method for all classes
       2. Concatenates two dictionaries

**Data\_Handler**

1. *Data:*
2. Abstract class for handling the data
3. Reads, shuffles, splits the data
4. Functions:
   1. setBatchSize
   2. getBatch
   3. splitData
   4. shuffle
   5. load\_from
5. All functions are to be defined in the inheriting classes
6. *Classifier\_Data:*
7. Inherits the data abstract class
8. Instance variables:
   1. Data
   2. Labels
   3. Num\_of\_samples
   4. Num\_of\_batches
   5. Classes
   6. Num\_of\_classes
   7. One\_hot\_labels
   8. Num\_of\_features
9. Functions:
   1. One\_hot:
      1. Static method for all classes
      2. Gets the data encoded using one hot encoding
   2. shuffle:
      1. Randomly shuffles the data and labels using a seed
   3. balanceData:
      1. Upsamples the data to manage data imbalance
      2. Uses SMOTE (Synthetic Minority Over-Sampling Technique) function from imbalance learn library of Python
      3. Reassigns values from new balanced dataset to the instance variables
      4. Print the number of samples in each class
   4. splitData:
      1. Function to split the data given a splitting fraction
      2. Shuffles the data and splits the data and labels using numpy split function
   5. load\_from:
      1. Reads the data from a given data file passed in as the argument
   6. getBatch:
      1. Divides the dataset into batches of previously specified batch size
   7. setBatchSize:
      1. Sets the instance variable for batch size
      2. Also computes number of batches = number of samples / batch size