

# **Classification using GCN**

***Submitted By:***

***Divya Saxena (1001773376)***

## Table of Contents

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Code Description.....</b>	<b>3</b>
<b>3. How to run code.....</b>	<b>6</b>
<b>4. Graph Convolutional Neural Network Model Description.....</b>	<b>6</b>
<b>5. Image Visualization Healthy vs Patient.....</b>	<b>7</b>
<b>6. Data Preprocessing.....</b>	<b>9</b>
<b>7. Model Training.....</b>	<b>12</b>
<b>8. Graphs/Accuracy/Loss.....</b>	<b>14</b>
<b>9. Driver script.....</b>	<b>17</b>
<b>10. References.....</b>	<b>17</b>

## 1. Introduction

Neuro-imaging is a branch of medical imaging that focuses on the brain. In addition to diagnosing disease and assessing brain health, neuro-imaging also studies:

- How the brain works
- How various activities impact the brain

Currently, most graph neural network models have a somewhat universal architecture in common. I will refer to these models as *Graph Convolutional Networks* (GCNs); convolutional, because filter parameters are typically shared over all locations in the graph

For these models, the goal is then to learn a function of signals/features on a graph  $G=(V,E)$  which takes as input:

- A feature description  $x_i$

for every node  $i$ ; summarized in a  $N \times D$  feature matrix  $X$  ( $N$ : number of nodes,  $D$

- : number of input features)
- A representative description of the graph structure in matrix form; typically in the form of an adjacency matrix  $A$
- (or some function thereof)

and produces a node-level output  $Z$

(an  $N \times F$  feature matrix, where  $F$  is the number of output features per node). Graph-level outputs can be modeled by introducing some form of pooling operation

2D Images have been given as Input to GCN for binary classification of images as Healthy and Patient.

Independent test dataset has been used for evaluating the model performance.

## 2. Code Description

### Python Files

```
/GCN/preprocess_data.py
/GCN/preprocessing_testing.py
/GCN/show_image.py
/GCN/dataset_split.py
/GCN/graphplt.py
/GCN/load_data_sets.py
/GCN/train.py
```

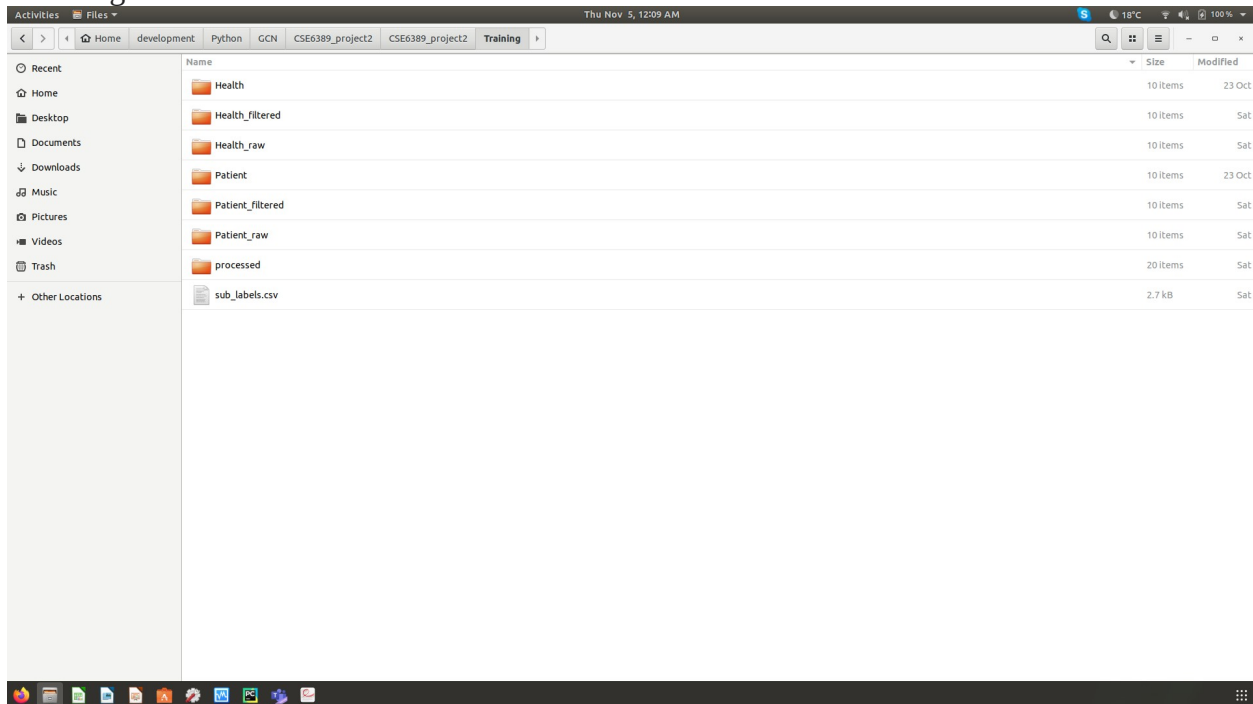
### Packages Used for Coding

numpy  
scipy  
os  
matplotlib.pyplot  
pandas  
sklearn  
skimage  
pathlib  
tensorflow  
spektral

## Folder Structure for dataset:

/GCN/CSE6389\_project2/CSE6389\_project2/Training

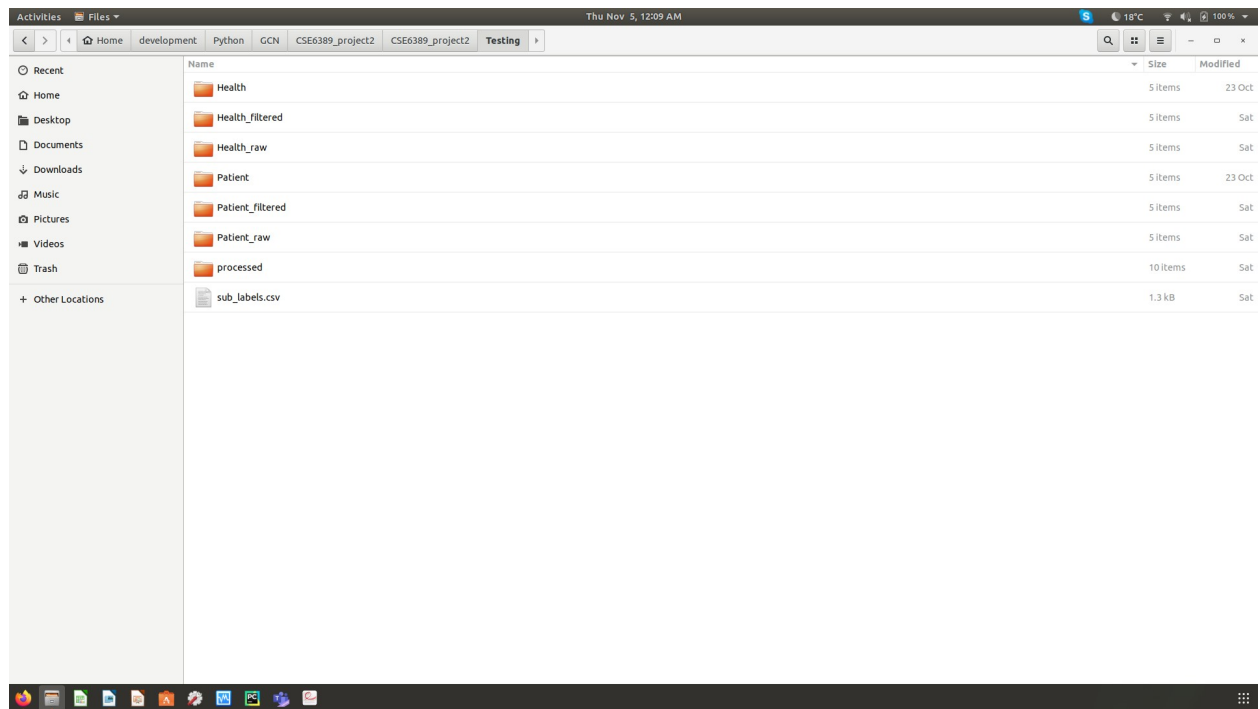
I have created some folders after each stage during pre-processing as attached below, images during each stage will get saved in new folder, so that we can easily identify the changes during each stage.



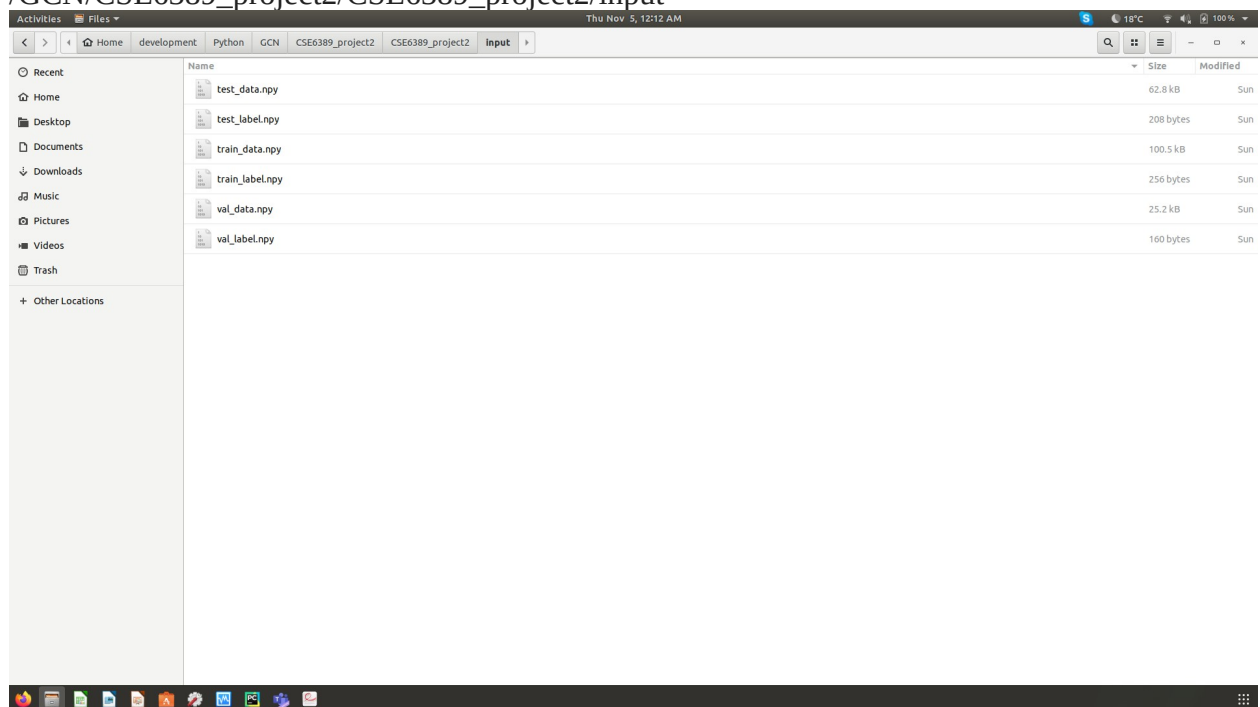
/GCN/CSE6389\_project2/CSE6389\_project2/Testing

We have created some folders after each stage during pre-processing as attached below, images during each stage will get saved in new folder, so that we can easily identify the changes during each stage.

# CSE-6389-ADV MM, GRAPHICS, IMAGE PROC



After pre-processing, some files have been generated for the GCN model as attached below.  
These files will be used by GCN model for training/testing/validation  
/GCN/CSE6389\_project2/CSE6389\_project2/input



### 3. How to run code

#### To Show Image

```
python /GCN/show_image.py
```

#### For Pre-processing Data

```
python /GCN/preprocessing_testing.py
```

#### To Train Model

```
python /GCN/train.py
```

### 4. Graph Convolutional Neural Network Model Description

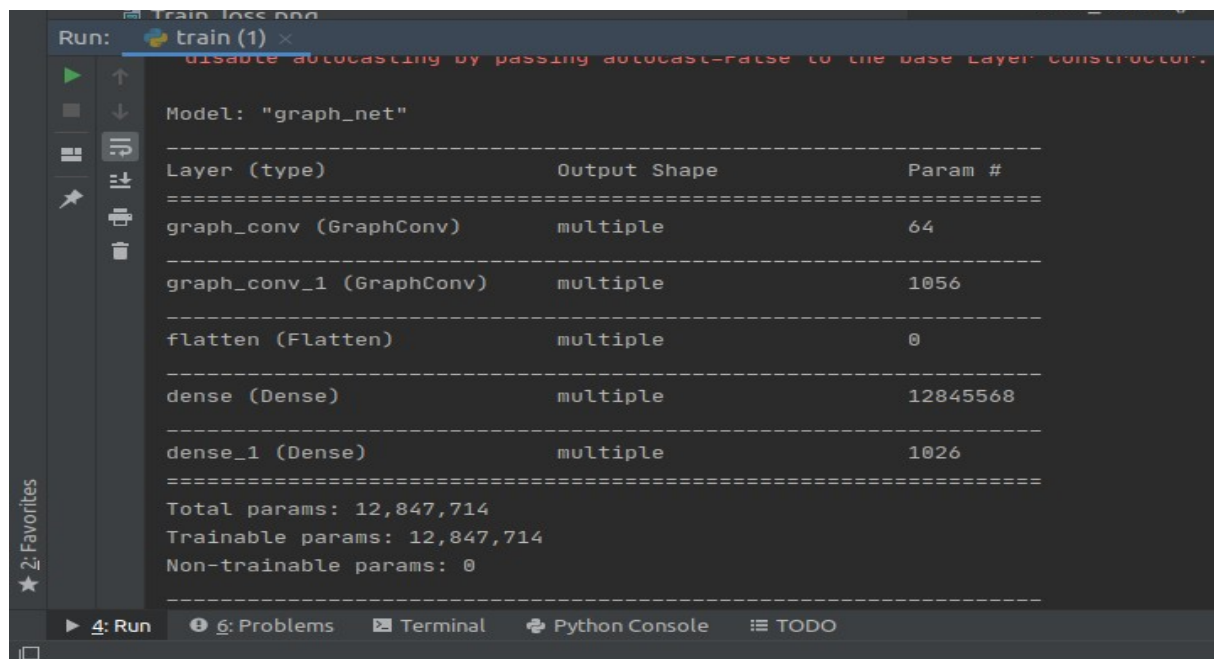
"GraphConv" model has been used to create GCN. 5 layers have used to create the model.

Attaching screenshot for the same. Images given are very big in size approximately 150\*150 as we can see [here](#). Image has been resized to 28\*28 due to limited GPU and CPU.

Adam optimizer has been used as it is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks.

SparseCategoricalCrossentropy has been used as it computes the crossentropy loss between the labels and predictions

SparseCategoricalAccuracy has been used as it Calculates how often predictions matches integer labels



The screenshot shows a Jupyter Notebook interface with a dark theme. The 'Run' button is highlighted, and the output of the 'train' function is displayed. The output shows the model name 'graph\_net' and a summary table of the layers. The table has three columns: 'Layer (type)', 'Output Shape', and 'Param #'. The layers listed are 'graph\_conv' (GraphConv), 'graph\_conv\_1' (GraphConv), 'flatten' (Flatten), 'dense' (Dense), and 'dense\_1' (Dense). The total number of parameters is 12,847,714, and the number of trainable parameters is also 12,847,714. The number of non-trainable parameters is 0.

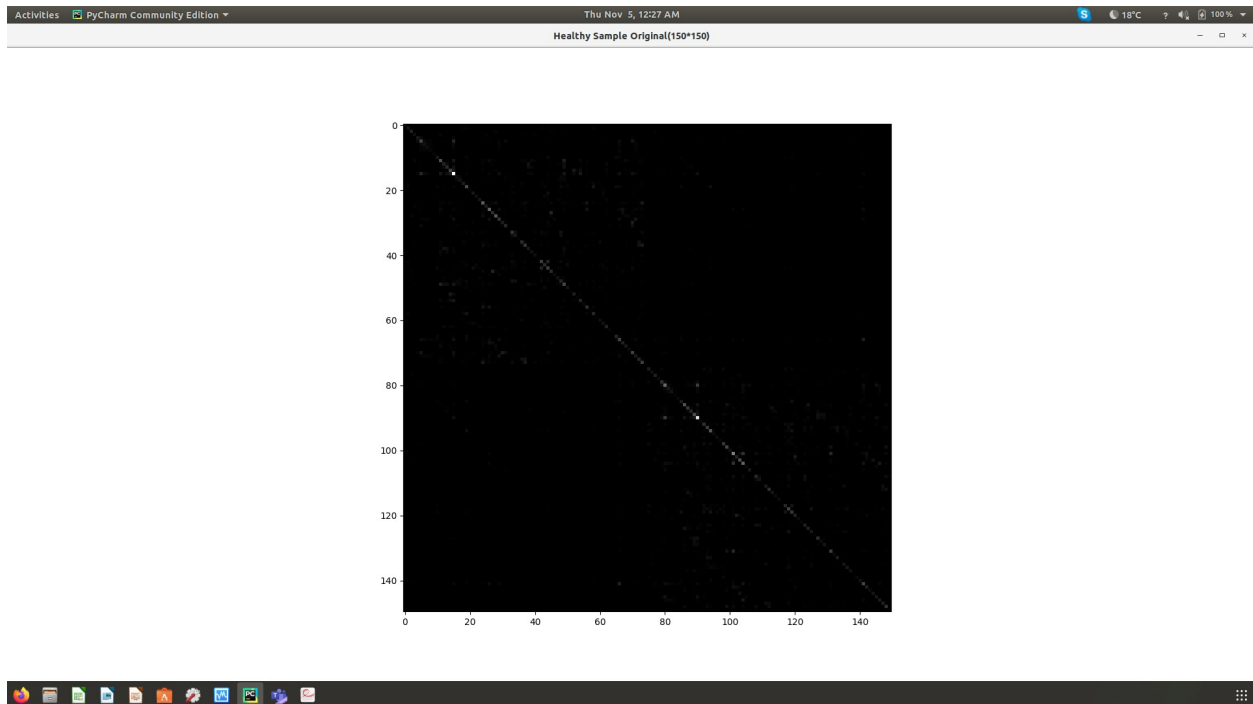
Layer (type)	Output Shape	Param #
graph_conv (GraphConv)	multiple	64
graph_conv_1 (GraphConv)	multiple	1056
flatten (Flatten)	multiple	0
dense (Dense)	multiple	12845568
dense_1 (Dense)	multiple	1026
Total params: 12,847,714		
Trainable params: 12,847,714		
Non-trainable params: 0		

## 5. Image Visualization Healthy vs Patient

/GCN/show\_image.py

In this python file, we have tried to show the 2D-image of both Healthy and Patient Dataset. We have randomly selected one image of both dataset given. The dimension used are the original ones i.e. 150\*150

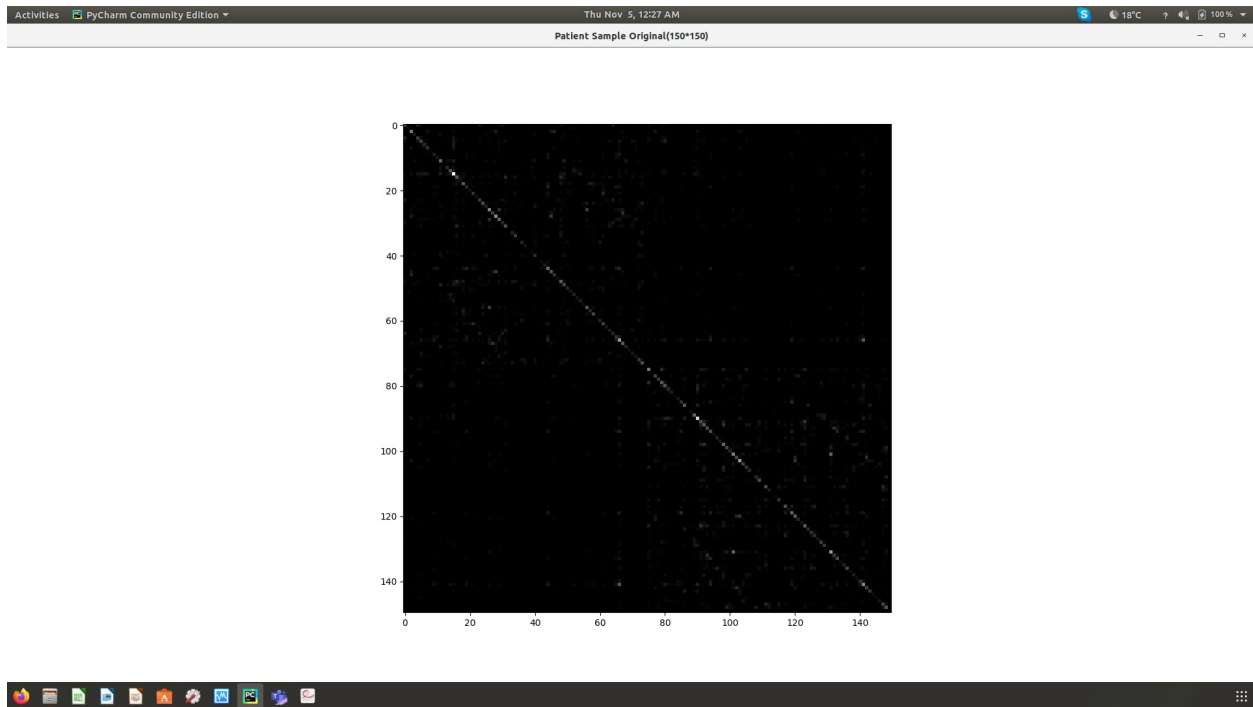
### 1. Health



### 2. Patient

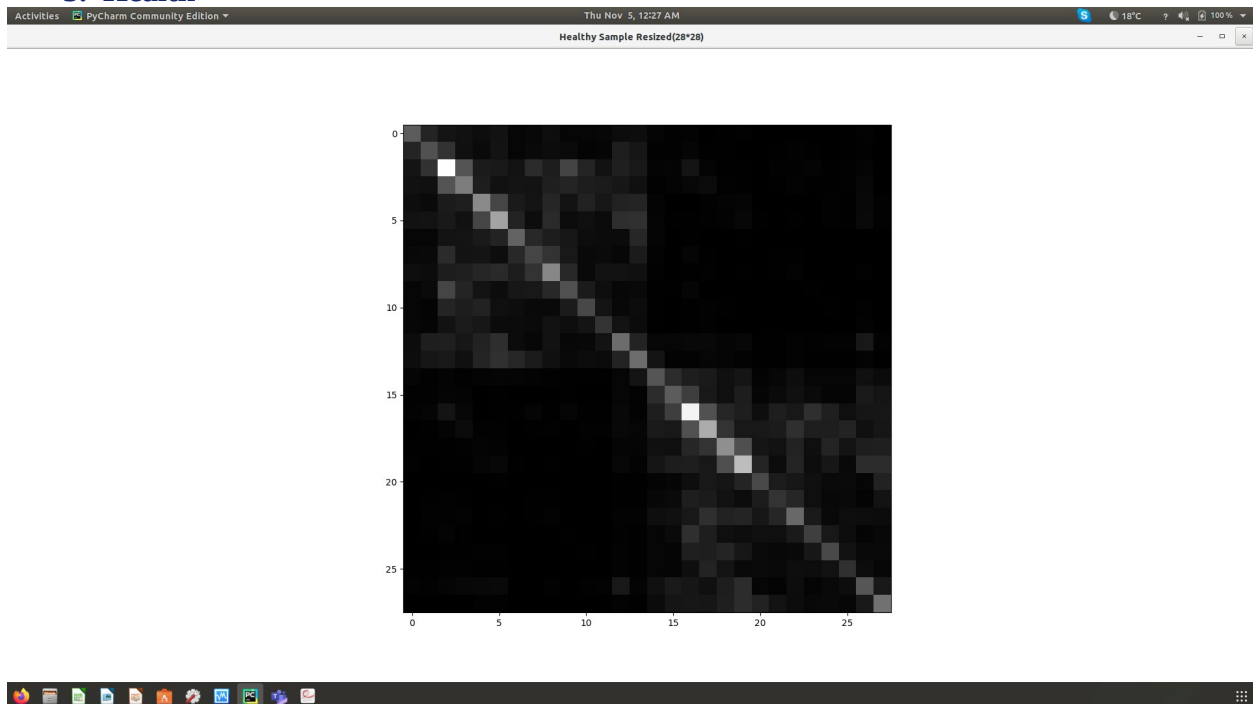
We can see some differences in both Health and patient images.

# CSE-6389-ADV MM, GRAPHICS, IMAGE PROC



We have randomly selected one image of both dataset given. The dimension used are the resized ones i.e.  $28 \times 28$ . As mentioned earlier, due to limited GPU and CPU, we have resized the original image to  $28 \times 28$  for training our model.

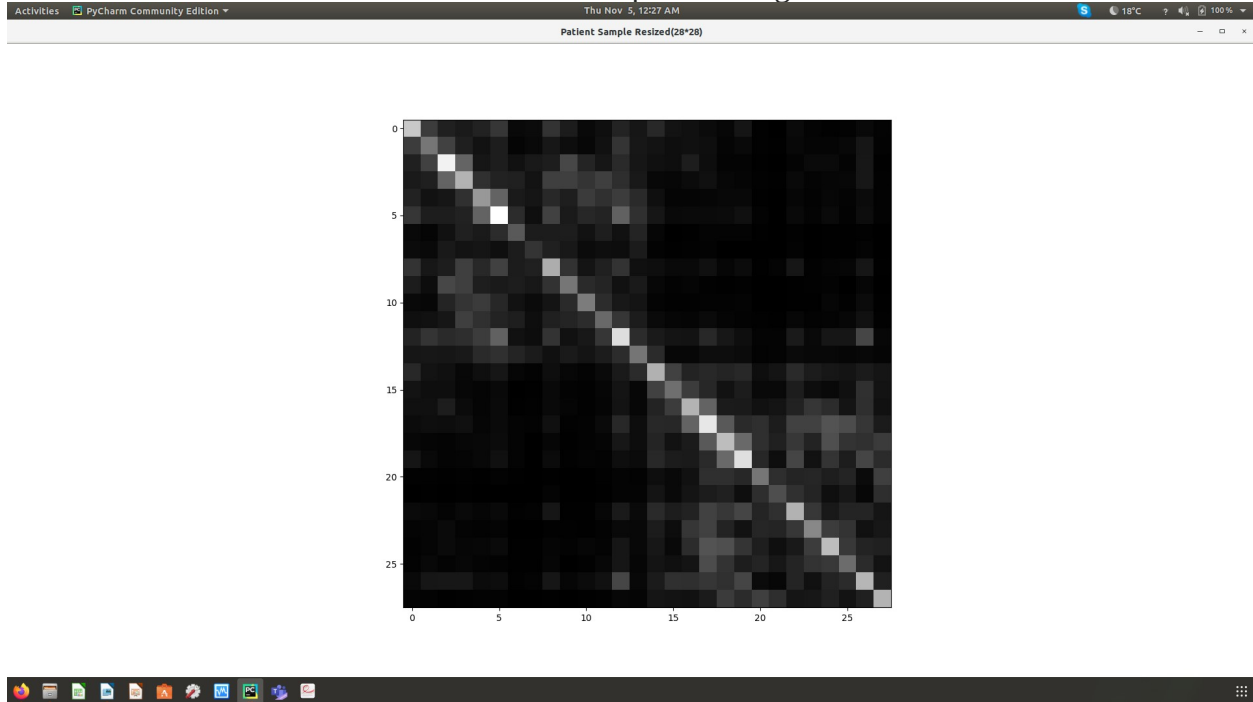
## 3. Health





#### 4. Patient

We can see some differences in both Health and patient images.



#### 6. Data Preprocessing

Same Pre-processing steps have been used as in previous project as stated below:

/GCN/preprocess\_data.py

/GCN/preprocessing\_testing.py

Below steps have been followed for both Training and Testing dataset but testing data kept separately to avoid over-fitting of the model.

Code has been implemented as below:

1.Put .txt files in patient\_raw, health\_raw

Matrices will be saved in folder separately for Training and testing i.e.

(/GCN/CSE6389\_project2/CSE6389\_project2/Training/health\_raw,/GCN/CSE6389\_project2/CSE6389\_project2/Training/patient\_raw)

and

(/GCN/CSE6389\_project2/CSE6389\_project2/Testing/health\_raw,/GCN/CSE6389\_project2/CSE6389\_project2/Testing/patient\_raw)

2.Filtered .txt to .npy in folder(patient\_filtered, health\_filtered)

(/GCN/CSE6389\_project2/CSE6389\_project2/Training/health\_filtered,/GCN/CSE6389\_project2/CSE6389\_project2/Training/patient\_filtered)

and

## CSE-6389-ADV MM, GRAPHICS, IMAGE PROC

(/GCN/CSE6389\_project2/CSE6389\_project2/Testing/health\_filtered,/GCN/CSE6389\_project2/CSE6389\_project2/Testing/patient\_filtered)

3.Processed all healthy/patient to processed folder

Pre-processing of original images. Pre-processing include: resizing

Image has been resized to 28\*28 due to limited GPU and CPU. Single folder for both Health and Patient. But the images can be differentiated by their names as for patient the images have been renamed like (%\_pat\_%)

(/GCN/CSE6389\_project2/CSE6389\_project2/Training/processed)

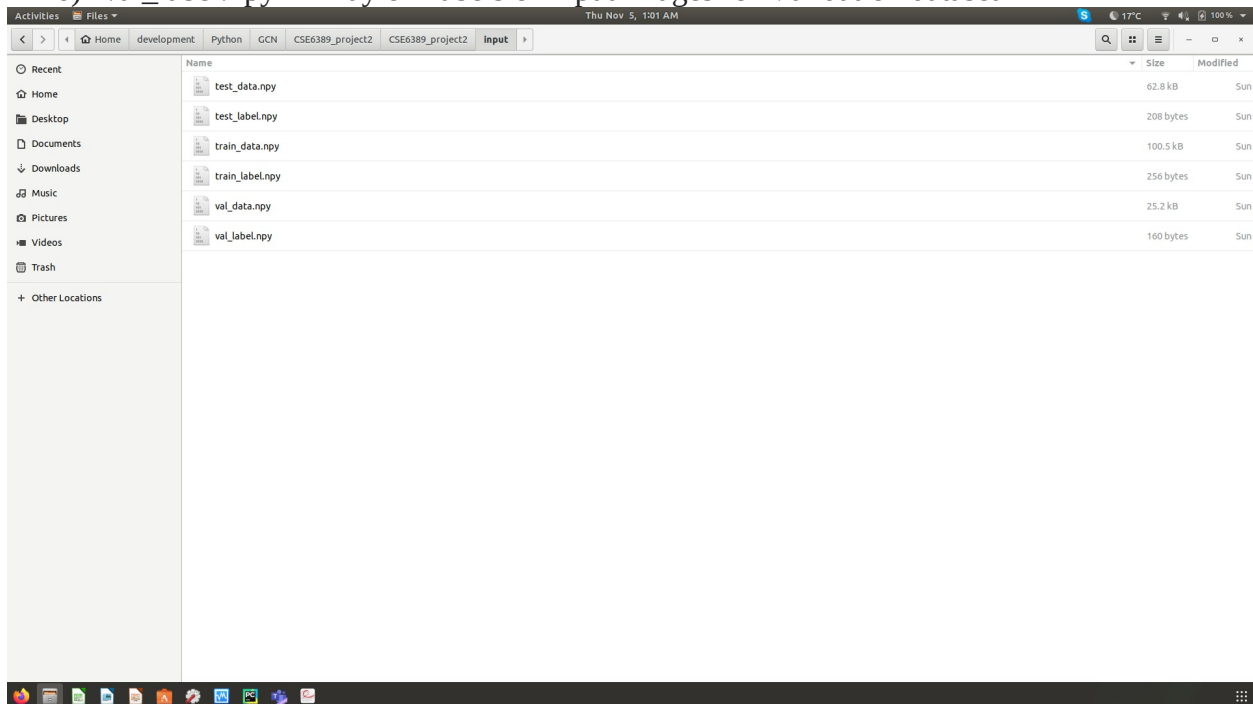
and

(/GCN/CSE6389\_project2/CSE6389\_project2/Testing/processed)

4.Final step is after pre-processing, data has been split into 2 parts :

Training and Validation. All the data has been converted into single files .npy. And Testing dataset as well. All files have been created in Input folder as data and their corresponding labels

- 1) test\_data.npy - Array of Input images for Testing dataset
- 2) test\_label.npy - Array of Labels of Input images for Testing dataset
- 3) train\_data.npy - Array of Input images for Training dataset
- 4) train\_label.npy - Array of Labels of Input images for Training dataset
- 5) val\_data.npy - Array of Input images for Validation dataset
- 6) val\_label.npy - Array of Labels of Input images for Validation dataset



5.Some manual steps required for Training the model

Created labels for Training/Validation in sub\_label.csv as Subject and Group as the heading at location /GCN/CSE6389\_project2/CSE6389\_project2/Training

## CSE-6389-ADV MM, GRAPHICS, IMAGE PROC

A	B	C
1	Subject	Group
2	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix1_processed.npy
3	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix2_processed.npy
4	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix3_processed.npy
5	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix4_processed.npy
6	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix5_processed.npy
7	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix6_processed.npy
8	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix7_processed.npy
9	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix8_processed.npy
10	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix9_processed.npy
11	Health	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix10_processed.npy
12	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix1_pat_processed.npy
13	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix2_pat_processed.npy
14	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix3_pat_processed.npy
15	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix4_pat_processed.npy
16	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix5_pat_processed.npy
17	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix6_pat_processed.npy
18	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix7_pat_processed.npy
19	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix8_pat_processed.npy
20	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix9_pat_processed.npy
21	Patient	/home/diviya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/processed/common_fiber_matrix10_pat_processed.npy

6.Created labels for Testing in sub\_label.csv as Subject and Group as the heading at location /GCN/CSE6389\_project2/CSE6389\_project2/Testing

The screenshot displays the LibreOffice Calc interface with a spreadsheet titled 'sub\_labels.csv'. The spreadsheet contains the following data:

A	B
Subject	Group
Health	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix1_processed.npy
Health	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix2_processed.npy
Health	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix3_processed.npy
Health	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix4_processed.npy
Health	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix5_processed.npy
Patient	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix1_pat_processed.npy
Patient	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix2_pat_processed.npy
Patient	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix3_pat_processed.npy
Patient	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix4_pat_processed.npy
Patient	/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Testing/processed/common_fiber_matrix5_pat_processed.npy

The status bar at the bottom indicates 'Sheet 1 of 1', 'Default', 'English (India)', and 'Average: Count: 0; Sum: 0'.

## 7. Model Training

/GCN/train.py

Model has been trained for 1000 number of epochs as attached below using Training and validation dataset. Testing dataset has been kept aside for evaluating the model. It has been used as Independent dataset to avoid over-fitting.

Firstly, pre-processed data has been loaded for the model as screenshot below:

The screenshot shows the PyCharm IDE with the file `load_data_sets.py` open. The code defines a function `load_data()` that loads training, validation, and test data from a specified path. The Run console at the bottom displays the output of the `train()` function, showing training and validation loss and accuracy over 1000 epochs.

```

1 import numpy as np
2 import os
3
4 path = os.getcwd()
5
6 def load_data():
7     x_train = np.load(path + "/CSE6389_project2/CSE6389_project2/input/train_data.npy",
8                       allow_pickle=True)
9     y_train = np.load(
10         path + "/CSE6389_project2/CSE6389_project2/input/train_label.npy",
11         allow_pickle=True)
12     x_test = np.load(path + "/CSE6389_project2/CSE6389_project2/input/test_data.npy",
13                     allow_pickle=True)
14     y_test = np.load(path + "/CSE6389_project2/CSE6389_project2/input/test_label.npy",
15                     allow_pickle=True)
16     x_val = np.load(path + "/CSE6389_project2/CSE6389_project2/input/val_data.npy",
17                    allow_pickle=True)
18     y_val = np.load(path + "/CSE6389_project2/CSE6389_project2/input/val_label.npy",
19                    allow_pickle=True)
20
21     return (x_train, y_train), (x_test, y_test), (x_val, y_val)
22
23
24 load_data()
  
```

Run: train (1)

Epoch	Train Loss	Train Acc	Valid Loss	Valid Acc	Test Loss	Test Acc
1	0.0083	0.8541	4.9333	0.8541	3.8123	0.8548
2	0.0083	0.8541	4.9338	0.8541	3.8126	0.8548
3	0.0083	0.8541	4.9313	0.8541	3.8116	0.8541
4	0.0083	0.8541	4.9294	0.8541	3.8184	0.8541
5	0.0083	0.8541	4.9283	0.8541	3.8188	0.8541
6	0.0083	0.8542	4.9282	0.8541	3.8186	0.8541
7	0.0083	0.8542	4.9298	0.8542	3.8121	0.8541
8	0.0083	0.8542	4.9281	0.8542	3.8148	0.8541
9	0.0082	0.8542	4.9385	0.8542	3.8151	0.8541
10	0.0082	0.8542	4.9382	0.8542	3.8155	0.8541
11	0.0082	0.8542	4.9292	0.8542	3.8152	0.8542
12	0.0082	0.8542	4.9288	0.8542	3.8147	0.8542
13	0.0082	0.8542	4.9271	0.8542	3.8146	0.8542

For training, we have used `tf.GradientTape`. TensorFlow provides the API for automatic differentiation; that is, computing the gradient of a computation with respect to some inputs, usually `tf.Variables`. TensorFlow "records" relevant operations executed inside the context of `tf.GradientTape` onto a "tape". TensorFlow then uses that tape to compute the gradients of a "recorded" computation using reverse mode differentiation.

# CSE-6389-ADV MM, GRAPHICS, IMAGE PROC

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
GCN - train.py

Project: GCN - development/Python/GCN
dataset_split.py load_data_sets.py graphplt.py preprocess_data.py preprocessing_testing.py show_image.py train.py

Run: train()
Model: "graph_net"
Layer (type) Output Shape Param #
-----
graph_conv (GraphConv) multiple 64
graph_conv_1 (GraphConv) multiple 1056
flatten (Flatten) multiple 0
dense (Dense) multiple 12845568
dense_1 (Dense) multiple 1820
Total params: 12,847,714
Trainable params: 12,847,714
Non-trainable params: 0

Train loss: 0.7843, acc: 0.5000 | Valid loss: 1.9245, acc: 0.5000 | Test loss: 1.7144, acc: 0.5000
Train loss: 1.5915, acc: 0.5000 | Valid loss: 0.8136, acc: 0.5000 | Test loss: 0.6118, acc: 0.5000
Train loss: 0.7118, acc: 0.5000 | Valid loss: 0.7148, acc: 0.5000 | Test loss: 1.5083, acc: 0.5000
Train loss: 0.8848, acc: 0.5000 | Valid loss: 0.7843, acc: 0.5000 | Test loss: 1.5358, acc: 0.5000
Train loss: 0.8738, acc: 0.5000 | Valid loss: 0.6981, acc: 0.5000 | Test loss: 1.2838, acc: 0.5000
Train loss: 0.8124, acc: 0.5000 | Valid loss: 0.6857, acc: 0.5000 | Test loss: 1.0469, acc: 0.5000
```

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
GCN - train.py

Project: GCN - development/Python/GCN
dataset_split.py load_data_sets.py graphplt.py preprocess_data.py preprocessing_testing.py show_image.py train.py

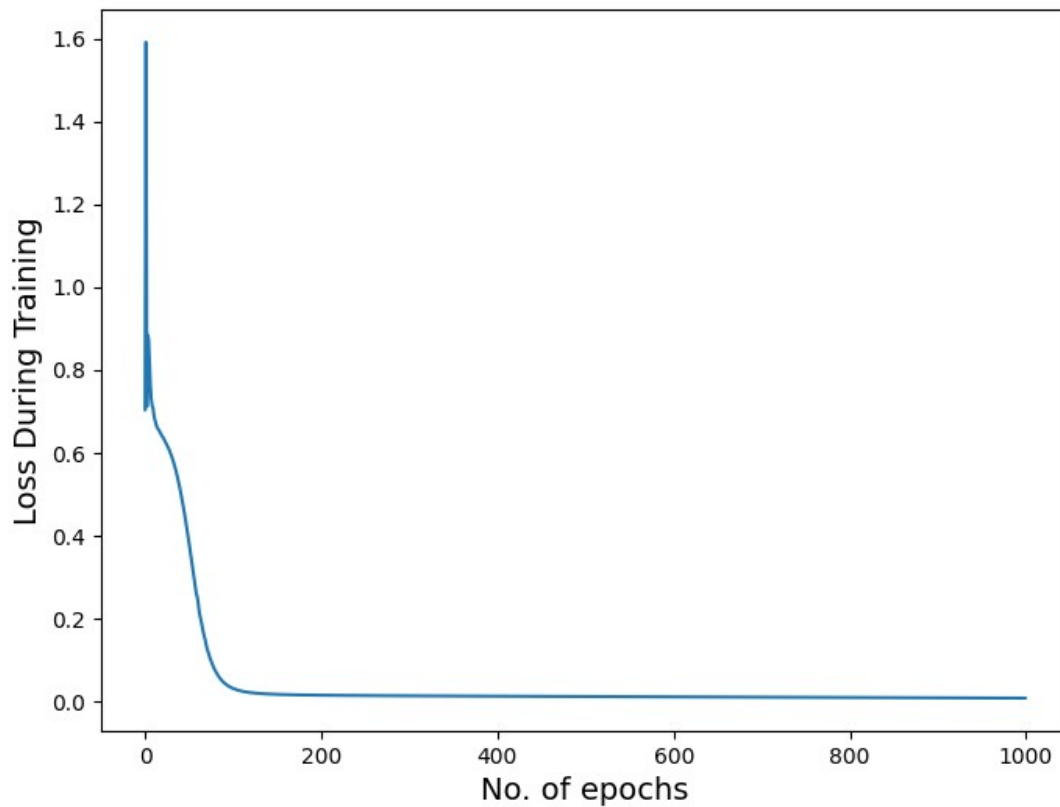
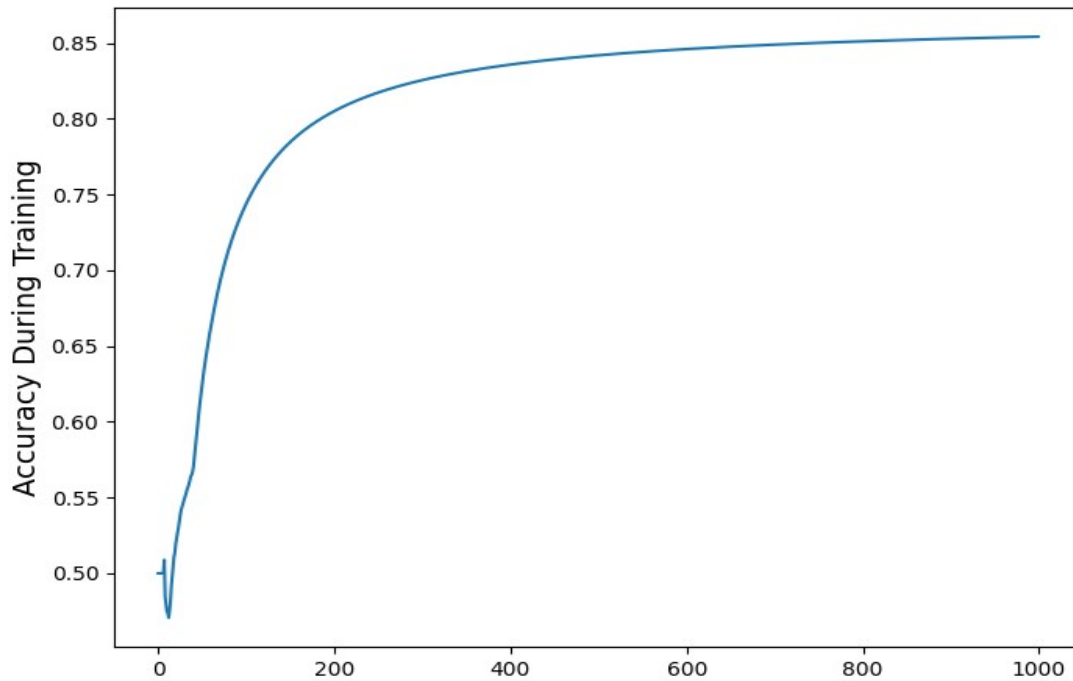
Run: train()
Train loss: 0.0083, acc: 0.8541 | Valid loss: 4.9323, acc: 0.8541 | Test loss: 3.0186, acc: 0.8540
Train loss: 0.0083, acc: 0.8541 | Valid loss: 4.9333, acc: 0.8541 | Test loss: 3.0123, acc: 0.8540
Train loss: 0.0083, acc: 0.8541 | Valid loss: 4.9338, acc: 0.8541 | Test loss: 3.0126, acc: 0.8540
Train loss: 0.0083, acc: 0.8541 | Valid loss: 4.9313, acc: 0.8541 | Test loss: 3.0118, acc: 0.8541
Train loss: 0.0083, acc: 0.8541 | Valid loss: 4.9294, acc: 0.8541 | Test loss: 3.0184, acc: 0.8541
Train loss: 0.0083, acc: 0.8541 | Valid loss: 4.9283, acc: 0.8541 | Test loss: 3.0188, acc: 0.8541
Train loss: 0.0083, acc: 0.8542 | Valid loss: 4.9282, acc: 0.8541 | Test loss: 3.0186, acc: 0.8541
Train loss: 0.0083, acc: 0.8542 | Valid loss: 4.9298, acc: 0.8542 | Test loss: 3.0121, acc: 0.8541
Train loss: 0.0083, acc: 0.8542 | Valid loss: 4.9381, acc: 0.8542 | Test loss: 3.0148, acc: 0.8541
Train loss: 0.0082, acc: 0.8542 | Valid loss: 4.9385, acc: 0.8542 | Test loss: 3.0151, acc: 0.8541
Train loss: 0.0082, acc: 0.8542 | Valid loss: 4.9282, acc: 0.8542 | Test loss: 3.0155, acc: 0.8541
Train loss: 0.0082, acc: 0.8542 | Valid loss: 4.9292, acc: 0.8542 | Test loss: 3.0152, acc: 0.8542
Train loss: 0.0082, acc: 0.8542 | Valid loss: 4.9288, acc: 0.8542 | Test loss: 3.0147, acc: 0.8542
Train loss: 0.0082, acc: 0.8542 | Valid loss: 4.9271, acc: 0.8542 | Test loss: 3.0146, acc: 0.8542
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9268, acc: 0.8542 | Test loss: 3.0158, acc: 0.8542
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9278, acc: 0.8543 | Test loss: 3.0159, acc: 0.8542
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9275, acc: 0.8543 | Test loss: 3.0172, acc: 0.8542
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9277, acc: 0.8543 | Test loss: 3.0181, acc: 0.8542
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9274, acc: 0.8543 | Test loss: 3.0185, acc: 0.8542
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9267, acc: 0.8543 | Test loss: 3.0186, acc: 0.8543
Train loss: 0.0082, acc: 0.8543 | Valid loss: 4.9261, acc: 0.8543 | Test loss: 3.0187, acc: 0.8543

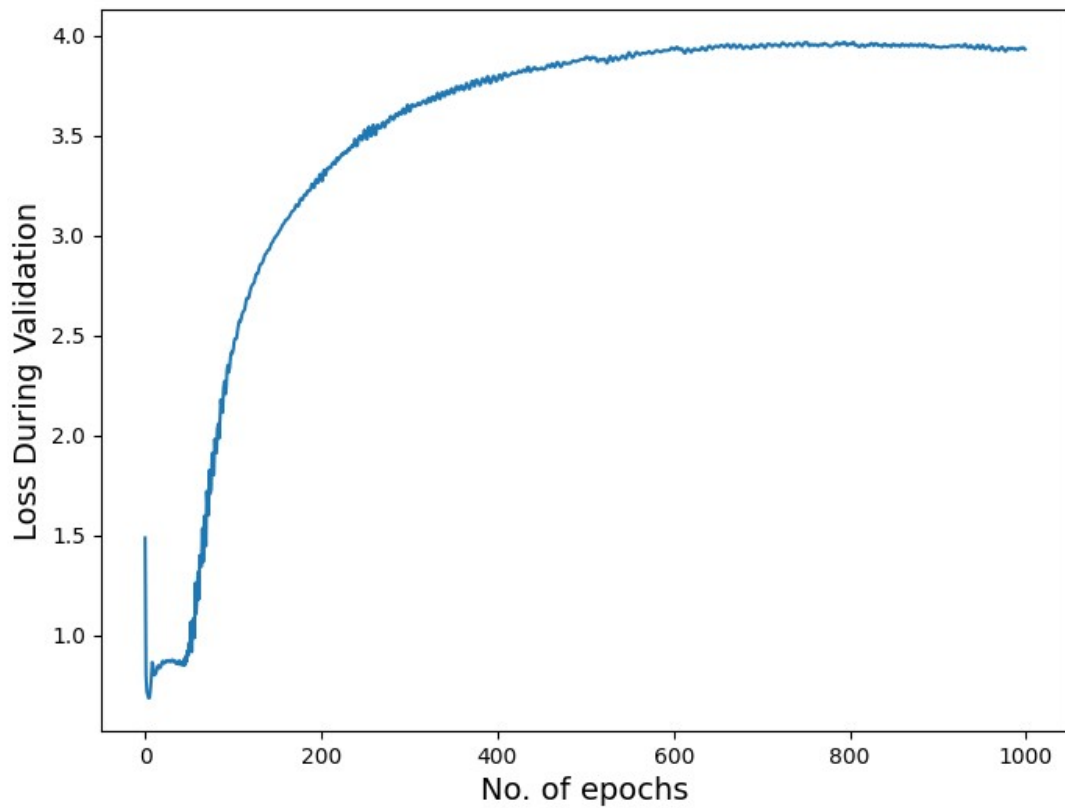
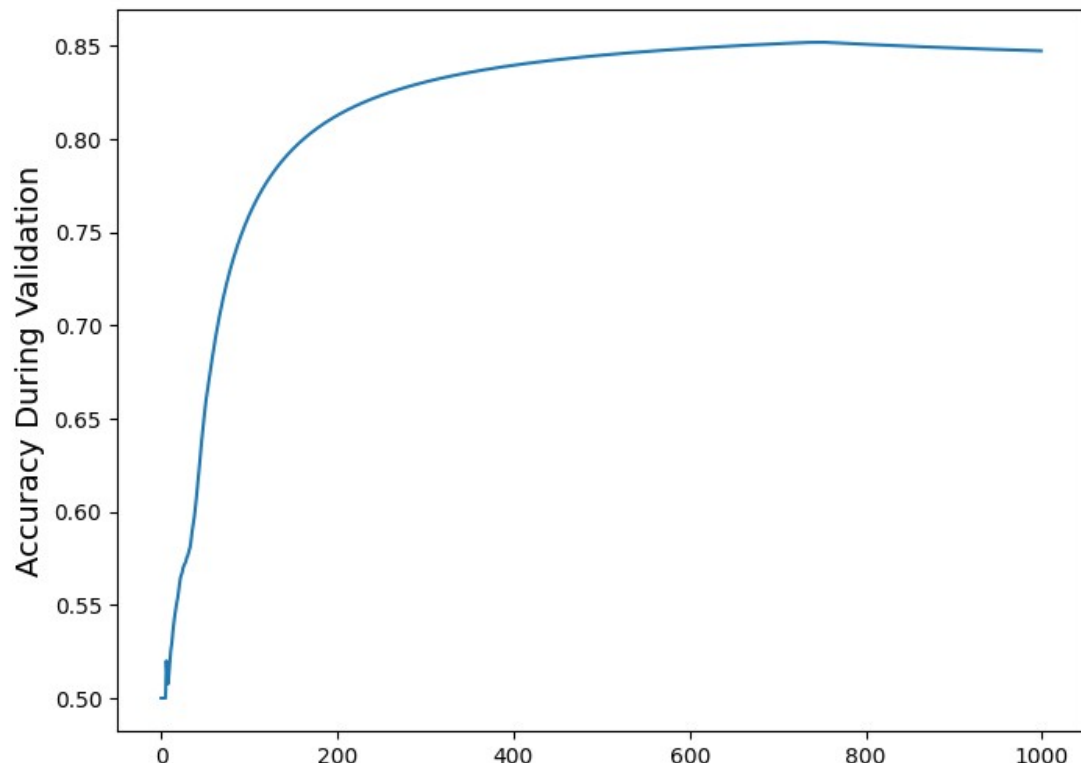
Process finished with exit code 0
```

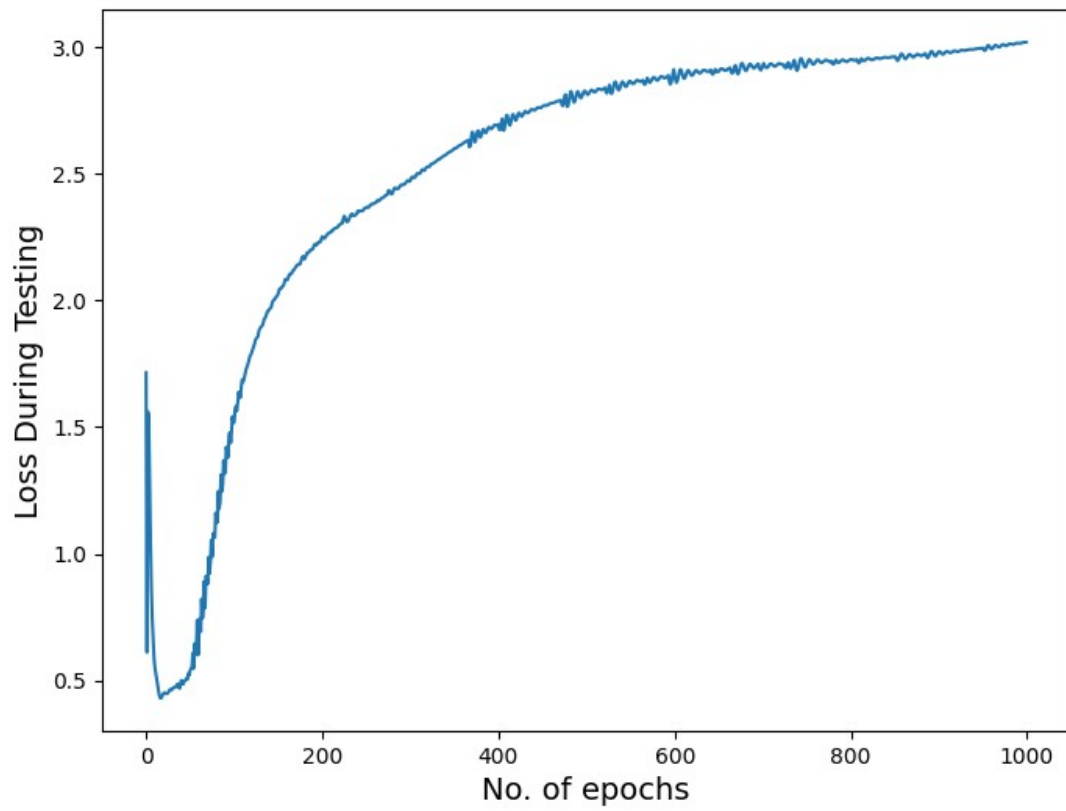
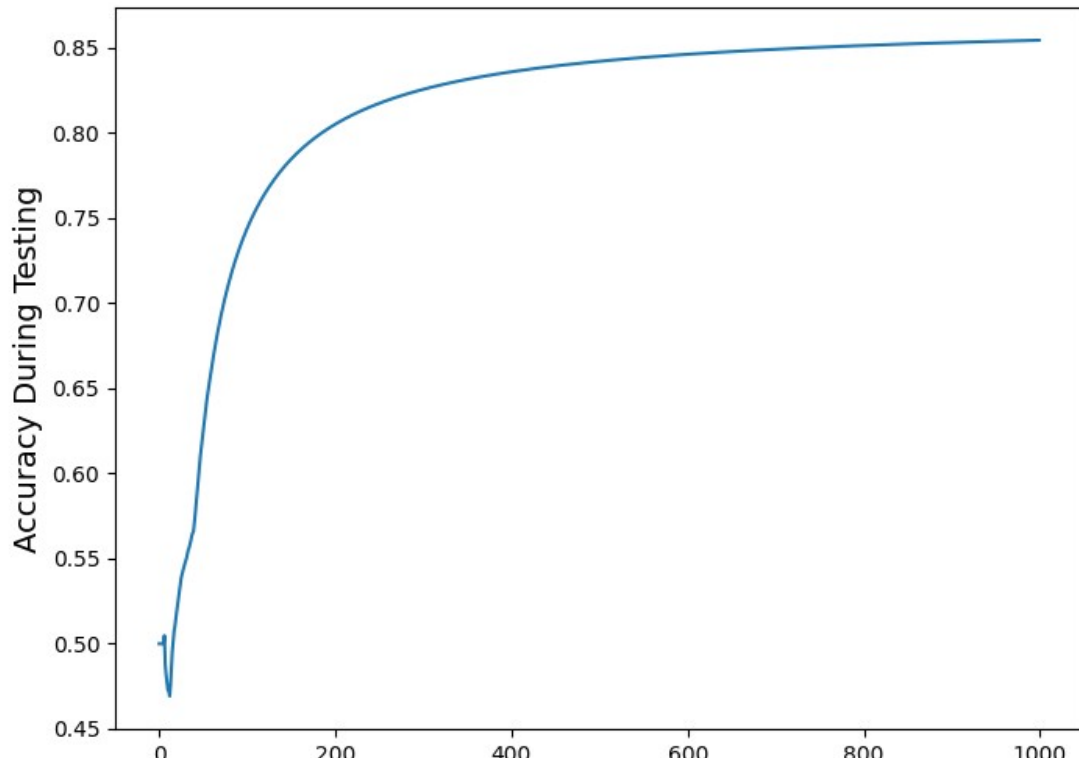
After training the model, graphs will get saved in GCN folder.  
Testing dataset has been used independently for evaluating performance

## 8. Graphs/Accuracy/Loss

As we can see that due to limited dataset, the model has been over-fitted.  
Attaching below graph for Training, validation and Testing - accuracy and loss.









## 9. Driver script

If we want to Pre-process data again, we have to empty our folders again using Terminal

```
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Training/health_filtered;
rm *.npy;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Training/health_raw;
rm *.nii;cd
/home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/Training/
patient_filtered;
rm *.npy;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Training/patient_raw;
rm *.nii;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Training/processed;
rm *.npy;
#####Testing
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Testing/health_filtered;
rm *.npy;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Testing/health_raw;
rm *.nii;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Testing/patient_filtered;
rm *.npy;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Testing/patient_raw;
rm *.nii;
cd /home/divya/development/Python/GCN/CSE6389_project2/CSE6389_project2/
Testing/processed;
rm *.npy;
```

## 10. References

[https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/Layer](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer)  
[https://www.tensorflow.org/guide/keras/transfer\\_learning](https://www.tensorflow.org/guide/keras/transfer_learning)  
<https://www.tensorflow.org/guide/autodiff>

