CHAPTER – 1:

INTRODUCTION

1.1 Aim :

Aim is to project an 3D view of moving image or a motion of a building blocks which shows how neural network looks, In this project we try to visualize this very basic framework using OpenGL. OpenGL is a computer graphics library on which many rendering and visualization softwares are built.

1.2 Overview :

Neural Network is one of the basic building blocks of Deep Learning. It is the fundamental algorithm that forms the basis of complex and advanced Deep Learning. Deep Leanring and Artificial Intelligence has shaped the modern technology and is certainly going to revolutionize the future.

1.3 Outcome :

It is a try of using computer graphics to form an moving image or a motion of a neural network in a 3D Manner. We can visualize how completely an Neural network looks in a 3D view.

CHAPTER – 2:

DESIGN AND IMPLEMENTATION

2.1 Flowchart :

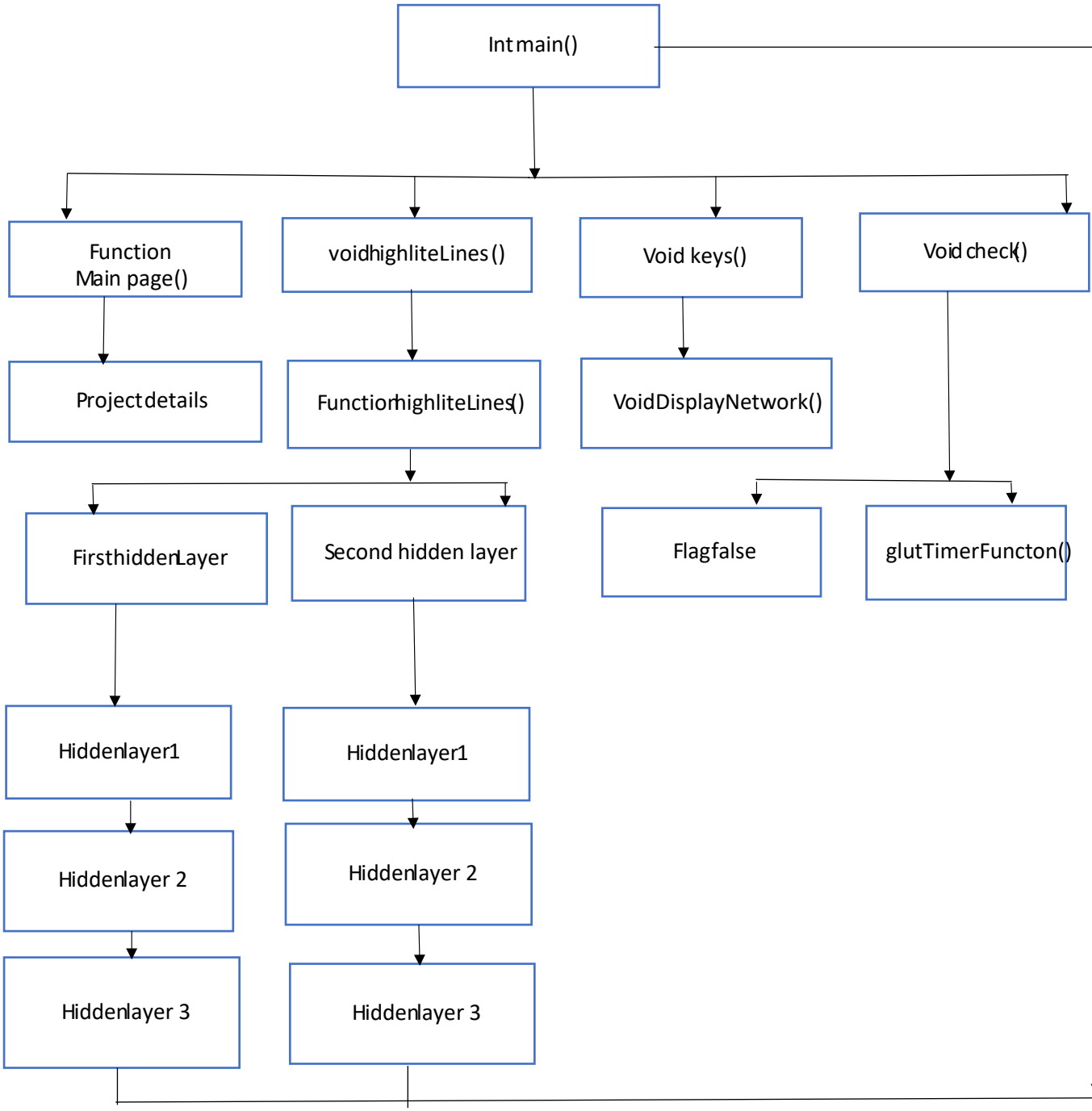

Fig 2.1.1 : The above flowchart show complete flow of how Neural Network code works

2.2 OpenGL API's Used with Description:

1. glutTimerFunc:

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds.
The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.

2. glVertex3f:

glVertex3f(x,y,z) - a vertex with 3 floating point coordinates

3. glColor4f:

- a colour of 4 floating point components

4. glBegin:

delimit the vertices of a primitive or a group of like primitives

5. glPointSize:

The function specifies the diameter of rasterized points.

6. glMatrixMode:

specify which matrix is the current matrix.

7. glClear(GL_COLOR_BUFFER_BIT):

glClear takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared. GL_COLOR_BUFFER_BIT Indicates the buffers currently enabled for color writing.

8. glLoadIdentity:

replace the current matrix with the identity matrix.

9. glTranslatef:

multiply the current matrix by a translation matrix.

10.glRotatef:

multiply the current matrix by a rotation matrix.

11.glScalef:

multiply the current matrix by a general scaling matrix.

12. glFlush:

force execution of GL commands in finite time.

13. glutSwapBuffers:

swaps the buffers of the *current window* if double buffered.

14. glutDisplayFunc:

calls a user specified function “display” whenever window needs to be drawn.

15. glutCreateWindow() :

creates a window of a pre-specified size

16. glutMainLoop():

enter an event processing loop so that graphics application continues to run & respond to user input until exited

15. glRasterPos2f:

The GL maintains a 3D position in window coordinates. This position, called the raster position, is used to position pixel and bitmap write operations. It is maintained with subpixel accuracy.

16. glutBitmapCharacter:

Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font.

17. gluPerspective:

gluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport.

18. glViewport:

The function specifies the affine transformation of *x* and *y* from normalized device coordinates to window coordinates.

19. glPolygonMode:

glPolygonMode controls the interpretation of polygons for rasterization. *face* describes which polygons *mode* applies to: front-facing polygons (GL_FRONT), back-facing polygons (GL_BACK), or both (GL_FRONT_AND_BACK). The polygon mode affects only the final rasterization of polygons. In particular, a polygon's vertices are lit and the polygon is clipped and possibly culled before these modes are applied.

20.glutReshapeFunc:

glutReshapeFunc sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established.

21.glutKeyboardFunc:

glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character.

2.3 Source Code:

```
#include <GL\glut.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include "iostream"
#include <time.h>
using namespace std;
bool flag = true;
GLfloat xRotated, yRotated, zRotated;
void check(int value)
{
    if (flag == true)
    {
        flag = false;
    }
    else if (flag == false)
    {
```

```
                flag = true;
            }
            cout << flag << "\n";
            glutPostRedisplay();
            glutTimerFunc(1000, check, 0);
        }
```

void highlight_lines(floatx,floaty,floatz,float live_transparency_line)

```
{
    float a, b, c;
    for (a = 0; a <= 0.5; a = a + 0.1)
    {
        for (b = 0; b <= 0.5; b = b + 0.1)
        {
            for (c = 0; c <= 0.5; c = c + 0.1)
            {
                //First Hidden Layer Plane 1
                glPointSize(3.0);
                glBegin(GL_POINTS);
                  glColor4f(1.0, 1.0, 1.0, 0.05);
                  glVertex3f(0.1 + b, 0.1 + c, 0.4);
                  glVertex3f(0.1 + b, 0.1 + c, 0.4);
                glEnd();
                glBegin(GL_LINE_LOOP);
                  glColor4f(1.0, 1.0, 1.0, live_transparency_line);
                  glVertex3f(x, y, z);
```

```
    glVertex3f(0.1 + b, 0.1 + c, 0.4);
    glVertex3f(0.1 + b, 0.1 + c, 0.4);
  glEnd();

  //First Hidden Layer Plane 4
  if (c < 0.44)
  {
      glPointSize(3.0);
      glBegin(GL_POINTS);
        glVertex3f(x, y, z);
        glVertex3f(0.17 + b, 0.16 + c, 0.44);
        glVertex3f(0.17 + b, 0.16 + c, 0.44);
      glEnd();
      glBegin(GL_LINE_LOOP);
        glColor4f(1.0,1.0,1.0,live_transparency_line);
        glVertex3f(x, y, z);
        glVertex3f(0.17 + b, 0.16 + c, 0.44);
        glVertex3f(0.17 + b, 0.16 + c, 0.44);
      glEnd();
  }
    //Second Hidden Layer Plane 1
    glPointSize(3.0);
    glBegin(GL_POINTS);
      glColor4f(1.0, 1.0, 1.0, 0.05);
      glVertex3f(0.1 + b, 0.1 + c, 0.8);
```

```
                glVertex3f(0.1 + b, 0.1 + c, 0.8);
            glEnd();
            glBegin(GL_LINE_LOOP);
              glColor4f(1.0, 1.0, 1.0, live_transparency_line);
                glVertex3f(0.1 + b, 0.1 + a, 0.4);
                glVertex3f(0.1 + b, 0.1 + c, 0.8);
                glVertex3f(0.1 + b, 0.1 + c, 0.8);
            glEnd();
            //output layer
            glPointSize(20.0);
            glBegin(GL_POINTS);
              glColor4f(0.0, 0.0, 1.0, 1.0);
              glVertex3f(0.2, 0.35, 1.2);
            glEnd();
            glBegin(GL_LINE_LOOP);
              glColor4f(1.0, 1.0, 1.0, live_transparency_line);
              glVertex3f(0.1 + b, 0.1 + a, 0.8);
              glVertex3f(0.2, 0.35, 1.2);
              glVertex3f(0.2, 0.35, 1.2);
            glEnd();
          }
        }
      }
    }
```

void displaynetwork(void)

```
{
        glMatrixMode(GL_MODELVIEW);
        // clear the drawing buffer.
        glClear(GL_COLOR_BUFFER_BIT);
        // clear the identity matrix.
        glLoadIdentity();
        // traslate the draw by z = -4.0
            // Note this when you decrease z like -8.0 the drawing
    will looks far , or smaller.
        glTranslatef(-1.0, -0.75, -3.5); // -1.6 for scaling of 2.3
        // Red color used to draw.
        glColor3f(0.8, 0.2, 0.1);
        glRotatef(yRotated, 0.0, 1.0, 0.0);
        glScalef(2.0, 2.0, 2.0);
        float dead_transparency_line = 0.08;
        float live_transparency_line = 0.15;
        float a, b, c;
        for (a = 0; a <= 0.5; a = a + 0.1)
        {
           for (b = 0; b <= 0.5; b = b + 0.1)
           {
              for (c = 0; c <= 0.5; c = c + 0.1)
              {
                 //Input Layer
                 glPointSize(15.0);
```

```
glBegin(GL_POINTS);
   glColor4f(1.0, 1.0, 1.0, 0.05);
   glVertex3f(0.1 + b, 0.1 + a, 0.0);
glEnd();
//First Hidden Layer Plane 1
glPointSize(3.0);
glBegin(GL_POINTS);
   glColor4f(1.0, 1.0, 1.0, 0.05);
   glVertex3f(0.1 + b, 0.1 + c, 0.4);
glEnd();
glBegin(GL_LINE_LOOP);
   glColor4f(1.0,1.0,1.0, dead_transparency_line);
   glVertex3f(0.1 + b, 0.1 + a, 0.0);
   glVertex3f(0.1 + b, 0.1 + c, 0.4);
   glVertex3f(0.1 + b, 0.1 + c, 0.4);
glEnd();
//First Hidden Layer Plane 2
if (c < 0.47)
{
   glPointSize(3.0);
   glBegin(GL_POINTS);
      glVertex3f(0.1 + b, 0.1 + a, 0.0);
      glVertex3f(0.13 + b, 0.13 + c, 0.42);
   glEnd();
   glBegin(GL_LINE_LOOP);
```

```
                glColor4f(1.0, 1.0, 1.0, dead_transparency_line);
                glVertex3f(0.1 + b, 0.1 + a, 0.0);
                glVertex3f(0.13 + b, 0.13 + c, 0.42);
                glVertex3f(0.13 + b, 0.13 + c, 0.42);
            glEnd();
        }
        //First Hidden Layer Plane 3
        glPointSize(3.0);
        glBegin(GL_POINTS);
            glVertex3f(0.1 + b, 0.1 + a, 0.0);
            glVertex3f(0.07 + b, 0.07 + c, 0.42);
            glVertex3f(0.07 + b, 0.07 + c, 0.42);
        glEnd();
        glBegin(GL_LINE_LOOP);
            glColor4f(1.0, 1.0, 1.0, dead_transparency_line);
            glVertex3f(0.1 + b, 0.1 + a, 0.0);
            glVertex3f(0.07 + b, 0.07 + c, 0.42);
            glVertex3f(0.07 + b, 0.07 + c, 0.42);
        glEnd();
        //First Hidden Layer Plane 4
        if (c < 0.44)
        {
            glPointSize(3.0);
            glBegin(GL_POINTS);
                glVertex3f(0.1 + b, 0.1 + a, 0.0);
```

```
            glVertex3f(0.17 + b, 0.16 + c, 0.44);
            glVertex3f(0.17 + b, 0.16 + c, 0.44);
        glEnd();
        glBegin(GL_LINE_LOOP);
            glColor4f(1.0, 1.0, 1.0, dead_transparency_line);
            glVertex3f(0.1 + b, 0.1 + a, 0.0);
            glVertex3f(0.17 + b, 0.16 + c, 0.44);
            glVertex3f(0.17 + b, 0.16 + c, 0.44);
        glEnd();
    }
    //Second Hidden Layer Plane 1
    glPointSize(3.0);
    glBegin(GL_POINTS);
       glColor4f(1.0, 1.0, 1.0, 0.05);
      glVertex3f(0.1 + b, 0.1 + c, 0.8);
      glVertex3f(0.1 + b, 0.1 + c, 0.8);
    glEnd();
    glBegin(GL_LINE_LOOP);
       glColor4f(1.0, 1.0, 1.0, dead_transparency_line);
       glVertex3f(0.1 + b, 0.1 + a, 0.4);
       glVertex3f(0.1 + b, 0.1 + c, 0.8);
       glVertex3f(0.1 + b, 0.1 + c, 0.8);
    glEnd();
    //Second Hidden Layer Plane 2
    if (c < 0.47)
```

```
    {
      glPointSize(3.0);
      glBegin(GL_POINTS);
        glVertex3f(0.1 + b, 0.1 + a, 0.42);
       glVertex3f(0.13 + b, 0.13 + c, 0.82);
       glVertex3f(0.13 + b, 0.13 + c, 0.82);
    glEnd();
    glBegin(GL_LINE_LOOP);
       glColor4f(1.0,1.0,1.0,dead_transparency_line);
      glVertex3f(0.1 + b, 0.1 + a, 0.0);
      glVertex3f(0.13 + b, 0.13 + c, 0.82);
      glVertex3f(0.13 + b, 0.13 + c, 0.82);
    glEnd();
}
//Output Layer
glPointSize(3.0);
glBegin(GL_POINTS);
  glColor4f(1.0, 1.0, 1.0, 0.05);
  glVertex3f(0.1 + b, 0.35, 1.2);
  glVertex3f(0.1 + b, 0.35, 1.2);
glEnd();
glBegin(GL_LINE_LOOP);
   glColor4f(1.0,1.0,1.0, dead_transparency_line);
   glVertex3f(0.1 + b, 0.1 + c, 0.8);
   glVertex3f(0.1 + a, 0.35, 1.2);
```

```
            glVertex3f(0.1 + a, 0.35, 1.2);
        glEnd();
      }
    }
  }
  if (flag == true)
  {
    //Inut image '1'
    glPointSize(15.0);
    glBegin(GL_POINTS);
      glColor4f(0.0, 0.0, 1.0, 1.0);
      glVertex3f(0.3, 0.2, 0.0);
      glVertex3f(0.3, 0.3, 0.0);
      glVertex3f(0.3, 0.4, 0.0);
      glVertex3f(0.3, 0.5, 0.0);
      glVertex3f(0.3, 0.6, 0.0);
      glVertex3f(0.4, 0.5, 0.0);
      glVertex3f(0.2, 0.2, 0.0);
      glVertex3f(0.3, 0.2, 0.0);
      glVertex3f(0.4, 0.2, 0.0);
    glEnd();
    //Highlighting the active neurons
    highlight_lines(0.3, 0.2, 0.0, live_transparency_line);
    highlight_lines(0.3, 0.3, 0.0, live_transparency_line);
    highlight_lines(0.3, 0.4, 0.0, live_transparency_line);
```

```
        highlight_lines(0.3, 0.5, 0.0, live_transparency_line);
        highlight_lines(0.3, 0.6, 0.0, live_transparency_line);
        highlight_lines(0.4, 0.5, 0.0, live_transparency_line);
        highlight_lines(0.2, 0.2, 0.0, live_transparency_line);
        highlight_lines(0.3, 0.2, 0.0, live_transparency_line);
        highlight_lines(0.4, 0.2, 0.0, live_transparency_line);
    }
    //Flushing the whole output
    glFlush();
 // sawp buffers called because we are using double buffering
    glutSwapBuffers();
}
void keys(unsigned char key, int x, int y)
{
    if (key == 'x')
    {
        glutDisplayFunc(displaynetwork);
    }
    glutPostRedisplay();
}
void print(int x, int y, string str1, void* font)
{
            int length, k;
            glRasterPos2f(x, y);
            length = str1.length();
```

```
        k = 0;
        while (k < length) {
            glutBitmapCharacter(font, str1[k]);
            k++;
    }
}
void mainpage()
{
    glColor3f(0.000, 0.749, 1.000);
    print(33, 500, " MAHARAJA INSTITUTE OF TECHNOLOGY ",
    GLUT_BITMAP_TIMES_ROMAN_24);
    print(23,486,"(Affiliated to Visvesvaraya Technological University,
    Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE,
    New Delhi) ", GLUT_BITMAP_HELVETICA_12);
    print(40,470,"MANDYA,SRIRANGPATNA",GLUT_BITMAP_H
    ELVETICA_18);
    print(30, 400, " DEPARTMENT OF COMPUTER SCIENCE AND
    ENGINEERING ", GLUT_BITMAP_HELVETICA_18);
    glColor3f(1, 1, 1);
    print(43,300," SHELL SORTING ",
    GLUT_BITMAP_TIMES_ROMAN_24);
    glColor3f(0.000, 0.749, 1.000);
    glColor3f(0, 1, 1);
    print(65, 175, " By: ", GLUT_BITMAP_HELVETICA_18);
    print(65,150,"KARAN .M.N - 4MH19CS037",
    GLUT_BITMAP_HELVETICA_18);
```

```
    print(65,120,"K.M.CHAITHRASHREE - 4MH18CS039",
    GLUT_BITMAP_HELVETICA_18);
    print(5,120,"Press X key to continue!!!",
    GLUT_BITMAP_TIMES_ROMAN_24);
}

void reshapenetwork(int x, int y)
 {
    if (y == 0 || x == 0) return;
    //Nothing is visible then, so return
    //Set a new projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //Angle of view:40 degrees
    gluPerspective(40.0, (GLdouble)x / (GLdouble)y, 0.5, 20.0);
    glViewport(0, 0, x, y);
    //Use the whole window for rendering
}
void idlenetwork(void)
{
   yRotated += 0.05;
   displaynetwork();
}
int main(int argc, char** argv)
{
```

```
        //Initialize GLUT
        glutInit(&argc, argv);
        //double buffering used to avoid flickering problem in animation
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        // window size
        glutInitWindowSize(1350, 950);
        glutInitWindowPosition(0, 0);
        // create the window
        glutCreateWindow("network Rotating Animation");
        glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
        yRotated = 40;
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        //Assign  the function used in events
        glutDisplayFunc(mainpage);
        glutKeyboardFunc(keys);
        glutReshapeFunc(reshapenetwork);
        glutIdleFunc(idlenetwork);
        //Let start glut loop
        glutTimerFunc(100, check, 0);
        glutMainLoop();
        return 0;
}
```

CHAPTER – 3
RESULT ANALYSIS

3.1 SnapShots:

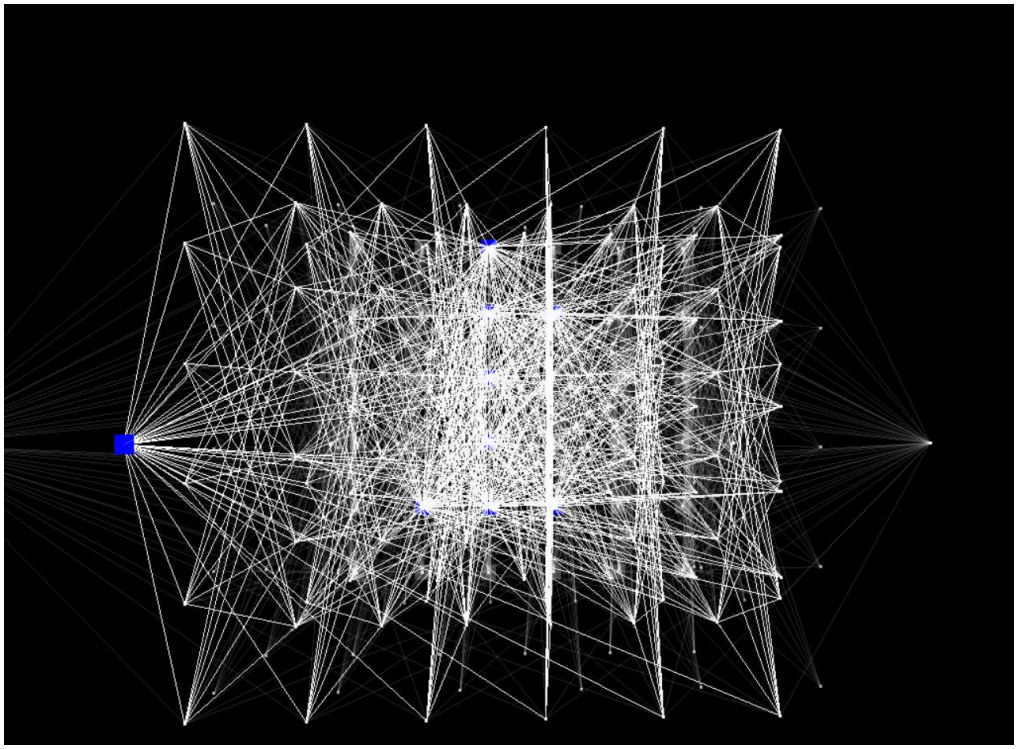

Fig 3.1.1 : This figure shows 1st angle of Neural Network where the internal connections are shown

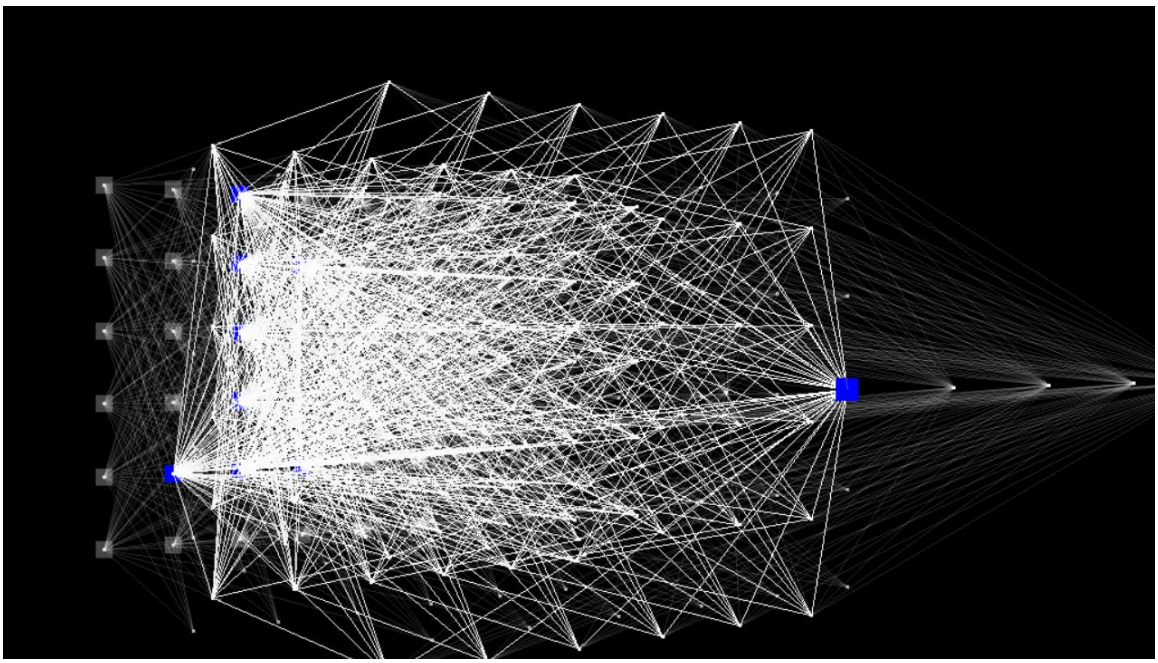

Fig 3.1.2: This figure shows the 2nd angle of Neural Network where the behide points are shown

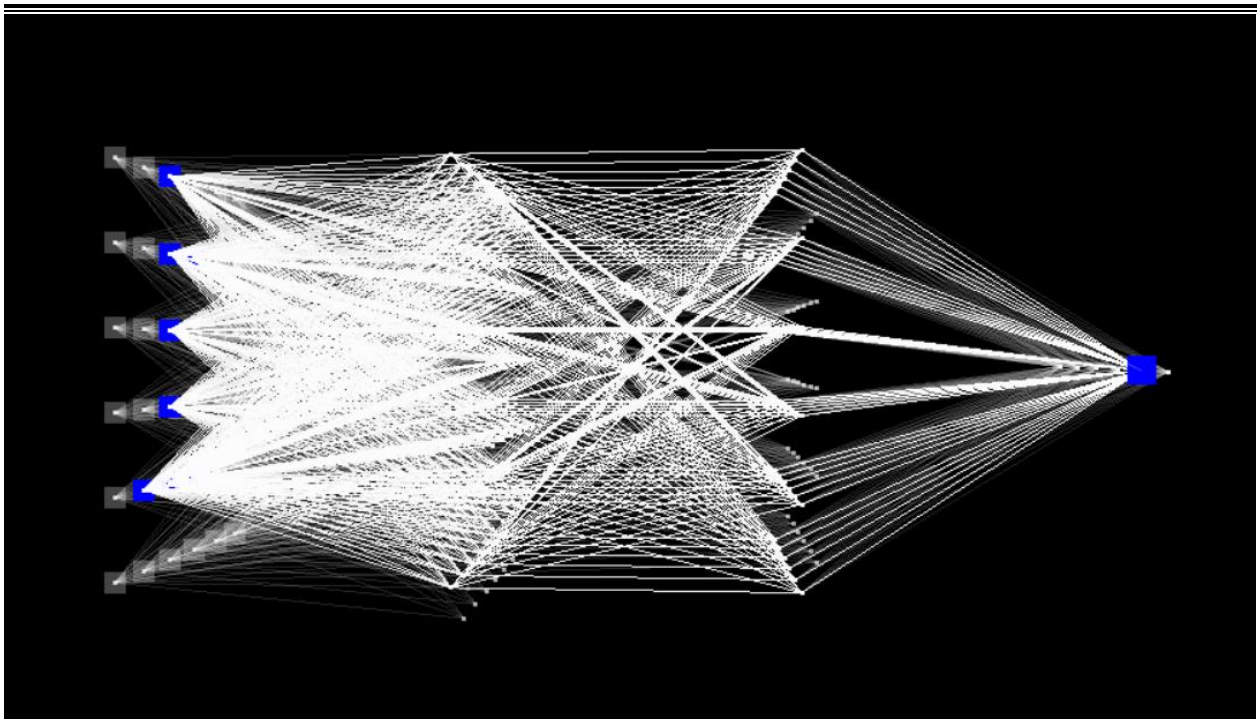

Fig 3.1.3 : This figure the 3rd angle of the Neural Network where side angle is shown.

The right side corner point is shown

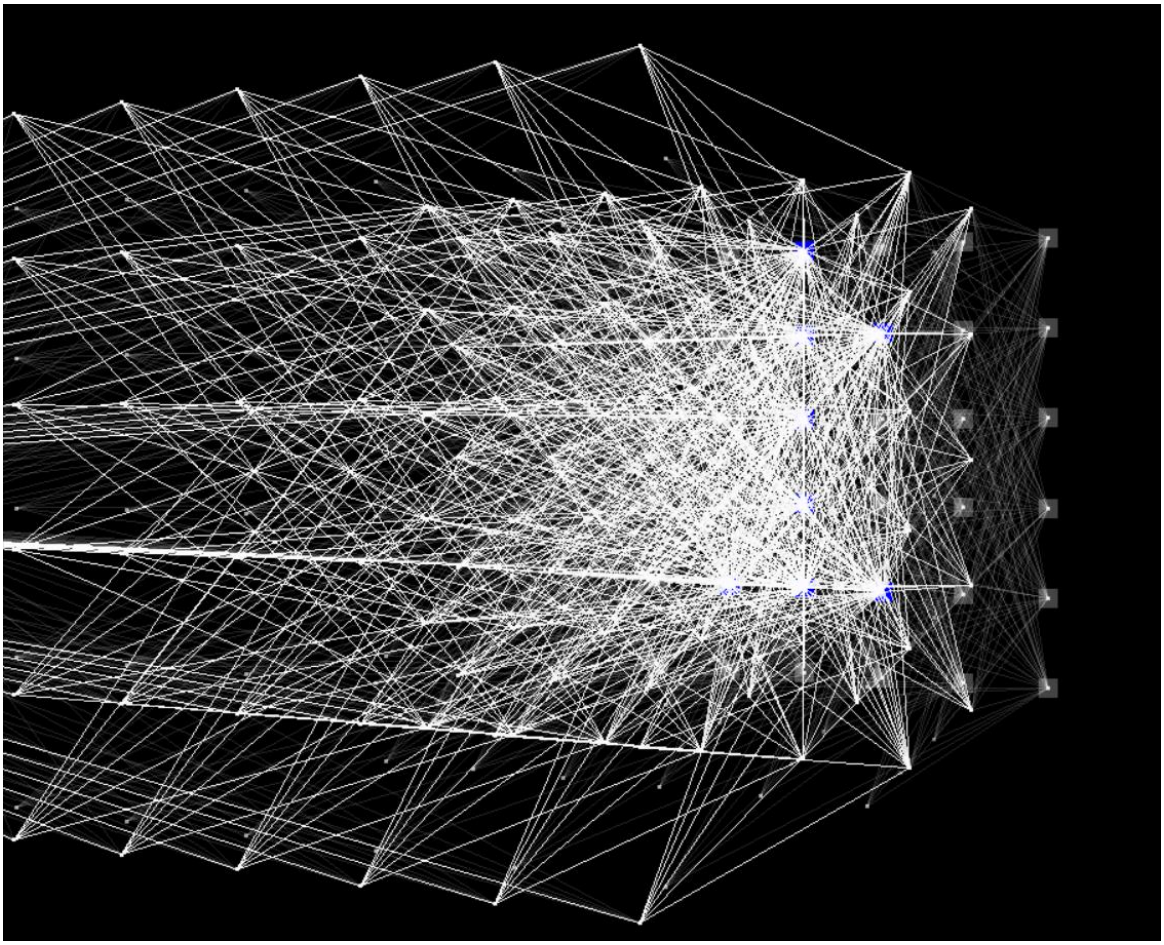

Fig 3.1.4: This figure show the 4th angle of Neural Network and It shows the internal connected nodes.

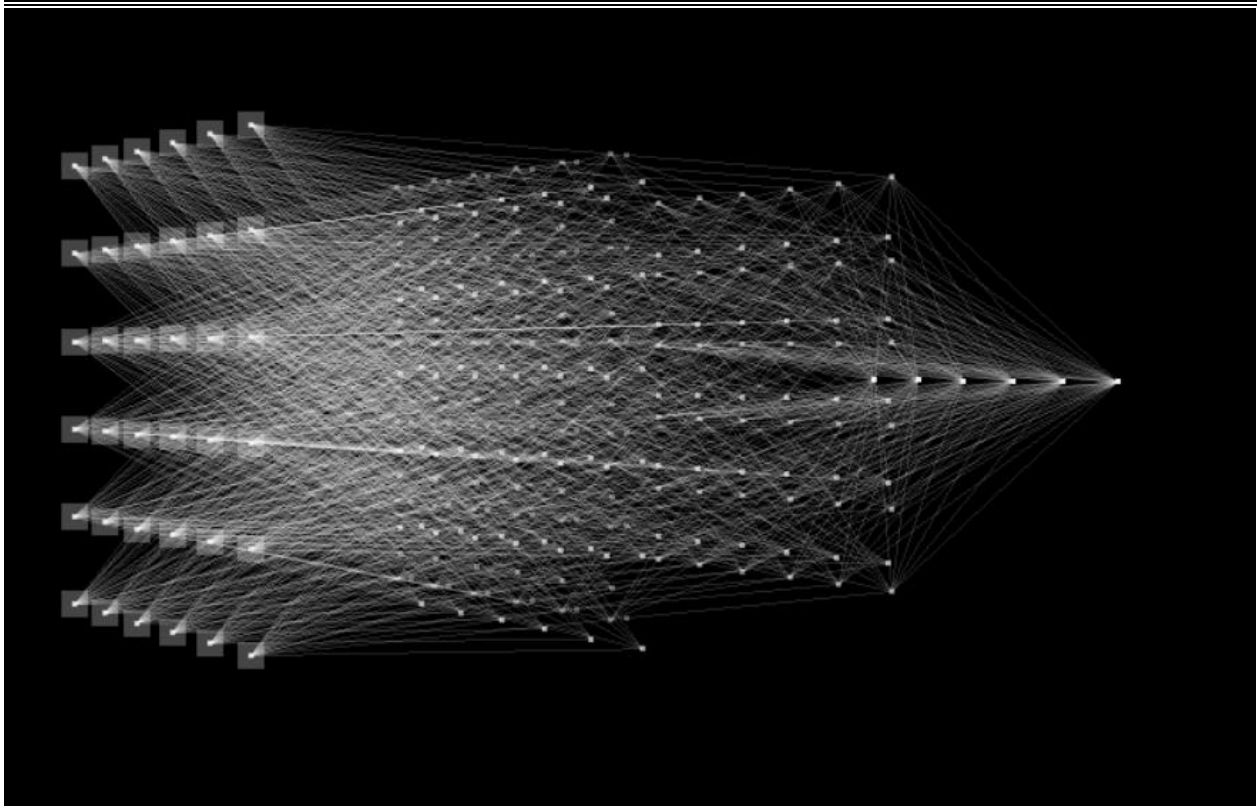

Fig 3.1.5: This figure shows 5th angle of Neural Network where it shows points of connected nodes behind the Neural Network

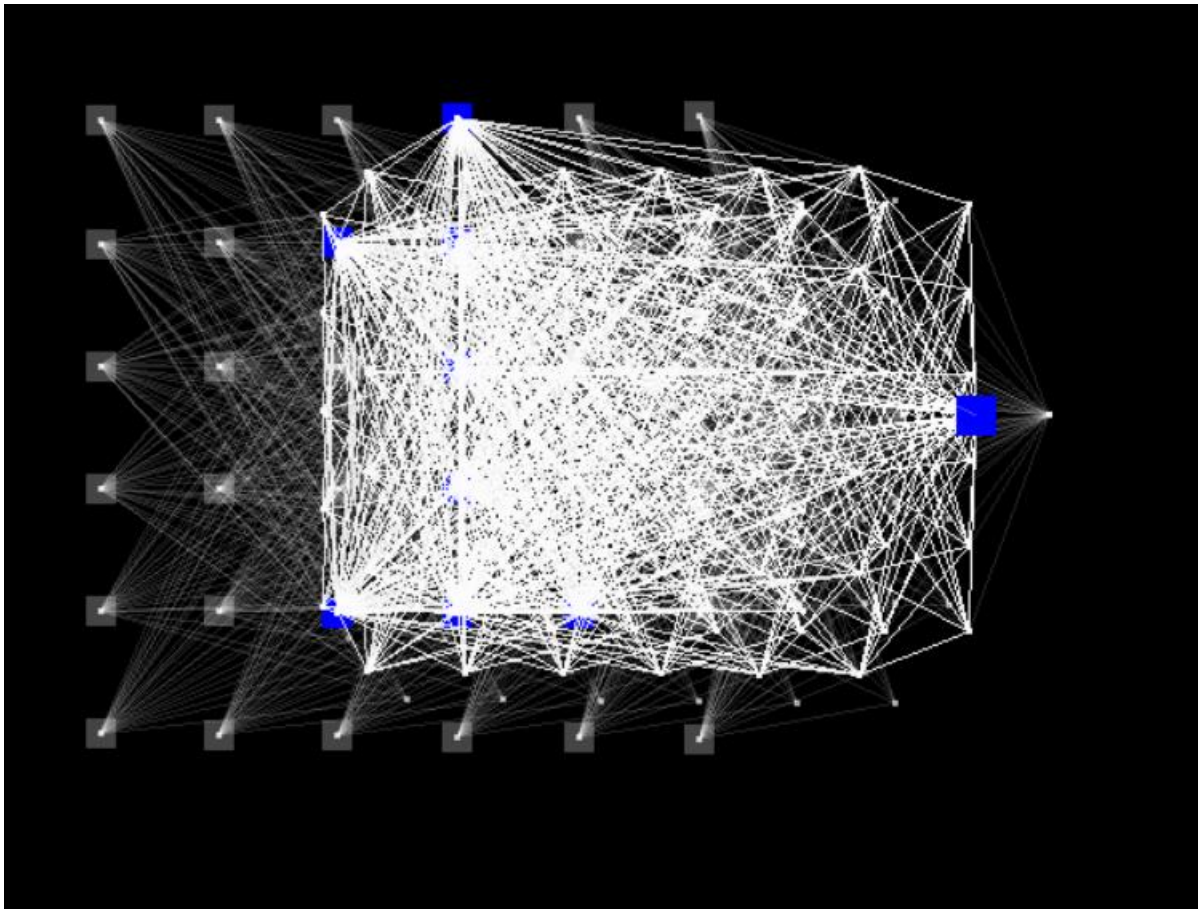

Fig 3.1.6: This figure shows the 6th angle of Neural Network where it shows side nodes of Neural Network

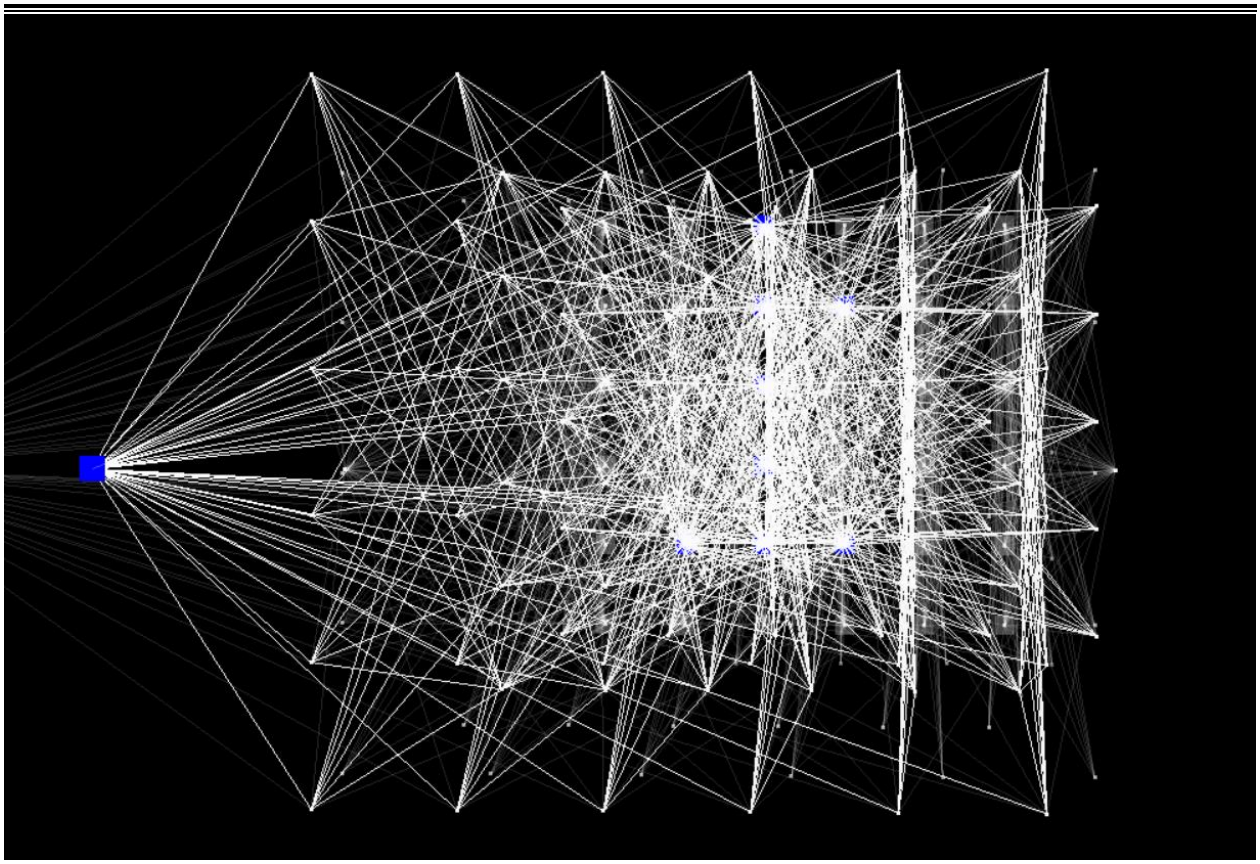

Fig 3.1.7 This figure shows the 7th angle of Neural Network where it another connected lines of Neural Network

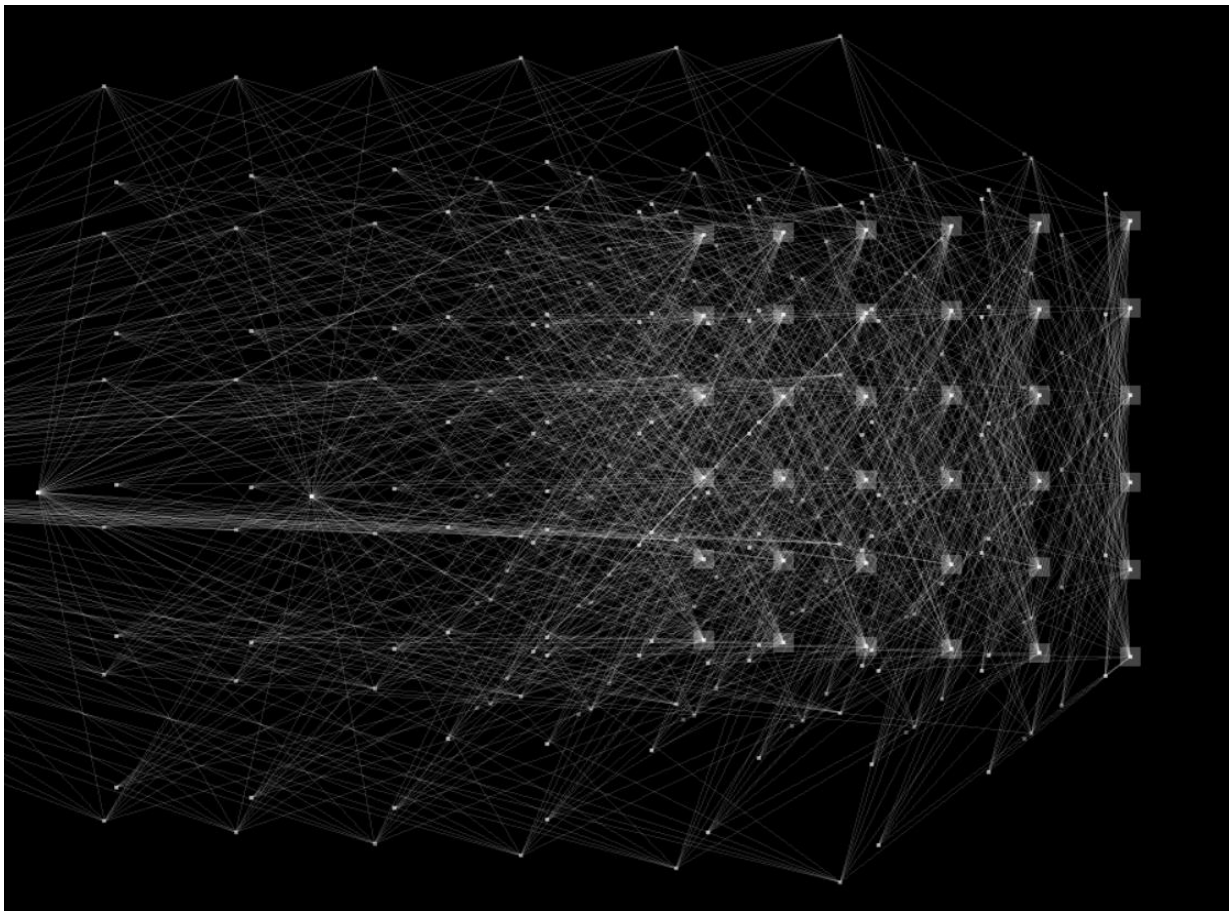

Fig 3.1.8: This figure shows the 8th angle of Neural Network where we can see every small nodes of Neural Network

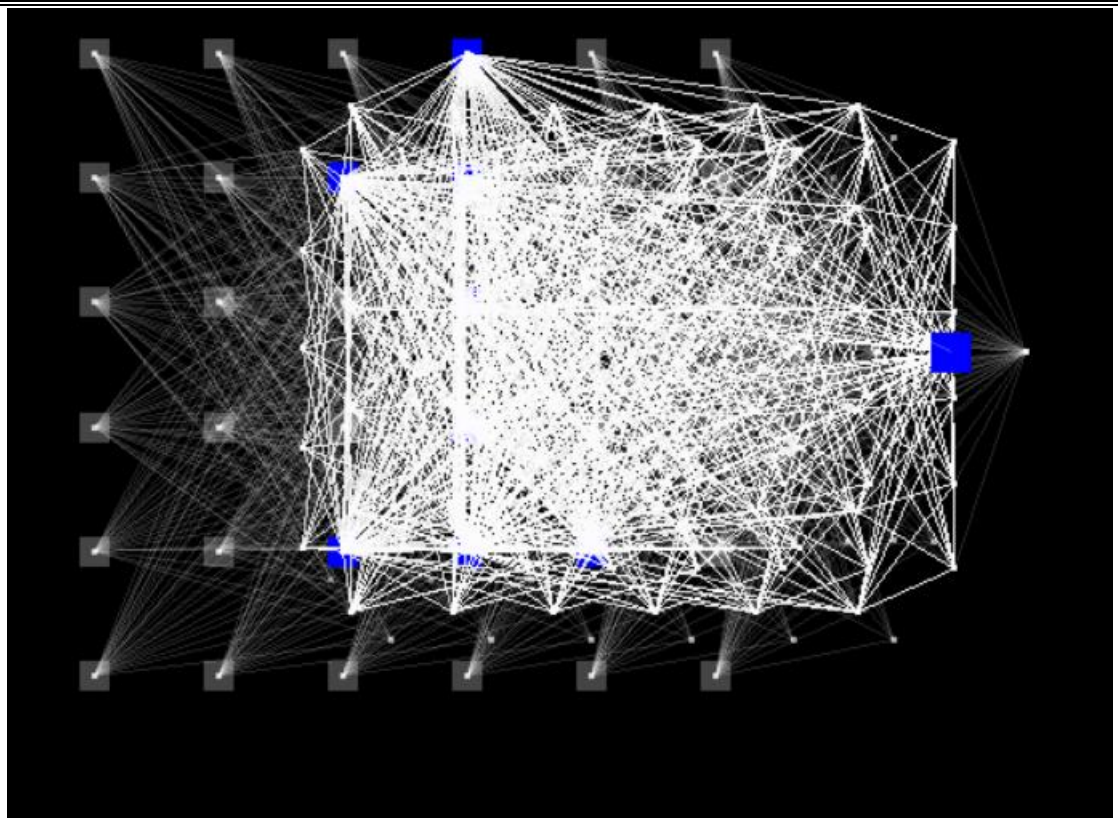

Fig 3.1.9: This figure shows the 9th angle of Neural network

3.2 Discussion :

It is to be discussed that how Neural Network can be build using OpenGL . Where it shows complete 3D Projection of Building Blocks of Neural Network Works by 360 degree rotating motion.

It is the fundamental algorithm that forms the basis of complex and advanced Deep Learning. It has shaped the modern technology and is certainly going to revolutionize the future.

Neural networks can help computers make intelligent decisions with limited human assistance. This is because they can learn and model the relationships between input and output data that are nonlinear and complex.

CHAPTER – 4:

CONCLUSION AND FUTURE WORK

4.1 Conclusion:

We have made a Neural Network having 2 hidden layers. The Input is highlighted with a 6x6 image of the digit '1'.

The highlighted cell in the output shows that the image was of the digit '1'.

The model is constantly rotating in order to help better visualize the Neural Network.

4.2 Future Enhancement:

- ☑ Improve the existing Visualization and introduce keybpard/mouse control.
- ☑ Make the model flexible, i.e. taking user input for the hidden layers, input and output sizes.
- ☑ Make the visualization on the basis of a real time Neural Network
- ☑ Create library for the extended versions like CNNs, RNNs, GNNs, etc.

CHAPTER – 5

REFERENCES

1. OpenGL Documentation - https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/
2. Transformations in Opengl - https://learnopengl.com/Getting-started/Transformations
3. Blending in OpenGL - https://learnopengl.com/Advanced-OpenGL/Blending
4. Inspiration - https://youtu.be/3JQ3hYko51Y
5. James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: pearson education