DEPARTMENT OF COMPUTER SCIENCE

CSC I1910 4GH [33559]

FINAL PROJECT

DEEP NEURAL NETWORKS AND APPLICATIONS WITH TENSORFLOW

Guided by: Dr. Michael Grossberg

Submitted By:
Nikita Acharya
Saurav Pradhan
Vedika Saravanan

Spring 2021

ACKNOWLEDGEMENT:

First and foremost, we would like to thank our professor **Dr. Michael Grossberg,** who gave us opportunity to explore a lot regarding Deep Neural Network using Tensorflow and Keras. This study would not have been possible without the guidance, thoughts, suggestions and feedbacks from our professor.

INDEX

LIST OF FIGURES

LIST OF ABBREVIATIONS

TERMS	ABBREVIATION
BIST	*Built-In Self Test*
CAD	*Computer Aided Design*
CC	*Combinational Controllability*
CC0	*Combinational Controllability of 0*
CC1	*Combinational Controllability of 1*
CNN	*Convolutional Neural Network*
CO	*Combinational Observability*
DFT	*Design For Test*
DNN	*Deep Neural Network*
EO	*Easy to Observe*
GCN	*Graph Convolutional Neural Network*
HO	*Hard to Observe*
IC	*Integrated Circuits*
LFSR	*Linear Feedback Shift Register*
LL	*Logic Level*
ML	*Machine Learning*
PI	*Primary Input*
PO	*Primary Output*
RAM	*Random Access Memory*
ROSE	*Random Over Sampling Examples*
SCOAP	*Sandia Controllability/Observability Analysis Program*
SMOTE	*Synthetic Minority Over-sampling Technique*
TAGCN	*Topology Aware Graph Convolutional Neural Network*
TG	*Type of Gate*
ULSI	*Ultra Large-Scale Integration*
VLSI	*Very Large-Scale Integration*
WG	*Weight of Gates*

ABSTRACT

In the field of Very Large-Scale Integration (VLSI), the complexity of digital circuits increases, and are moving towards nano scaled structures. Since the size of the VLSI circuits are getting smaller, the most noticeable problem is that how to efficiently test these circuits. There are several Design-for-Test (DFT) tools that provides approximate solution rather than accurate solution. This was the reason researchers started to explore various machine learning (ML) based approaches have been proposed to tackle the problem of scalability in testing electronic design automation. Since the circuit is a data dependent graph it is hard for ML based models to generalize this type of problem. Therefore, we exploit several Deep Neural Network (DNN) models to provide efficient way of testing and also to reduce the area and time required for testing VLSI circuits using design-based tools.

INTRODUCTION

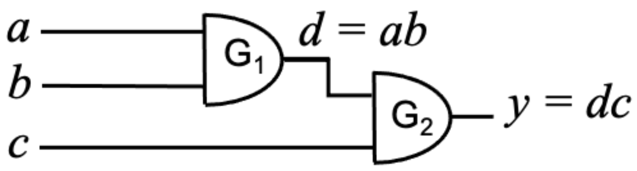
Today's VLSI industry is moving towards Ultra Large-Scale Integration (ULSI). The more the complex is the design, the less the cost per logic gates along with improvement in the performance. But the issue of deciding whether a chip has been tested efficiently at the same time maintaining the quality of the chip. It is believed that prudent testing must be acquired if a testing procedure is received during the underlying chip design. Design-for-Test (DFT) has been utilized to demonstrate those design methods used to improve chip testability. The majority of the Design-for-Test methods endeavor to improve the recognizability and controllability of a circuit design. Three significant types of DFT are the ad-hoc approaches, the structured approaches, and the built-in self test (BIST) approaches. The ad-hoc techniques can be applied to a given design yet are not aimed at solving testability as a rule. Ad-hoc techniques have been a compelling method to execute testability without causing significant error overhead. In any case, they have the hindrance of relying vigorously upon the ability of the designer. The most popular approach of ad hoc technique is the test point insertion approach that is widely used in VLSI design to improve testability. This methodology includes embedding test nodes to increase both controllability and observability. Whereas the structured techniques tries to solve the general test problems for a design. It incorporates complex CAD tools to automatically generate test patterns along with the design rules while designing a chip. The extension of the structured approach is BIST. The goal is to develop a chip that can perform testing by itself if sufficient clock cycles are applied. BIST are used to self test RAM in which Linear Feedback Shift Registers (LFSR) are used to create address and data patters automatically for testing RAM.

As the complexity of integrated circuits (IC) increases it is more challenging throughout the design flow to achieve scalability during test and verification. The motivation is to reduce the area cost and time required for testing larger circuits. There are tools that gives approximate solution but not exact solution. Several ML based techniques are proposed to overcome these challenges. The main disadvantage of ML based learning is that it will not fully leverage the structure of the circuit which leads to miss prediction of the output. Since the VLSI circuits are represented as directed acyclic graph of netlist, thus, we explore different DNN models like Graph Convolutional Network (GCN), Chebysheb Convolutional Network, Topology Aware Graph Convolutional Network (TAGCN) and Graph Sage Convolutional Network which leverage the node feature and graph structure to learn the hidden representation and improve the prediction of testability.

1. BACKGROUND

1.1 Very Large Scale Integration (VLSI) Circuits

The circuits can be widely classified into combinational and sequential circuits. In our project we focus only on combinational circuits since it is faster in execution and easier to design compared to sequential. In combinational circuits the output relies on the condition of the most updated inputs. For example, a combinational circuit shown in Figure 1.1(a). consist of two AND gates, the inputs to the circuit are *a, b, c* and the outputs are *d,* and *y*. The corresponding truth table for the circuit is shown in Figure 1.1(b). A truth table is used to represent the logic of the circuit in connection to the Boolean function. This table will help us during testing process to check the functionality of the circuit. For *n* bits, the truth table consists of 2^n combinations of input patterns. These patterns are used during the testing process to check whether the output is correct or not for the given logic (Boolean function).



(a)

Input			*Output*	
a	*b*	*c*	*d*	*y*
0	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*0*
0	*1*	*0*	*0*	*0*
0	*1*	*1*	*0*	*0*
1	*0*	*0*	*0*	*0*
1	*0*	*1*	*0*	*0*
1	*1*	*0*	*0*	*0*
1	*1*	*1*	*1*	*1*

(b)

Fig. 1.1: (a) Combination VLSI circuit and (b) Truth Table of the circuit

1.2 Stuck-at-Faults

Stuck-at-Faults is a structural test approach, in which certain set of test vectors are used for testing rather than considering all the combinations of 0's and 1's of the circuit. The faults can be either 0 or 1 and referred as Stuck-at-Zero (s-a-0) or Stuck-at-One (s-a-1), respectively. In order to explain in detail about these faults let us consider the same example as shown in Figure 1.2. Let us assume that input *d* is s-a-0, now let us look at the truth table and see which combination of the input test vector will help us to propagate the fault in the node '*d*'.

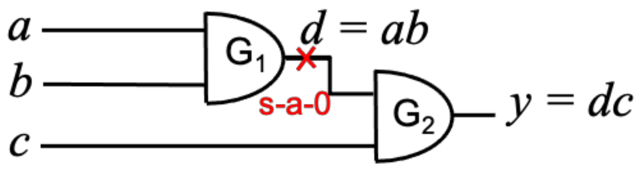


Fig. 1.2 : Input *d* is stuck-at-zero (s-a-0)

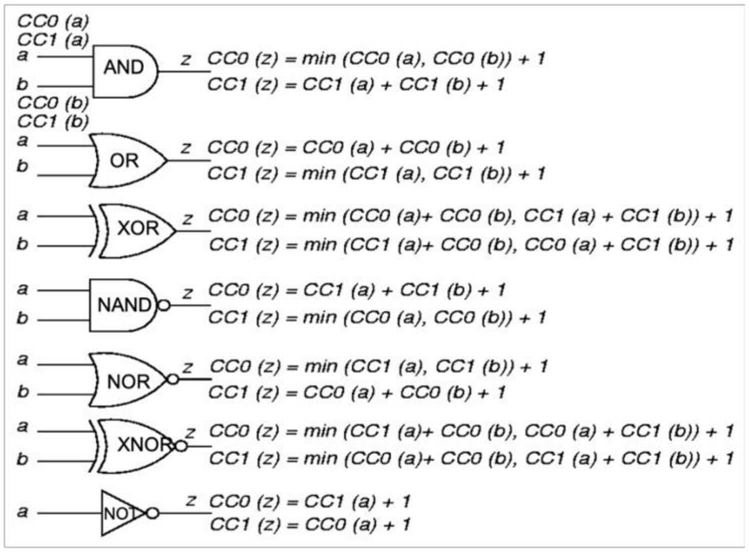
Since the gate associated with the node '*d*' is AND gate if and only if both the inputs '*a*' and '*b*' holds a value of 1, the s-a-0 fault can be propagated to the output and can identify that the node '*d*' is defective. Similarly, test input vectors can be used for all the nodes (inputs and outputs). For smaller circuits it is easy to find the faults but when it comes to larger circuits the input combination is going to be high and it makes it difficult to propagate the error in the circuit. Hence SCAOP metric was discovered to evaluate a logic circuit.

1.3 Sandia Controllability/Observability Analysis Program (SCAOP)

To evaluate the difficulty of controlling and observing the circuit, SCAOP a testability metrics was developed [1]. In a circuit, each input, fanout and output can have testability metrics like controlling and observing either 0 or 1 on it. Fanout is defined as the number of inputs that are connected to a specific output.

Combinational Controllability (CC)

In controllability the Primary Input (PI) is always 1. After propagating through each gate in the circuit, the value of CC increases due to the presence of gate and is incremented by 1. CC can be 1 or 0, if CC is 1 which means that the node is s-a-0 and vice-versa for CC = 0. Each gate in VLSI circuit holds its own equation to compute controllability. The CC for all gates are shown in Figure 1.3.



CC0 (a)
CC1 (a)
a
b
CC0 (b)
CC1 (b)
AND
z
CC0 (z) = min (CC0 (a), CC0 (b)) + 1
CC1 (z) = CC1 (a) + CC1 (b) + 1
OR
CC0 (z) = CC0 (a) + CC0 (b) + 1
CC1 (z) = min (CC1 (a), CC1 (b)) + 1
XOR
CC0 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
CC1 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
NAND
CC0 (z) = CC1 (a) + CC1 (b) + 1
CC1 (z) = min (CC0 (a), CC0 (b)) + 1
NOR
CC0 (z) = min (CC1 (a), CC1 (b)) + 1
CC1 (z) = CC0 (a) + CC0 (b) + 1
XNOR
CC0 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
CC1 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
NOT
CC0 (z) = CC1 (a) + 1
CC1 (z) = CC0 (a) + 1

Fig. 1.3 : Combinational Controllability of logic gates

From the figure, it is clear that CC increases from the PI to the output. The more the CC increases, the more the line is difficult to hold a 0 or 1. In other words, the higher the CC it is harder to detect and propagate the fault in the circuit.

Combinational Observability (CO)

Combinational Observability is another metric used to observe a line of the circuit. In CO, the primary outputs (PO) are easily observable as the lines are all outputs. Similar to CC of PI, CO of PO's will always be 0. The observability value is determined by backtracking from the PO's to the inputs. The CO for all gates is shown in Figure 1.4. The higher the CO the more difficult is to find the value at that line of the circuit.

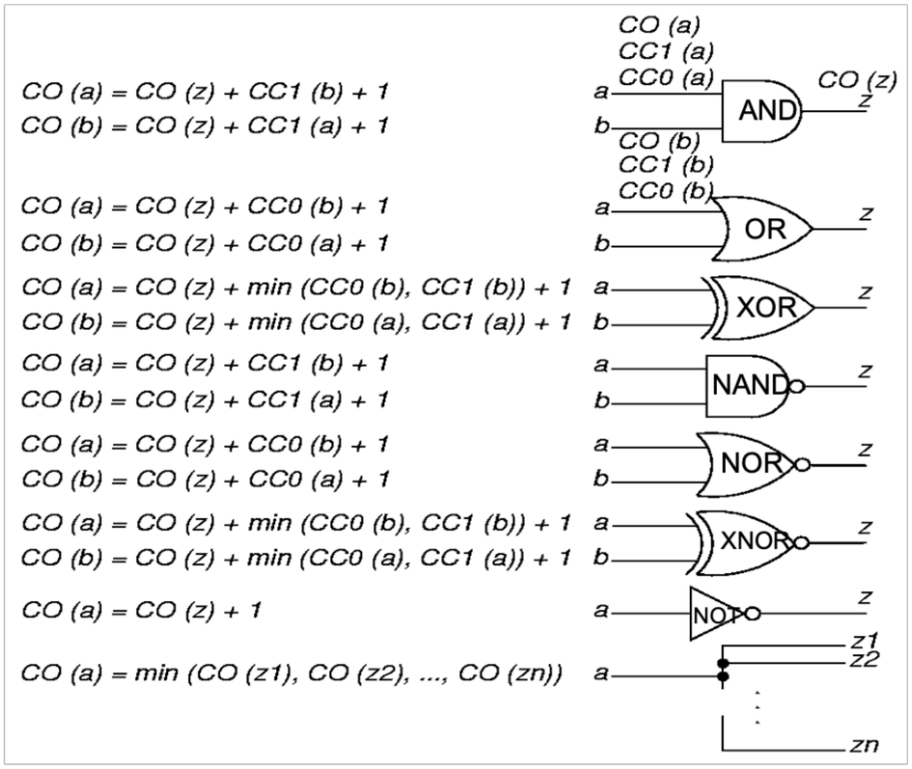

CO (a) = CO (z) + CC1 (b) + 1
CO (b) = CO (z) + CC1 (a) + 1
CO (a) = CO (z) + CC0 (b) + 1
CO (b) = CO (z) + CC0 (a) + 1
CO (a) = CO (z) + min (CC0 (b), CC1 (b)) + 1
CO (b) = CO (z) + min (CC0 (a), CC1 (a)) + 1
CO (a) = CO (z) + CC1 (b) + 1
CO (b) = CO (z) + CC1 (a) + 1
CO (a) = CO (z) + CC0 (b) + 1
CO (b) = CO (z) + CC0 (a) + 1
CO (a) = CO (z) + min (CC0 (b), CC1 (b)) + 1
CO (b) = CO (z) + min (CC0 (a), CC1 (a)) + 1
CO (a) = CO (z) + 1
CO (a) = min (CO (z1), CO (z2), ..., CO (zn))
CO (a)
CC1 (a)
CC0 (a)
CO (z)
CO (b)
CC1 (b)
CC0 (b)
AND
OR
XOR
NAND
NOR
XNOR
NOT
a
b
z
z1
z2
zn

Fig. 1.4 : Combinational Observability of logic gates

1.4 Related Work

From the SCOAP measures, we can infer that if CC is small and CO is high it makes hard the process of testing. There are several approaches that tries to train a model to predict the ground truth labels (hard-to-observe (HO) and easy-to-observe (EO)). A GCN based model [2] was proposed to examine the state-of-the-art DL for IC test and highlights two critical unaddressed challenges. The first challenge involves identifying fit-for-purpose statistical metrics to train and evaluate ML model performance and usefulness in IC test.

DNN are used to detect the hotspots in a physical layout of the design in lieu of lithograph simulation [3]. The routability of the design can be estimated using DNN [4]. The approximate function is a well know capability of deep learning that provides fast test turnaround when compared to analytic and heuristics techniques [5],[6]. To make use of these, an engineer in testing has to map the integrated circuit test issues to machine learning based problems which will eventually require deep understanding of IC test area to select exact features from the design to train the models. Else the model will provide poor prediction.

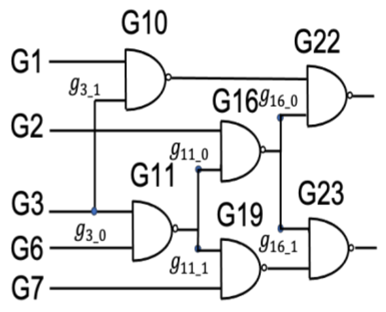
A scalable semi supervised graph-based learning approach was proposed in [7]. It is based on structured variant of CNN that runs directly on the graphs. The idea here is to use spectral graph convolutional network based on localized first order approximation to choose the convolutional model.

To leverage the feature information of the nodes and to systematically generate the embeddings of the node from previous unseen data, a inductive framework called GraphSAGE [8] is proposed. In this model, the embeddings are generated by sampling and aggregating the features from the local neighborhood of the nodes rather than training individual embedding for each node.

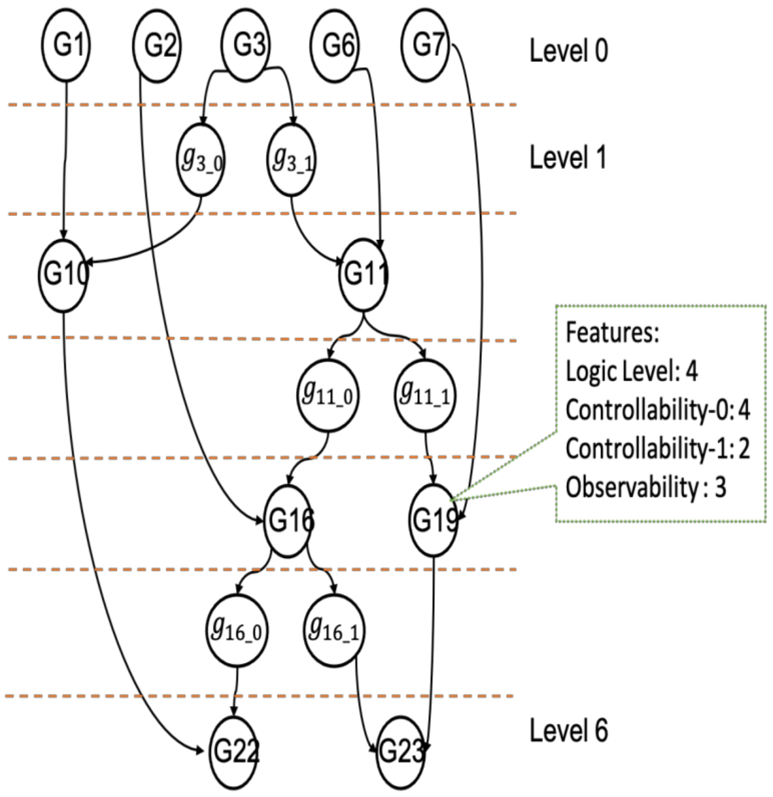
A novel GCN model was proposed in the vertical domain known as topology adaptive graph convolutional network (TAGCN) [9]. To design a set of fixed size learnable filters this model provides a systematic way to carry out convolutions on graphs. In this model no approximating of convolution is required hence TAGCN provides better than already available spectral convolutional neural network

2. DATA EXTRACTION

The main data source is the benchmark circuits. We downloaded the publicly available benchmark circuits. The circuits that we worked on were ISCAS '85 [10]: c1, c5, c17, c432, c499, c880, c1355, c1908, c2670, c3540, c5315, c6288, c7552 and the EPFL Combinational Benchmark Suite [11]: Adder, arbiter, cavlc, ctrl, dec, i2c, int2float, max, multiplier, priority, router, sin, voter. The researchers who work on DFT of VLSI circuits makes use of these benchmark circuits to evaluate their approaches. This dataset has 240587 nodes. We were not able to have access to the industrial DFT tool that would report the circuit nodes that are hard-to-observe and easy-to-observe (ground truth), still we were able to find and extend a publicly available tool to extract those information from these circuits. Initially we extracted features such as type of the gate (TG), weight of the gate (WG), Logic Level (LL), Combinational Controllability of 0 (CC0), Combinational Controllability of 1 (CC1) and Combinational Observability (CO). But controllability measures itself will provide unique information for each type of gates based on the equation shown in Figure 2 and also the WG feature is useful when the gates location is altered which is our motivation. Therefore, we dropped TG and WG features and considered the four-dimensional features of each node of the circuit which are the LL, CC0, CC1 and CO, and the ground truth labels which are hard-to-observe (HO) and easy-to-observe (EO) nodes. We extended the tool to extract the features for each node in the circuit and save the circuits graph, input features and the labels as circuit.graph, circuit.x and circuit.y pickle files which are preprocessed later and fed into the model. Let us consider an example of c17 benchmark circuit shown in Figure 2.1 (a) and the graphical representation of the features of each node and adjacency matrix of the circuit is shown in Figure 2.1 (b) and 2.1 (c).



(a)



(b)

$$\begin{array}{c|ccccccccccccccccc}
 & \text{G1} & \text{G2} & \text{G3} & \text{G6} & \text{G7} & \text{g\_3\_0} & \text{g\_3\_1} & \text{G10} & \text{G11} & \text{g\_11\_0} & \text{g\_11\_1} & \text{G16} & \text{G19} & \text{g\_16\_0} & \text{g\_16\_1} & \text{G22} & \text{G23} \\
\text{G1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{G2} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{G3} & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{G6} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{G7} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{g\_3\_0} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{g\_3\_1} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{G10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{G11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{g\_11\_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{g\_11\_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{G16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\text{G19} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\text{g\_16\_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
\text{g\_16\_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
\text{G22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{G23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

(c)

Fig. 2.1 : (a) c17 benchmark circuit, (b) directed acrylic graph of c17 and (c) Adjacency matrix for c17 with self-loops.

3. METHODS

In this section we discuss about pre-processing of the data which is obtained from the tool and we try different DNN models along with different ML models.

3.1 Data Preprocessing

After extracting the circuits information as a pickle file, we loaded all the circuit data and looked into the input distribution for the circuits. There was a huge range of data for all the feature columns when considering all the circuits as shown in Figure 3.1.

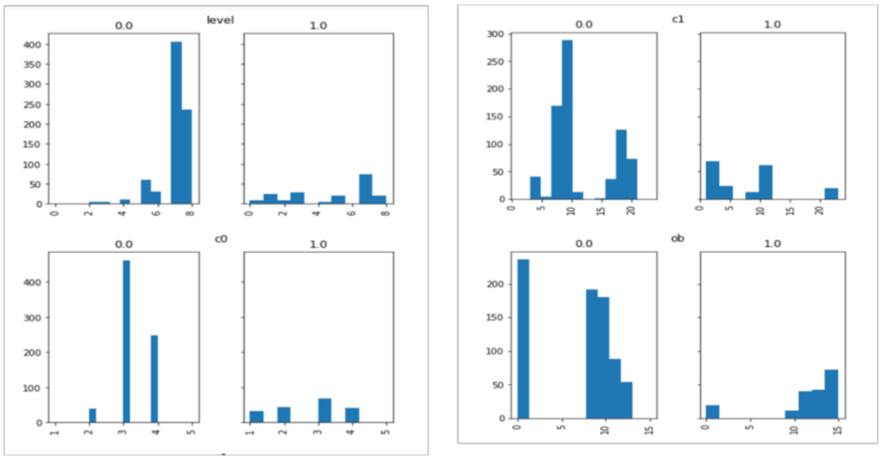


Fig. 3.1 : Raw dataset nodes features (level, c0, c1 and Observability) distribution based on the output labels for a circuit before normalization

So, we normalized the whole set of input features using MinMax Scaler as shown in Figure 3.2. We generated the adjacency matrix of the graph and converted it to the compressed sparse matrix such that it reduces memory allocation while loading the dataset to feed into the model.

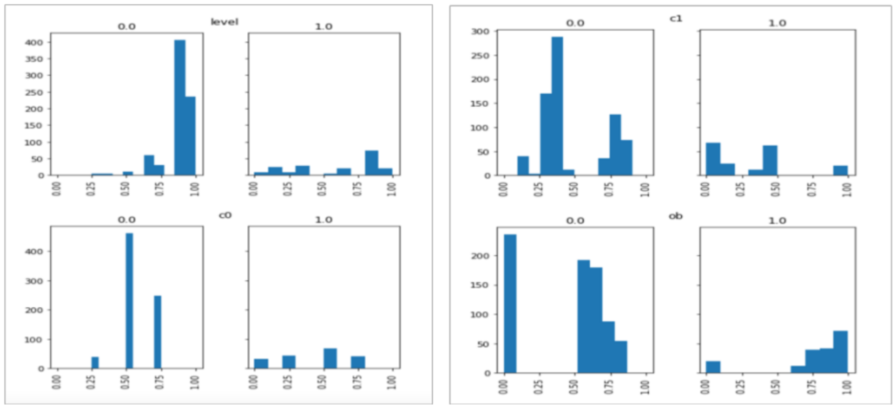


Fig. 3.2 : Nodes features distribution based on the output labels for a circuit after normalization

From the Figure 3.2, we infer that the dataset is unbalanced (EO has more nodes compared to HO), this might be a problem while training the model. This was one of the challenges for us. Initially we had started with one circuit, but we wanted to train with different circuits such that we can have more data points and reduce the unbalanced dataset as well.

We then started looking into the way of combining all the circuits together as a disjoint graph and feed to the model. But the problem on doing so was unlike images, or other datasets which had symmetric datasets, we were dealing with an asymmetric circuit graph which is also one of the important input features to the model for the aggregation of the information for each node based on the neighbor nodes. We could not reshape, cut such that we can fit it to the tensor of a pre-defined shape. So, we needed a way to define the data loader such that it considers the max size adjacency matrix and add the padding to other circuits to generate a set of disjoint graphs and load it as a batch for training, validation and test dataset. Also, most of the examples that were utilizing the Graph Convolutional Network were considering on one big, huge graph and performing the classification on nodes or graphs after pooling. But our use case was different, we want to perform the node classification considering different circuits with different node sizes. For this we were able to use the Spectral python library which allowed us to define the circuit dataset as a disjoint graph as shown in Figure 3.3 by defining a list of graph data set with x as graph node features, y as target, and a as the adjacency matrix.

This data loader returned a python iterator that would load the data in batches and can be passed to Keras Models. The loader returned, **A** a sparse block diagonal matrix where each block is the adjacency matrix a_i of the i-th graph, **X** is obtained by stacking the node features x_i E is also obtained by stacking the edges e_i, Target **Y** also obtained by stacking the labels y_i which were passed to the models for training. For our use case, Edge was not relevant, so we ignored it. Next, we also added a self-loop to the graph nodes such that during the aggregation, it retains its information as well instead it would have just aggregated the neighbor info and encode it excluding its own info which was not desirable. For each layer, we also normalized the adjacency matrix using the Spektral preprocess utility.

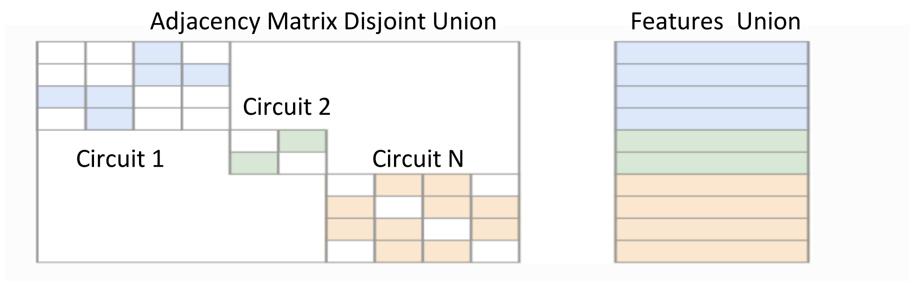


Fig. 3.3: Generate a set of graphs as a single graph using spectral Disjoint Loader, which is their disjoint union where the nodes is the number of all the nodes in the set of graphs.

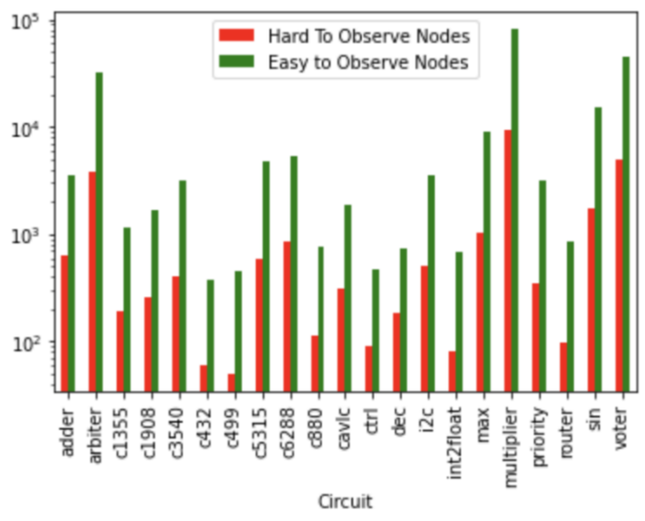


Fig. 3.4: Output Labels, Easy to Observe and Hard to Observe, distribution of the circuits used as the dataset for the training. For each circuit, the number of nodes that are easy to observe are 10 times more than the hard to observe nodes.

However, the unbalanced dataset problem still prevailed with these combined circuits. We can see the unbalanced dataset for all the circuits as given by the distribution of the labels for different circuits is shown in Figure 3.4 where the easy to observe nodes are 10 times more than the hard to observe nodes. In order to solve the problem of unbalanced dataset, we looked into multiple options such as SMOTE, ROSE, up sampling, down sampling techniques but none of these really fit into our use case. For SMOTE and ROSE we needed some embedding such that we could make the approximation. But we could not perform that for the graph as we need the adjacency matrix connections for each node that we newly introduce. So, we looked for other options available. On research, we came across the weighted loss function based on the class prediction which behavior would help to maintain loss proportionality between the two disproportional classes. Thus, we started exploring the different techniques to define the weight for the loss function while training the model which are discussed in the section below.

3.2 DNN models

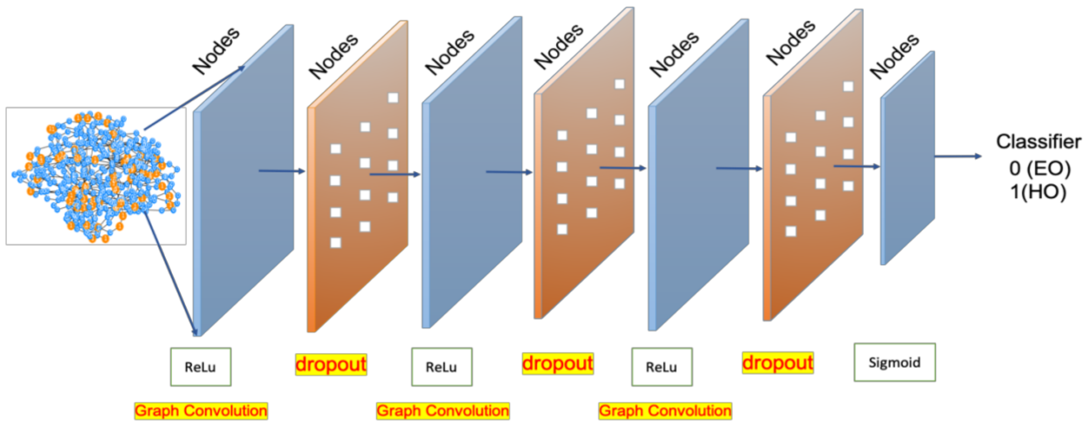


Fig. 3.5 : General Architecture of GCN model for training

We explored different convolutional/message passing layers which propagates the messages of the neighbor nodes and computes embeddings for each node in the graph which can be passed to the final dense layer for the classification of nodes or graph. As here we deal with the nodes classification and not graph, so Pooling was not required. We started with the GCN[3] model which is fast, localized spectral filtering generalized CNN for high dimensional irregular domains. It is a first-order approximation of spectral graph convolution that learns the hidden layer representations that encodes both the local graph structure and features of the nodes. We defined a sequential 3- layer GCN Model as shown in Figure 3.5 with 64 channels, Relu activation followed by the final node classification layer with sigmoid activation. Each GCN layer generates a 64-dimensional feature vector for each node by performing aggregation and encoding the node information which is passed on to the final layer for the classification. The inputs to this model are the node features, the adjacency matrix and the graph index and the output labels. After training and evaluating the Model we were able to get good overall accuracy however it was only contributed from the majority class i.e all the nodes were predicted to be of the majority class (EO), none of the nodes were predicted to be of the minority class (HO) as shown below in the Figure 3.6.

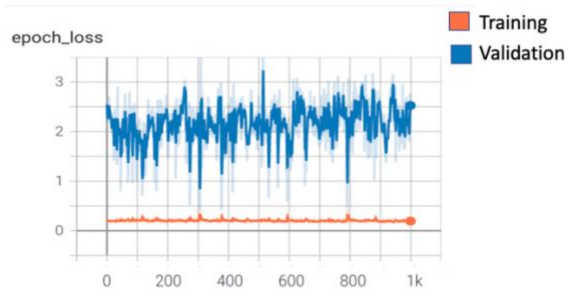


Fig. 3.6 : Model not able to learn as the loss is constant and all the predictions are the same

We worked on the custom train and evaluation function for keras and tried different class weight ratios for negative to positive samples such that the model learns both the classes in proportional manner. With this we saw slight progress and we were able to see predictions for the minority class as well, however, this was not the case for all the test samples. It was able to make predictions for one circuit while for the other it was again all 0. Then, we tried binary focal loss which also didn't help much and then finally used the sklearn compute_class_weight to compute the class weights and pass on the Keras during the training. We computed the class weights based on the training dataset. Initially, we normalized the input features based on global normalization parameters. Later on, we used circuit level normalization to normalize the node features of respective circuits. This with the class weight along with increased GCN layers from 3 to 8(in multiple iterations), increased channels from 64 to 1024(in multiple iterations), using binary cross entropy as a loss function and increased learning rate to 1e-5 we were able to make significant improvement across all the performance stats for both the class (Precision, Recall and F1 score). Then, the model starts to overfit indicating the model has enough capacity to learn as shown in Figure 3.7.

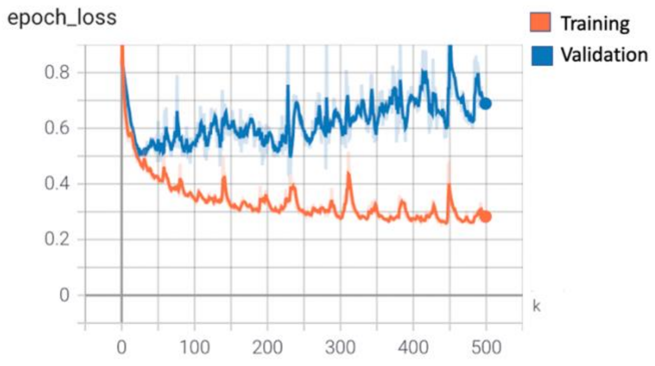


Fig. 3.7: Overfitting Model

To overcome the overfitting, we introduced l2 regularizers on kernel and bias, Dropout Layers after each convolutional layer, decreased the number of convolutional layers and number of channels as well. We also added Early Stopping. This helped in better training as shown in Figure 3.8 and values of kernel converged into normal distribution as shown in Figure 3.9.

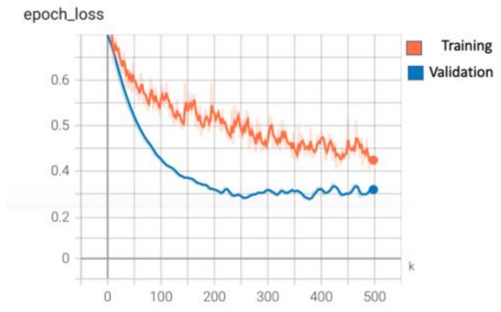


Fig. 3.8: Better trained loss report without overfitting

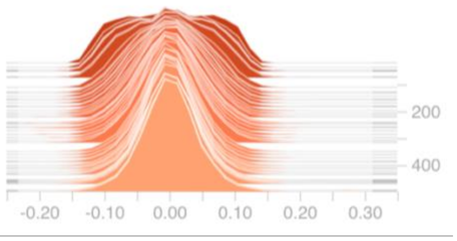


Fig. 3.9: Kernel values converged to normal distribution for well-trained model.

Then we started to experiment with different models as explained below and their hyperparameters such as learning rate, number of channels, regularizers and improve the precision, recall and F1 score of the model as for the unbalanced dataset, the accuracy metrics was not much useful utilizing tensorboard to get the evolution of weights, kernels, bias and layer activations as training progressed through the epochs as we did before for the above model.

We then explored Chebyshev Convolutional layer [12], again a localized spectral generalized CNN with k-order approximation. We performed the experiments with a multilayer convolutional layer with different values of k. We went with the min 25-order approximation as it provides better approximation. We experimented with multiple parameters.

The other layer that we experimented is Topology Aware Graph Convolutional network (TagConv) which tends to overcome the fix-size learnable parameters of k-approximation method. In TagConv, the topology of filters is adaptive to the topology of the graph, and we can propagate up to k-neighbor. This was a fit for our use case as there is propagation of faults from N-neighbor that cause the gates to be stuck at 0 or 1 and resulting in hard to observe nodes. We defined a model with sequential multiple TagConv layers followed by dropout and batch normalization. We experimented with multiple layers 1 to 3(multiple iterations), updating the hyperparameters, learning rates and value of k from 3-10(multiple iterations).

All of these were propagation models, we also explored a sampling module, GraphSage. It leverages node features to generate the node embeddings for previously unseen data. It performed the sampling and aggregation of features from node's local neighborhood. It is said that the sampling module with the propagation module goes well so we defined a sequential different convolutional layer stacked sequential model. We stacked GraphSage followed by the other layers such GCNConv or ChebySev or TCNConv as a Sequential Model. But there was no improvement. So instead, we went ahead and experimented concatenating the node embedding generated by TagConv which was giving better result and GraphSage which was supposed to give a better result on unseen graph as well and then flatten the k-dimensional node features followed by a dense layer for the classification as shown in Figure 3.10 using Keras Functional API. The third input is ignored as it is the index to the graph which is useful when performing graph classification while performing the pooling.

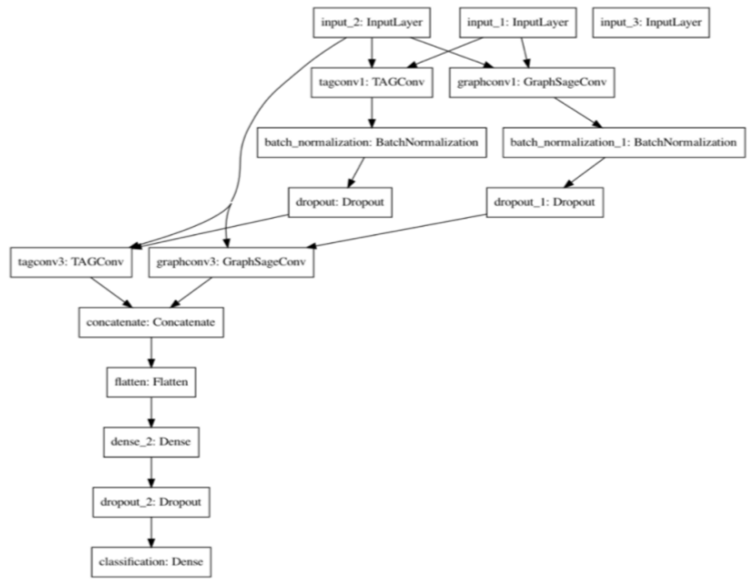


Fig. 3.10: Multi Input single output Model

3.3 Base ML model

For our based model we used ML models like Decision Tree and Random Forest to see how good the prediction is. Decision Tree (DT) is a tree-based structure in which each node represents each input feature, and each edge represents the relation between the features. The leaf nodes are the labels that are used to decide the performance of the model. Random Forest (RF) is a combination of tree predictions. Each tree in the random forest relies on the independent sample values of a random vector.

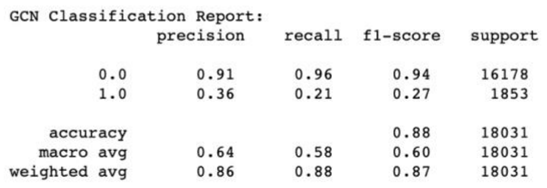
The input features are CC0, CC1, LL and CO and the ground truth labels are HO and EO. We used cross fold validation to split the dataset into train, test and validation set.

4. EVALUATION:

4.1 DNN Model Evaluation

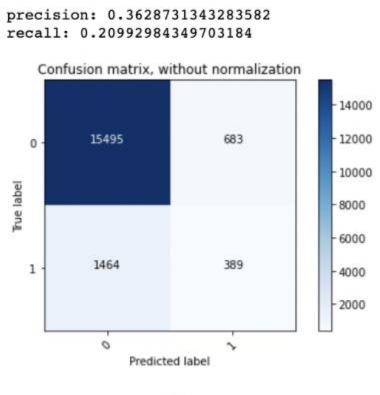
GCNConv Model:

This model has low precision and recall scores with AUC of 87% for the overall test circuits. Looking at evaluation for individual circuits in test set as shown in Figure 4.1, we find that two test circuits have 100% recall while the third one has 100% precision as shown in Figure 4.2.



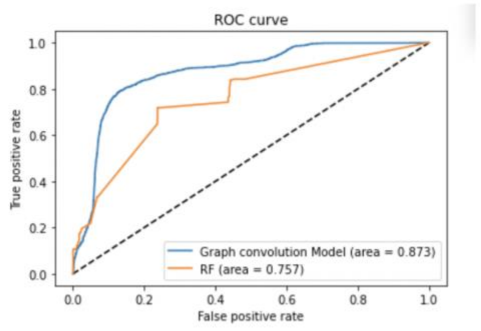
```
GCN Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.96      0.94     16178
         1.0       0.36      0.21      0.27      1853

    accuracy                           0.88     18031
   macro avg       0.64      0.58      0.60     18031
weighted avg       0.86      0.88      0.87     18031
```

(a)



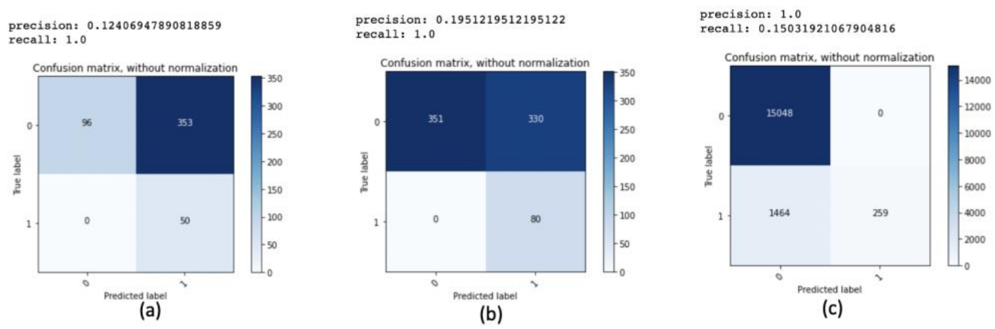
```
precision: 0.3628731343283582
recall: 0.20992984349703184
```

(b)



(c)

Fig. 4.1: (a) Classification report (b) Confusion Matrix (c) ROC Curve with AUC of 87%



```
precision: 0.12406947890818859
recall: 1.0
```

```
precision: 0.1951219512195122
recall: 1.0
```

```
precision: 1.0
recall: 0.15031921067904816
```

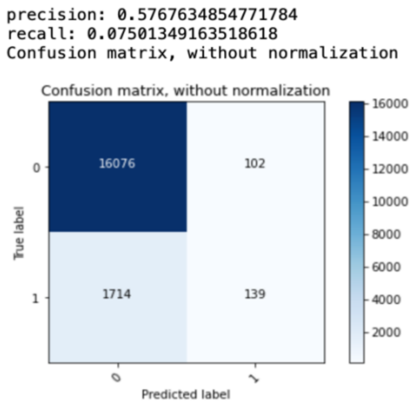
Fig. 4.2: Test set circuits evaluation reports
(a) Int to Float circuit (b) Sin Circuit (c) c499 circuit

TagConv Model

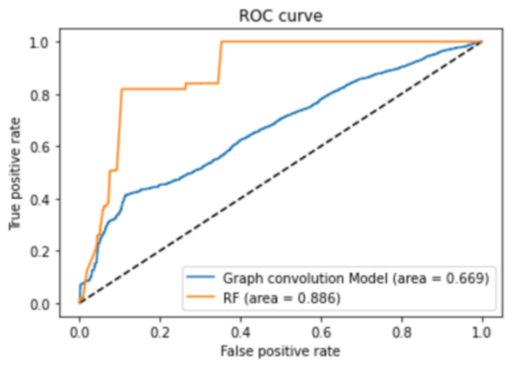
This model has low precision and recall scores with AUC of 67% for the overall test circuits. Looking at evaluation for individual circuits in test set as shown in Figure 4.3, we find that two test circuits have 100% recall while the third one has 100% precision as shown in Figure 4.4.

```
GCN Classification Report:
               precision    recall  f1-score   support

         0.0       0.90      0.99      0.95     16178
         1.0       0.58      0.08      0.13      1853

    accuracy                           0.90     18031
   macro avg       0.74      0.53      0.54     18031
weighted avg       0.87      0.90      0.86     18031
```

(a)

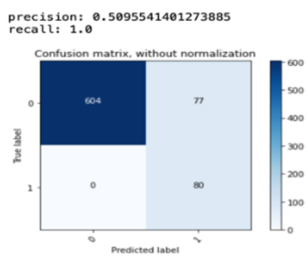


```
precision: 0.5767634854771784
recall: 0.07501349163518618
Confusion matrix, without normalization
```

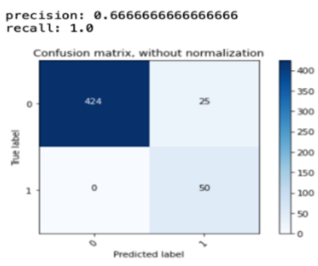


(b) (c)

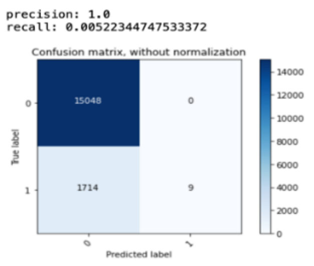
Fig. 4.3: (a) Classification report (b) Confusion Matrix (c) ROC Curve with AUC of 67%



```
precision: 0.5095541401273885
recall: 1.0
```



```
precision: 0.6666666666666666
recall: 1.0
```



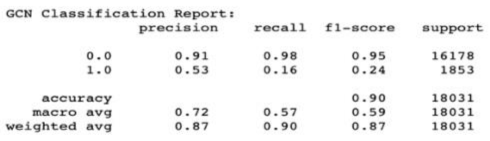
```
precision: 1.0
recall: 0.00522344747533372
```

(a) (b) (c)

Fig. 4.4: Test set circuits evaluation reports
(a) Int to Float circuit (b) Sin Circuit (c) c499 circuit

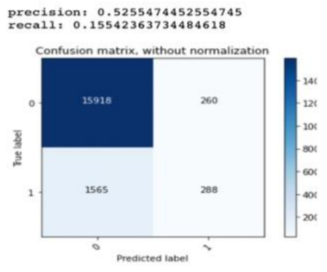
TagConv and Graph Sage Conv

Precision is higher than recall for overall test data as shown in Figure 4.5. However, both are less than 55%. But AUC ROC is at 88% which is considerably higher value. From ROC curve we see that this model performs better than base Random Forest Model. Looking at evaluation for individual circuits in test set as shown in Figure 4.6, we find that two test circuits have 100% recall while the third one has 100% precision. The ROC for all the test circuits are between 92 to 97%.

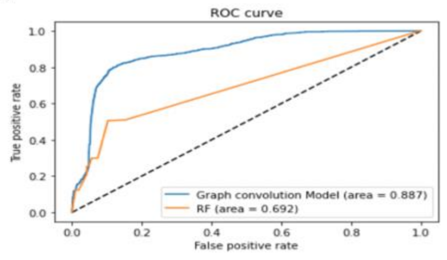


```
GCN Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.98      0.95     16178
         1.0       0.53      0.16      0.24      1853

    accuracy                           0.90     18031
   macro avg       0.72      0.57      0.59     18031
weighted avg       0.87      0.90      0.87     18031
```

(a)

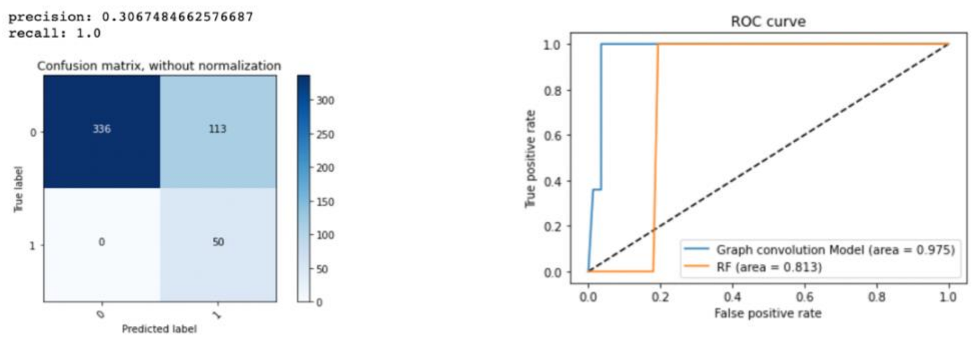


```
precision: 0.5255474452554745
recall: 0.15542363734484618
```



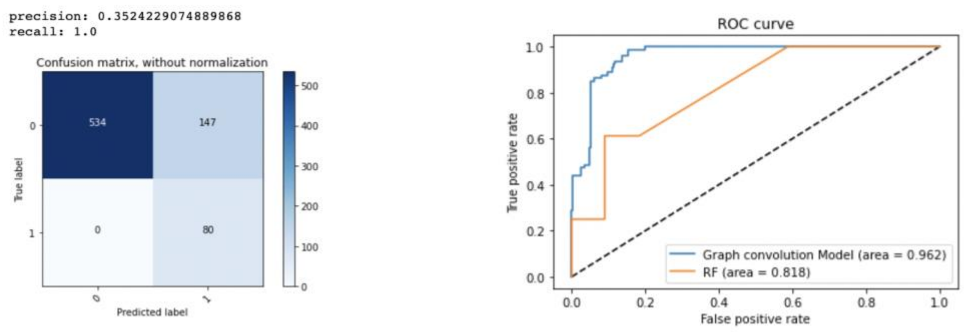
(b) (c)

Fig. 4.5: (a)Classification report (b) Confusion Matrix (c) ROC Curve with higher percentage of area under the curve 88%



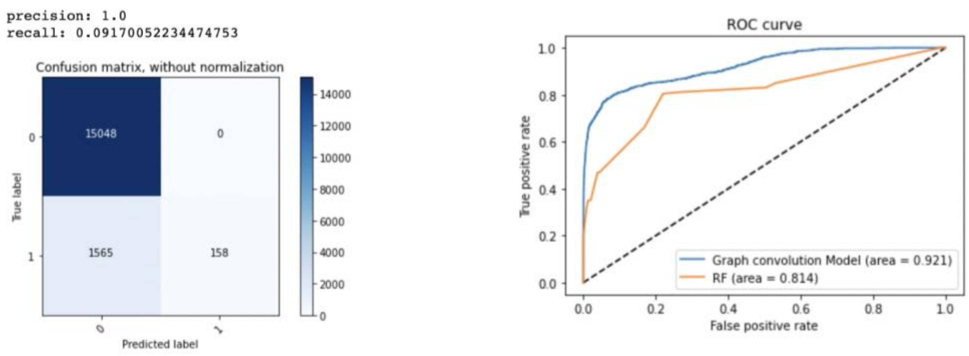
```
precision: 0.3067484662576687
recall: 1.0
```

(a)



```
precision: 0.3524229074889868
recall: 1.0
```

(b)



```
precision: 1.0
recall: 0.09170052234474753
```

(c)

Fig. 4.6: Test set circuits evaluation reports
(a) Int to Float circuit (b) Sin Circuit (c) c499 circuit

4.2 Model Comparison

GCN Conv(4.1.1) vs TagConv(4.1.2) vs TagConv and GraphSage Conv(4.1.3)

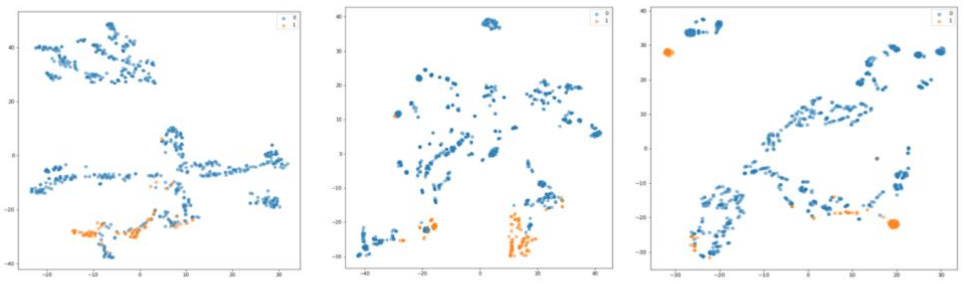


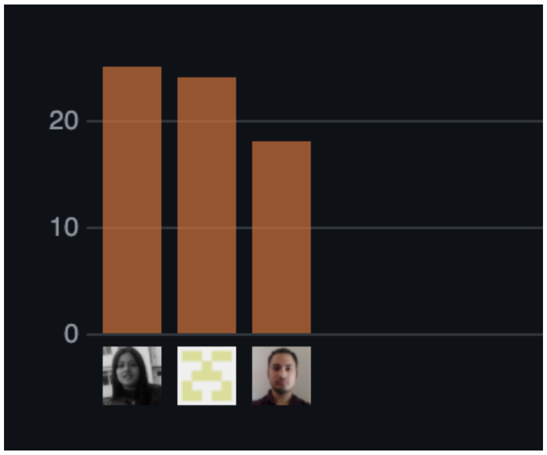
Fig. 4.7: T-SNE embedding of the input to the final classification layer
(a) GCNConv (b) TagConv (c) TagConv and GraphSageConv

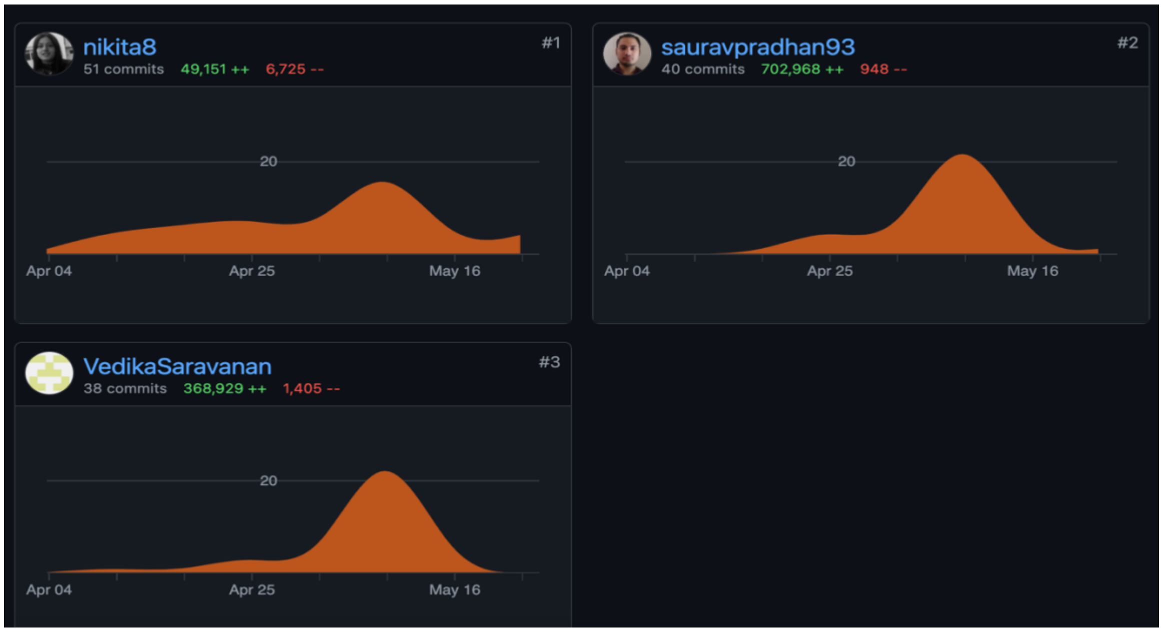
Looking at the T-SNE embedding of the inputs before the classification layer for multiple models as shown in Figure 4.7 above, we see that for GCNConv, there is overlapping of the two classes while for TCNConv, we see most of the hard to observe nodes are concentrated in an area separate from easy to observe nodes, but still the easy to observe nodes are spread over the larger area. While for the TagConv and GraphSage Conv combined model, we see most of the hard to observe nodes are embedded in a compact region and are separate from easy to observe nodes. Those nodes which are not in the concentrated on the two hard to observe clusters are mixed with easy to observe nodes, but the numbers are comparatively less than the other two models. Thus, we think having a separate concentrated area for hard to observe nodes results in comparatively better performance than the other two models.

5. CONCLUSION:

On solving the problem for finding the nodes that are hard to observe for testability of VLSI benchmark circuits we were able to leverage deep convolutional deep neural network to classify the nodes. Even though traditional classification model can somehow make predictions, DNN had better performance. Of all the different model architectures that we tried, TagConv and GraphSageConv performed better. But still, for each circuit it would either have 100% recall or 100% recall but not a balanced combination of both. We saw this model was able to cluster the hard to observe nodes better than the other models which could be argued as the reason for its better performance.

6. ATTRIBUTION:





We created task in the project and every week we used to meet to discuss the work progress and create a story in the project for the task to be done for the next week. We have all the notebooks that we worked on inside the Notebook folder in GitHub. We spent 5 hours a day working on this project.

Vedika Saravanan

Collected benchmark circuits are extracted the features for the model and tried training different models.

Nikita Acharya

Generate the pickle files, input pre-processing and tried working on different models

Saurav Pradhan

Worked on base model and tried different DNN models

BIBLIOGRAPHY:

[1] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program," *17th Design Automation Conference*, 1980, pp. 190-196, doi: 10.1109/DAC.1980.1585245.

[2] A. B. Chowdhury, B. Tan, S. Garg and R. Karri, "Robust Deep Learning for IC Test Problems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, doi: 10.1109/TCAD.2021.3054808.

[3] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu and E. F. Y. Young, "Layout Hotspot Detection With Feature Tensor Generation and Deep Biased Learning," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1175-1187, June 2019, doi: 10.1109/TCAD.2018.2837078.

[4] A. F. Tabrizi, N. K. Darav, L. Rakai, I. Bustany, A. Kennings and L. Behjat, "Eh?Predictor: A Deep Learning Framework to Identify Detailed Routing Short Violations From a Placed Netlist," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1177-1190, June 2020, doi: 10.1109/TCAD.2019.2917130.

[5] Y. Ma *et al*., "High Performance Graph ConvolutionaI Networks with Applications in Testability Analysis," *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1-6.

[6] Y. Sun and S. Millican, "Test Point Insertion Using Artificial Neural Networks," *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 253-258, doi: 10.1109/ISVLSI.2019.00054.

[7] Kipf, Thomas N., and Max Welling, "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

[8] W. L. Hamilton, R. Ying, J. Leskovec, "Inductive Representation Learning on Large Graphs." ." *arXiv preprint arXiv:* 1706.02216 (2017).

[9] Du, Jian, et al. "Topology adaptive graph convolutional networks." *arXiv preprint arXiv:1710.10370* (2017).

[10] http://www.pld.ttu.ee/~maksim/benchmarks/iscas85/verilog/

[11] https://www.epfl.ch/labs/lsi/page-102566-en-html/benchmarks/

[12] M. Defferrard ,X. Bresson and P. Vandergheynst, " Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering", *arXiv preprint arXiv:* 1606.09375 (2017)