technische universität
dortmund

AI-Based SI/PI/EMC-Compliant PCB Design

Implementation of ML/AI Modules to Support Interference-Resistant Design of Microelectronic Systems

Signal Integrity (SI) Design Support on PCB Net Structures Based on Machine Learning (ML) for Application to Differential Signaling Deployed with a Streamlit Dashboard Application

Project Group WISE 2023/24

submitted to

Faculty of Electrical Engineering and Information Technology

AG Datentechnik / Information Processing Lab

Technische Universität Dortmund

by
Prasanna Agarwal
Yahia Mesharaf
Madhurya Lakshminarasimha
Sushmitha Govindaraj
Thiyanayugi Mariraj

Academic Supervisors:
Dr. Ing. Werner John
Prof. Dr. Ing. Jürgen Götze
M.Sc. Emre Ecik
M.Sc. Nima Ghafarian Shoaee
M.Sc. Julian Withöft

DT/IPL Arbeitsgebiet Datentechnik
Information Processing Lab

Abstract

The objective of this project is to develop and implement advanced regression and classification models to analyze and predict the performance of high-speed digital circuits, with a focus on transient responses and eye diagrams of differential and single-ended pairs. The project encompasses several key areas, including literature research, tool installation and setup, data generation, and the application of machine learning models.

Initially, a comprehensive literature review was conducted to understand the state-of-the-art techniques in regression analysis and artificial neural networks (ANNs) for signal integrity analysis. This review provided a foundational understanding of the current methodologies and identified gaps that this project aims to address.

To facilitate the implementation of these models, various software tools and simulation environments were installed and configured. These tools were essential for generating accurate and reliable data for both transient responses and eye diagrams. Data generation involved simulating the transient responses of single-ended and differential pairs under various conditions to create a robust dataset. Similarly, eye diagrams were generated to capture the signal integrity of the circuits, providing critical data for model training and evaluation.

For the transient response analysis, a regression model was developed to predict the behavior of single-ended signals. This model was trained using the generated dataset and rigorously validated to ensure its accuracy and reliability. The regression model demonstrated a high degree of precision in predicting transient responses, making it a valuable tool for circuit designers.

In parallel, a convolutional neural network (CNN) model was designed and trained to classify eye diagrams. The CNN model aimed to predict whether the eye diagram meets predefined performance thresholds, thus offering a rapid and automated assessment tool for signal integrity. The CNN was trained on a diverse dataset of eye diagrams, enabling it to generalize well across different signal conditions.

The results of this project highlight the effectiveness of using regression and CNN models in the context of high-speed digital circuits. The regression model provides precise predictions of transient responses, facilitating the design and optimization of circuits. Meanwhile, the CNN model offers reliable classification of eye diagrams, enabling quick and accurate assessments of signal integrity. This dual approach enhances the efficiency and accuracy of circuit design and verification processes.

Overall, this project contributes significantly to the advancement of automated signal integrity analysis. By leveraging machine learning techniques, it provides tools that can greatly enhance the efficiency and accuracy of high-speed digital circuit design and validation. The integration of these models into the design workflow has the potential to streamline the development process, reduce time-to-market, and improve the performance and reliability of digital systems.

Contents

1 Objective and Introduction

Due to several factors the interference-resistant design of today's microelectronic systems is becoming more and more challenging in all fields of industrial applications (automotive/robotics/telecommunication/...).

At first there is the ever-increasing complexity of the systems being designed as the number of PCB layers and its transistor density continue to grow. This development is accompanied by increasing electrical requirements regarding IC current needs, clock rates, rise- and fall times and voltage levels, which are not least driven by technological progress.

However, while these high-speed advancements may enable improved integrability and enhanced computational capability they also entail numerous interacting electromagnetic effects, which may cause severe problems, if not resolved.

Finally, complicating matters is the fact that design cycles during the PCB development process are getting shorter and shorter and are being performed by smaller and smaller teams. This is exactly the connecting factor for ML/AI methods, which have already found their way into various engineering domains and are therefore promising to effectively support the IC/PCB design.

1.1 Objective

The focus lies on the application of existing already established AI/ML methods and toolkit on the domain of microelectronic system design to ensure an interference-free PCB design.

Therefore, the interdisciplinary application of knowledge from two different fields is encouraged. Furthermore, to reach the desired objectives, practical and theoretical work is equally important.

On the one hand, the participants of the project group are to be taught the currently common practice and industry relevant working methods in electronics design from the knowledge domain of interference-free PCB design.

On the other hand, the participants are empowered to expand and sharpen their scientific working methods.

1.2 Introduction

Assuring signal integrity (SI) on a PCB network is crucial to comply with electromagnetic compatibility (EMC) constraints.
In today's microelectronic component design, it is becoming increasingly challenging to comply with these constraints requiring multiple design cycles for the PCB developer, which results in a very nerve-wracking, time-consuming and costly process.
Therefore, AI or in particular methods from the machine learning (ML) subdomain have received growing attention in the field of electronic design automation (EDA) and PCB design under SI constraints.
Based on these considerations an AI framework to support SI design was established at our institute. Moreover, a graphical user interface in form of a Streamlit application (https://streamlit.io/) was developed.

2 Task Definition and Literature research

2.1 Task Definition

In a modern-day world, we aspire to use and design systems that do not take up a lot of time to complete their job. This is mainly due to the advancements in processor design technology, high number of incoming tasks for the processor and an aim to conserve electrical power and increase efficiency. Hence, we usually resort to digital components which have very fast edge rates and therefore, take up less computation time.

However, these 'high-speed' components use standard PCB materials, and they end up causing signal degradation with time. Some signal integrity problems are:

- Excessive insertion loss in long channels.
- Dominant reflection loss in short channels.
- Dispersion, leading to spreading of digital signals and phase distortion.
- Strong crosstalk, or ground bounce that mimics crosstalk.
- Skew and jitter can only be identified with power integrity analysis.
- Reflection at a receiver due to impedance mismatch.

Hence, signal integrity is crucial for ensuring that data can be passed conveniently between the components on a PCB. It is also necessary to comply with electromagnetic compatibility constraints.

Simulation is extremely important to study the effects of varying transmission line lengths, impedance and noise on the signals. In today's microelectronic component design, it is becoming increasingly challenging to comply with these constraints requiring multiple design cycles for the PCB developer, which results in an extremely nerve-wracking, time-consuming and costly process. Therefore, AI or in particular methods from the Machine Learning subdomain have received growing attention in the field of electronic design automation (EDA) and PCB design under SI constraints.

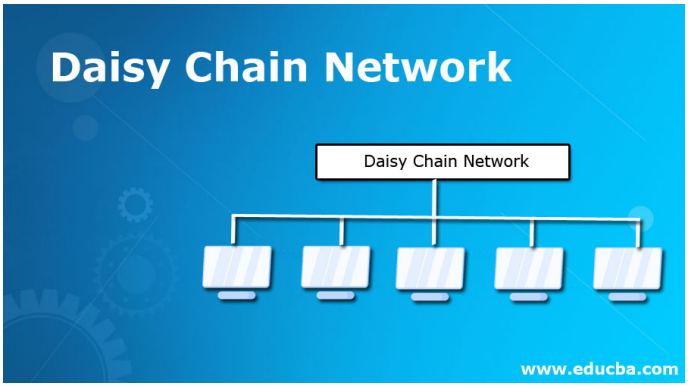


Figure 2.1: An Example of Daisy Chain

Machine Learning reduces the time and complexity associated with designing all kinds of circuits. The main aim of applying Machine Learning in circuit design is to automate the design process, which reduces the design cycle time. Machine learning-based EDA accelerates the physical circuit design process without requiring a human. Machine learning algorithms shorten the design time in EDA, thus minimizing the design to market time.

We consider a daisy chain for our implementation. A daisy chain refers to an electrical circuit wiring scheme where multiple components/devices are wired together in a ring fashion. It resembles a garland of daisy flowers, hence the name.

We also resort to two types of signalling applications: single-ended and differential signalling. We implement the following steps:

• Get used to the data generation process with the Zuken eCADSTAR simulation software, the feature extraction tool chain (python-libraries: pandas, numpy, scipy, etc.) and the AI models (python-libraries: tensorflow, keras, scikit-learn, pymoo, stable-baselines3) using the example of a basic single-ended PCB topology. Perform some initial analyses/-comparisons regarding different feature constellations and ML algorithms.

• Specification/development of a suitable differential pair topology and parameter space as well as necessary adjustments with respect to the feature space and finally the selection of suitable ML algorithms.

• Implementation and subsequent evaluation of the specified/modified process for the application to differential signals.

• Adjustment and improvement of the Python-based Streamlit dashboard application.

• Summarization and presentation of the results and insights.

2.2 Literature Research

The concept of Artificial Neural Networks (ANNs) and Classification in Machine Learning is fairly old, but only finds consistent use in research and development in the modern years. However, maintaining Signal Integrity in an electrical system, whether simple or complex is quite an ancient problem.

According to [2], ML methods such as DNNs, Linear Regression and SVMs are used to predict the eye-diagram metrics, using massive amounts of data gathered from prior simulations. The regression-based learning process has three components: a generator, a supervisor and a regressor. The generator produces one set of predictors as a vector X. The supervisor, returns the eye height or width 'y' corresponding to X. The learning process is essentially the selection of the right regression function f(X, THETA), where THETA contains the parameters to be learned. The three methods of regression used are also compared with respect to handling non-linearities, accuracy and performance. Finally, it is proven that DNN regression methods have good accuracy, good capability to handle non-linearities and good overall performance. Hence, we use it to predict Eye Diagram and Transient measurements based on data simulated in eCADSTAR.

In [3], an inverse design method has been proposed to reduce the time and effort spent on unnecessary amounts of data simulations. A desired Eye Diagram model with predefined characteristics is chosen and a novel ML architecture called Lifelong Learning is proposed. The desired characteristics are fed as input and the resulting output is the ideal/desired transmission line and impedance parameters for circuit design. The resulting system has been shown to demonstrate approximately the same results as going in the Layman's way. This is a breakthrough in the field of ML, as it reduces the usual amount of time and efforts by a significant margin, whilst still ensuring desired results. This method could be used for training a system that relies on higher amounts of data for learning.

According to [4], another algorithm named Deep Genetic Algorithm is used to optimize the high-speed channel for signal integrity. To reduce the time consumed for the whole process, a DNN algorithm was further embedded into a GA algorithm to directly predict eye diagram information from design parameters of the high-speed channel. The GA algorithm consists of the following stages:

• Initialize the Chromosomes - Suppose that the design parameters are limited in a given search space. A set of design parameters are chosen randomly in the search space. Then they are coded to form a set of chromosomes.

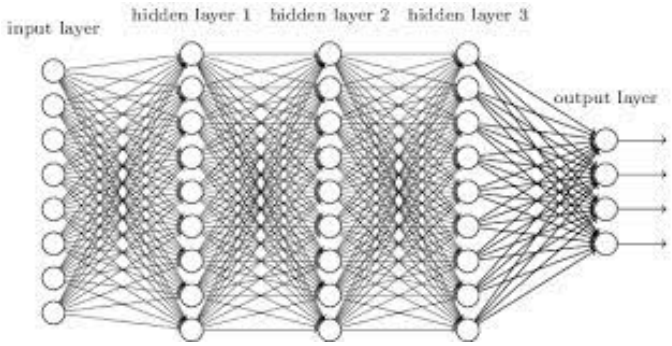


Figure 2.2: An Illustration of Deep Neural Network Implemented in SI Analysis

• Evaluation - A fitness score is given to each chromosome according to the eye height and eye width of the eye diagram. A deep feed-forward neural network is constructed to characterize the fitness function.

• Selection - Chromosomes are selected based on the scores by using the roulette wheel method. The chromosomes with good fitness scores have a high probability to be chosen.

• Crossover - After two chromosomes are selected, crossover sites are chosen randomly. The genes of the two chromosomes at the crossover sites are exchanged to create two new chromosomes.

• Mutation - The genes of the chromosomes have a small probability to change to other genes.

The DNN is utilized to characterize the fitness function, which can avoid full wave simulation and significantly reduce the computational cost. The proposed method provides a universal and powerful tool for the optimization of high-speed channels for signal integrity.

Finally, in [1], we can analyse the procedures of creating ML modules for SI analysis in PCB design. ML modules such as kNN regression, Support Vector Regression and Neural Network(keras) are analysed. The data was obtained through simulation of various values of resistances and transmission line lengths in a PCB net 1.7 circuit. The simplest algorithm to be implemented was the k Nearest Neighbour(kNN) as a method of unsupervised learning. The generalization of NN model was well-adjusted based on the plots of training and validation loss. For the SVR implementation, two scenarios were analysed for the star point net. For the worst-case scenario, inadequate slew rate was achieved and for the best-case scenario, a high slew rate with negligible overshoot was obtained. The multi-output SVR provided reasonable predictions. The above methods were analysed for other networks as well and were analysed for the worst and best-case scenarios.

3 Tool Setup and first examples

In this chapter, we will discuss in detail about the software used in the project, providing the outline of the software, followed by detailed explanation of the installation process. We briefly discuss the initial examples used to familiarize with the software's interface and get better understanding. These steps helps to proceed forward with the project in a hassle free manner.

3.1 eCadstar – Overview of the Electrical Design Software

eCADSTAR created by Zuken is an tool used to design and create schematic diagrams and printed circuit boards. A number of features help to facilitate the design and evaluation process with this tool. The software assists in Signal integrity (SI) analysis which helps in examining the functionalities of electrical signals in the circuit, enabling the stability and reliability.This helps to find and eliminate the potential issues such as signal interference of distortion. The software provides a user friendly interface that increases the efficiency in the design process. The software makes it is easy to draw schematic, place components and alter the values of each component with minimal effort.

Figure 3.1: Logo of Zuken eCadstar

After successful completion of literature review and task definition, eCADSTAR was chosen as the software to work for the project. For SI performance, monitoring the eye diagrams and transient responses plays a major role which is met by using the above tool.

3.2 Installation Procedure for eCADSTAR

The steps involved in installing the software is explained below. The installation procedure started with downloading the necessary files, gaining access to the VPN, network license configuration and installation of the software.

3.2.1 VPN Setup:

To work on the software, VPN (Virtual Private Network) was needed. The VPN was provided and installed to get a secure connection to the network.

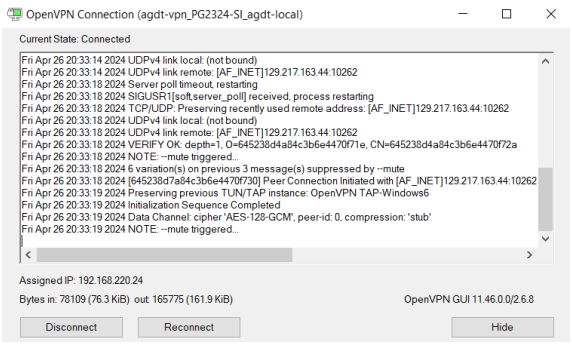


Figure 3.2: OpenVPN window

3.2.2 Access to Network and License Installation

After proper VPN connection, the network access for hosting the software was obtained. A network licensed installation was chosen to provide access for multiple users at the same time. The IP address for the network license server provided was 129.217.163.36.

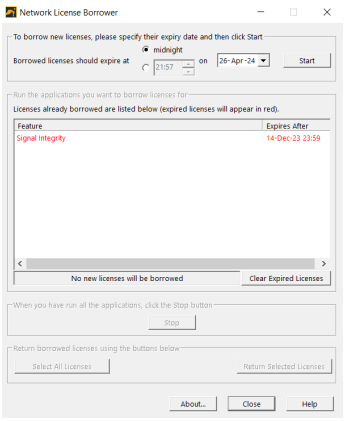


Figure 3.3: License Borrower

3.2.3 Software Download and Installation:

Upon successful access to the network and license, the .exe file for eCADSTAR was downloaded. Followed by, the installation process began by running the downloaded file and following the instructions to install the file.

The initial instructuions involved were the agreement of the license and providing the license information.

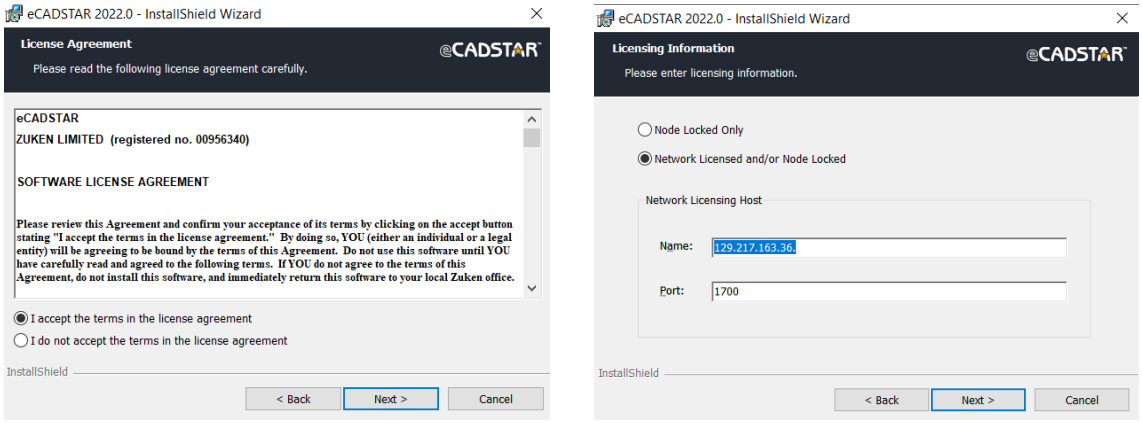


Figure 3.4: Installation steps

Followed by, the destination folder for the installation was chosen and the setup type was selected.Finally the overview of the installation settings were shown and proceeded with installing the software.

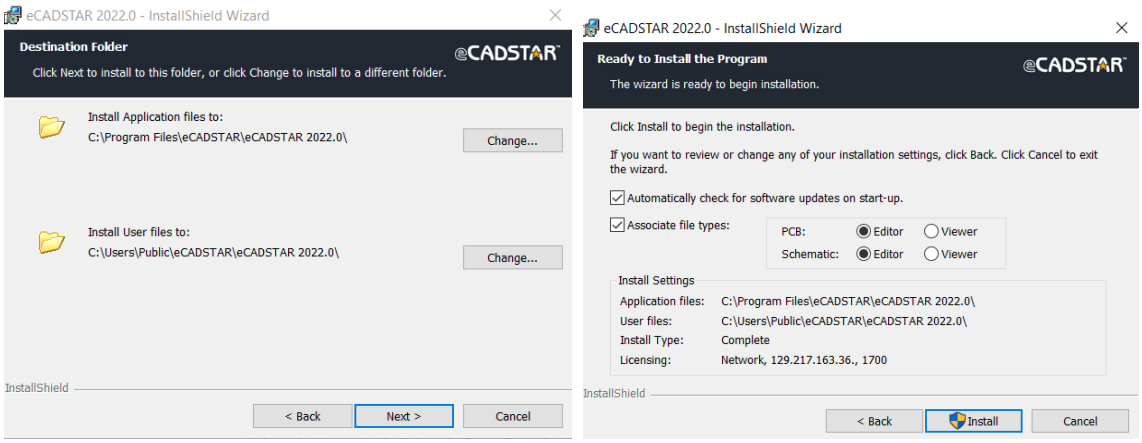


Figure 3.5: Installation steps

Hence,the procedure for installation was briefly explained.After installation , the electrical editor was launched. The basic functions and tools were checked to ensure proper installation. eCADSTAR can now be used for circuit design and simulations.

3.3 Initial Testing and Tool Overview of eCADSTAR

After successful installation of eCADSTAR, tried working on the software to familiarize with the tool's functionalities and workflow. On launching eCADSTAR, a user-friendly interface provided proper navigation and easy access to main features.

The schematic editor had tools for designing and editing schematic diagrams. The components can be easily added and wire connections can also be given quickly assisting a complete circuit diagram.The schematic can be simulated and analysed using the simulation tools which helps in signal integrity evaluation. The initial testing helped in better understanding of the interface, the features and its uses.

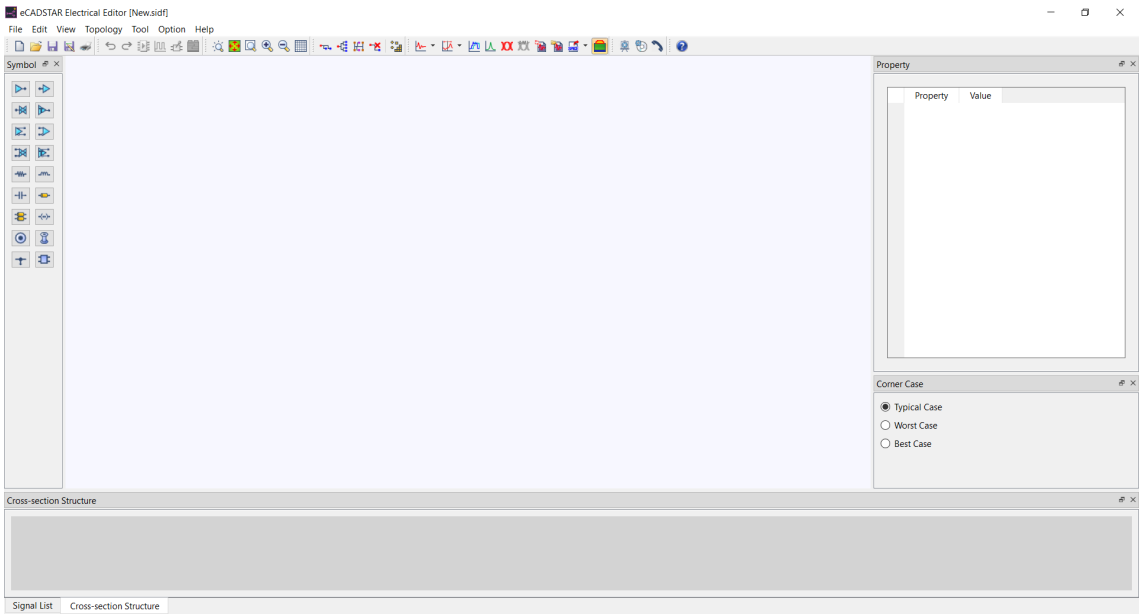


Figure 3.6: Tool interface

3.4 Hands-On Experience with Circuit Examples in eCADSTAR

For better understanding of the software, few examples were provided. The hands on experience of the examples , the learning outcomes and the insights understood through this process is given below.

First,with the help of the designed fundamental circuits (single-ended and differential pair topologies) ,we attempted possible simulations.We used eCADSTAR's schematic editor to view the circuits, followed by signal integrity analysis to evaluate how they responded over time. We employed simulation techniques to evaluate signal integrity and improve circuit performance.

During our practical instruction, we used examples to change the input values on circuit diagrams. This interactive technique allows us to explore the circuits' activity under various conditions and evaluate their responses to parameter changes. We investigated how input values such as impedance, transmission line length, and resistance affect circuit performance. This practical experimentation deepened our understanding of electronic circuits and also increased the ability to troubleshoot and optimize designs.

3.4.1 Single ended Topology

The first example involved was about single ended topology. Single ended signal transmission is the standard way of transmitting data on a PCB. One transmission line is referenced to the ground. The single ended ML applications have been considered for different topology.Using this circuit, we changed the input parameters and learned how eye diagram performs based on the values changed.

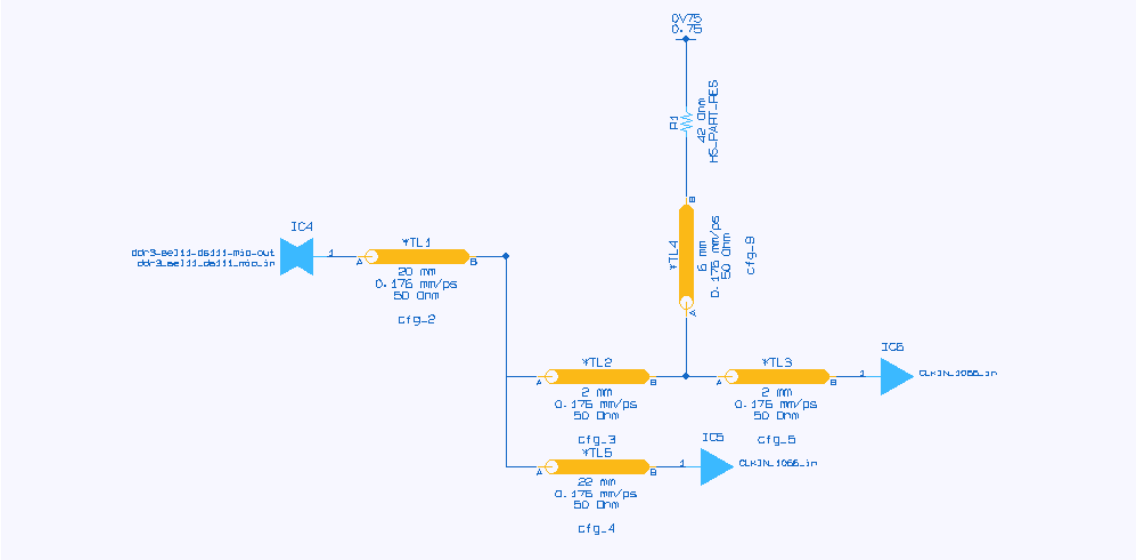


Figure 3.7: Single ended topology Example

3.4.2 Differential Pair Topology

The second example involved was about differential pair topology.In contrast to single-ended signaling, differential pair is not referenced to ground. Instead a second transmission line is used and the differential-mode voltage is the difference between the two signals, which ideally have opposite polarity.With this circuit, we again got to know how differential impacts the eye diagram respectively.

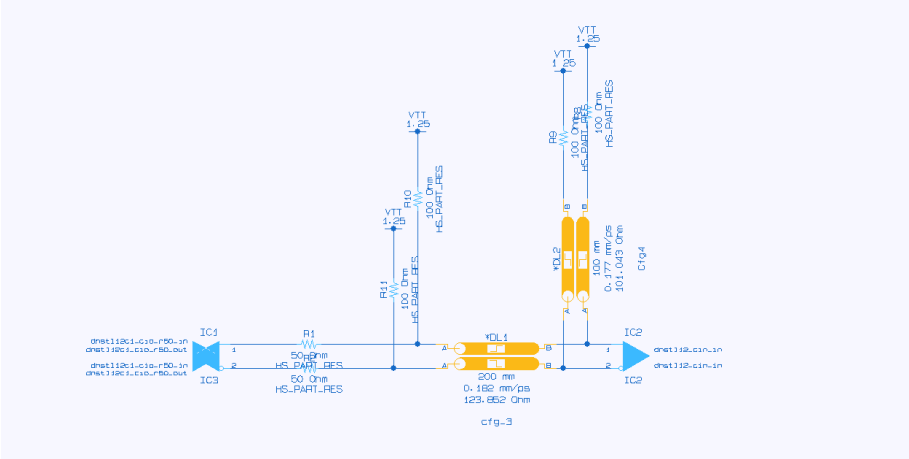


Figure 3.8: Differential Pair topology Example

Overall, these examples were really useful in strengthening theoretical knowledge and improving our ability in using eCADSTAR for electronic design jobs. We learned about eCADSTAR's schematic editor, simulation, and analysis tools through practical experience with circuit examples. We learned how to access the software interface and use its functions properly.

3.5 Lessons Learned and Upcoming Objectives

Following successful installation, the hands-on experience with circuit examples in eCADSTAR proved to be an excellent learning opportunity. Simulating and analyzing circuits helped us get real knowledge with electronic design principles, eCADSTAR capabilities, and simulation techniques.

In the subsequent phase of our project, we will use eCADSTAR to construct single-ended and differential pair architectures for our circuit designs. We will use eCADSTAR's simulation capabilities to investigate the eye diagrams and transient responses of different topologies in order to assess signal integrity and performance. The following chapter will cover the design processes, simulation outcomes, and insights acquired from analyzing single-ended and differential pair circuits with eCADSTAR.

4 Data Generation

4.1 Data Generation for Eye Diagram

4.1.1 Circuit Configuration

In order to better understand signal integrity in PCB design, our study explores the behaviour of single-ended and differential pair circuits, highlighting each circuit's unique properties and performance under a range of conditions.

Single Ended Circuit

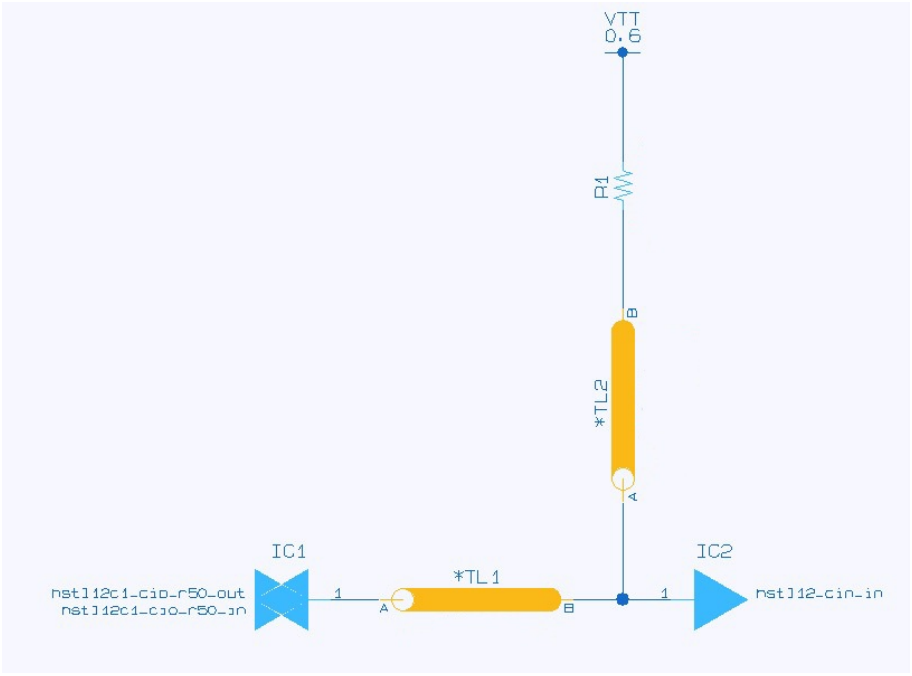


Figure 4.1: Schematic of Single Ended Circuit

The single-ended circuit is a fundamental component in a PCB design. Its common arrangement involves signal reference to a single ground point. By methodically varying parameters like Transmission line Length and Impedance, our study seeks to analyze

the complexities of signal flow in this circuit. We gain knowledge about possible signal deterioration or distortions by examining metrics such as amplitude, timing, and noise characteristics that come from simulations. This information helps us enhance design parameters for the desired performance of the circuit.

Differential Pair Circuit

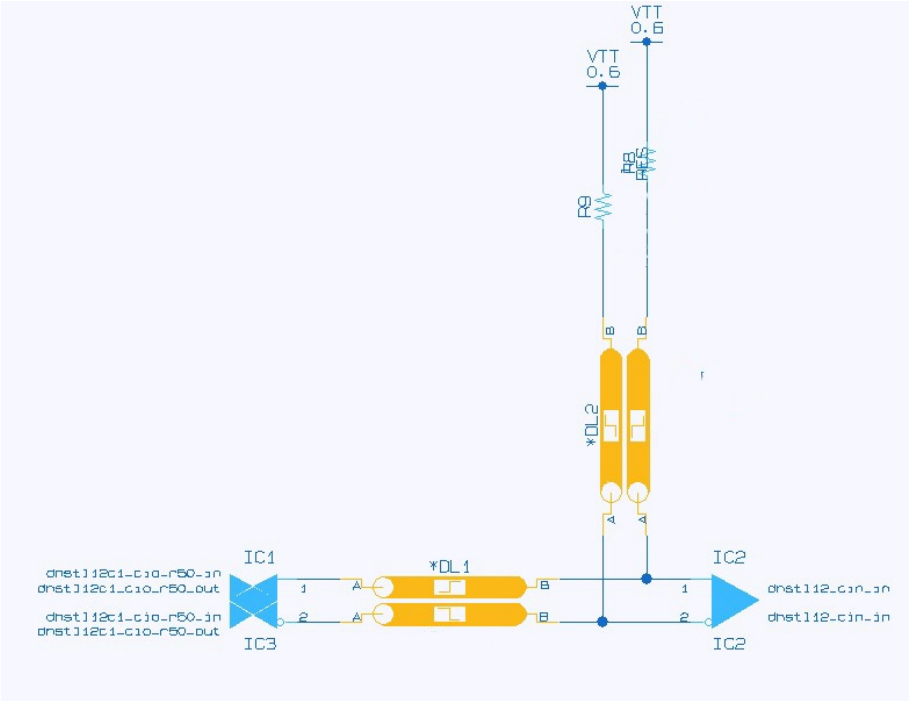


Figure 4.2: Schematic of Differential Pair Circuit

An essential step in PCB design signal integrity is the differential pair circuit. Because it consists of two conductors with equal and opposing signals, this design offers unmatched noise suppression, especially in high-speed data transmission scenarios. To optimise differential signalling for optimal performance, we analyse the intricate relationships between variables such as transmission line length, impedance, and resistances (R1 and R2).We seek to assess the circuit's ability to withstand external disturbances and common-mode noise in order to achieve optimal signal integrity and dependability in crucial electronic systems by carefully examining eye diagrams and probe sets generated by simulations.

4.1.2 Parameter Alterations

A circuit's fundamental characteristics determine how well it can function. The behaviour of the circuit can be affected by modifications to any of the parameters, which can affect

anything from geometric dimensions to electrical properties. This topic explores the significance of parameter changes in PCB design by analysing the effects of transmission length, impedance, and resistance variations on overall circuit dependability, power consumption, and signal integrity. Engineers can tailor PCB layouts for particular applications and performance needs by having a thorough understanding of the impacts of changing parameters and making well-informed design decisions.

Transmission Line Length (TL)

Transmission line length (TL) is an essential element that controls the propagation delays of signals inside the circuit as well as time. Timing issues and signal distortion can result from variations in TL, which can also add resistance and capacitance. By carefully controlling TL variations and ensuring that trace lengths are uniform throughout the PCB layout, engineers may lower signal distortion and enhance circuit performance. To prevent signal deterioration and preserve signal integrity, consistent trace lengths are essential.

Impedance

Impedance is an important factor in PCB design that impacts signal integrity. Signals may decline in quality and reliability due to impedance mismatches, leading to reflections and attenuation. The impedance of the PCB layout should be consistent to keep reflections minimal, and this is critical for rapid data transfer. Impedance variations should be carefully regulated to decrease signal deterioration and enhance circuit performance, ensuring that the impedance of the components and transmission line matches.

Resistance

Resistance characteristics are important in PCB design for power dissipation, energy conservation, and circuit dependability. Voltage drop, power losses, and circuit dependability are all affected by variations in the level of resistance. Power losses and energy consumption are reduced by resistance tuning. This is especially true in the case of battery-powered devices. If circuit dependability is a requirement, then engineers must pay special attention to variations and resistor selection. By proper design and variability reduction, engineers may minimize the effects of resistance alterations while optimizing for their application-specific requirements.

4.1.3 Simulation

Loading the necessary circuit into ECADSTAR Electrical Editor is the first step in the signal integrity analysis process. Transmission line length (TL), impedance, and resistance are varied using the given circuit configuration to see how they affect signal behavior. Following that, corresponding icon in the software interface must be clicked to launch the eye diagram simulation.

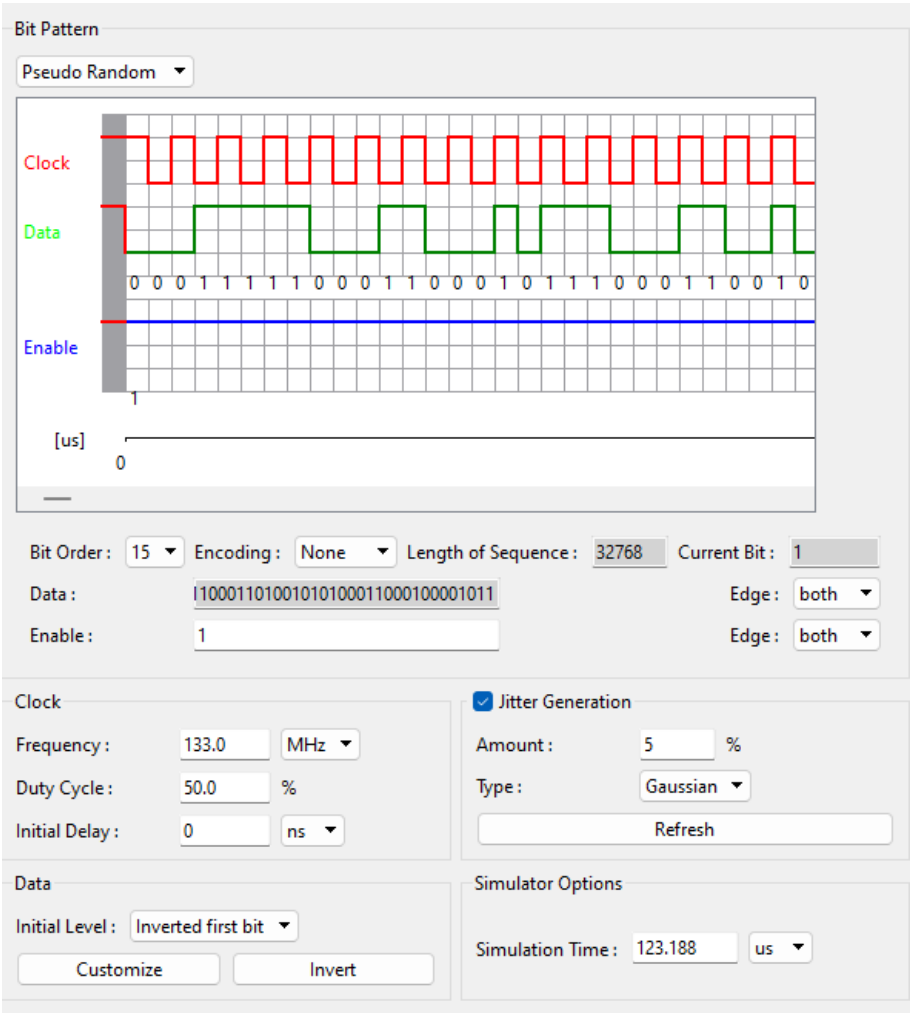


Figure 4.3: Stimulus setting

According to the instructions in figure 4.3, the stimulus is configured to run at a frequency of 133MHz in order to produce proper simulation results. This methodical approach guarantees accurate simulation results, which makes it easier to create comprehensive eye diagrams. This approach helps to optimize circuit performance and reliability by providing important insights into signal integrity.

Output overview

The Analysis Result Viewer Tab displays the analysis results in ECADSTAR Electrical Editor after the simulation procedure. Here, eye diagrams for each of the ICs provide an extensive visual depiction of the circuit's behavior to the users. These eye diagrams, which show variables including the amplitude, timing, and noise characteristics, provide insights into signal quality. Additionally, users have access to the probe set, which includes critical metrics for both ICs, such as Zero level (V), One level (V), Mean level (V), Amplitude (V), Height (V), Width (ns), Opening Factor, Signal to Noise, Crossing Percentage, and Rise time (ps/ns) and Fall time (ps/ns). Users are able to assess circuit performance and pinpoint areas that require optimization due to this thorough study.

4.1.4 Output Analysis

Probeset

By systematically varying inputs over 400 iterations, a comprehensive analysis is conducted to provide a detailed understanding of signal integrity in PCB design. This comprehensive approach yields 400 datasets with two different sets of output values for each IC that are analysed. The datasets offer a wide range of crucial metrics that are essential for evaluating circuit performance, including timing, amplitude, noise characteristics, and other pertinent elements. In order to ensure comprehensive documentation and facilitate additional analysis, every generated data is meticulously categorised and stored in a designated Excel sheet. This enormous data storage is a helpful tool for identifying patterns, spotting trends, and improving design strategies to increase the dependability and performance of circuits.

Image

In addition to the data stored in Excel sheets, an image is produced for each of the 400 models, for a total of 800 photos. An example of an Eye diagram is displayed in picture 4.4. Every photograph is labelled with the relevant probe set number from the Excel sheet, and it is meticulously organised and stored in a dedicated folder. This structure ensures quick and easy access to the circuit performance visual representations for examination and future reference. By means of the image-to-dataset correlation process, users can gain a deeper comprehension of fluctuations in signal integrity and make informed decisions targeted at optimising both circuit performance and reliability.

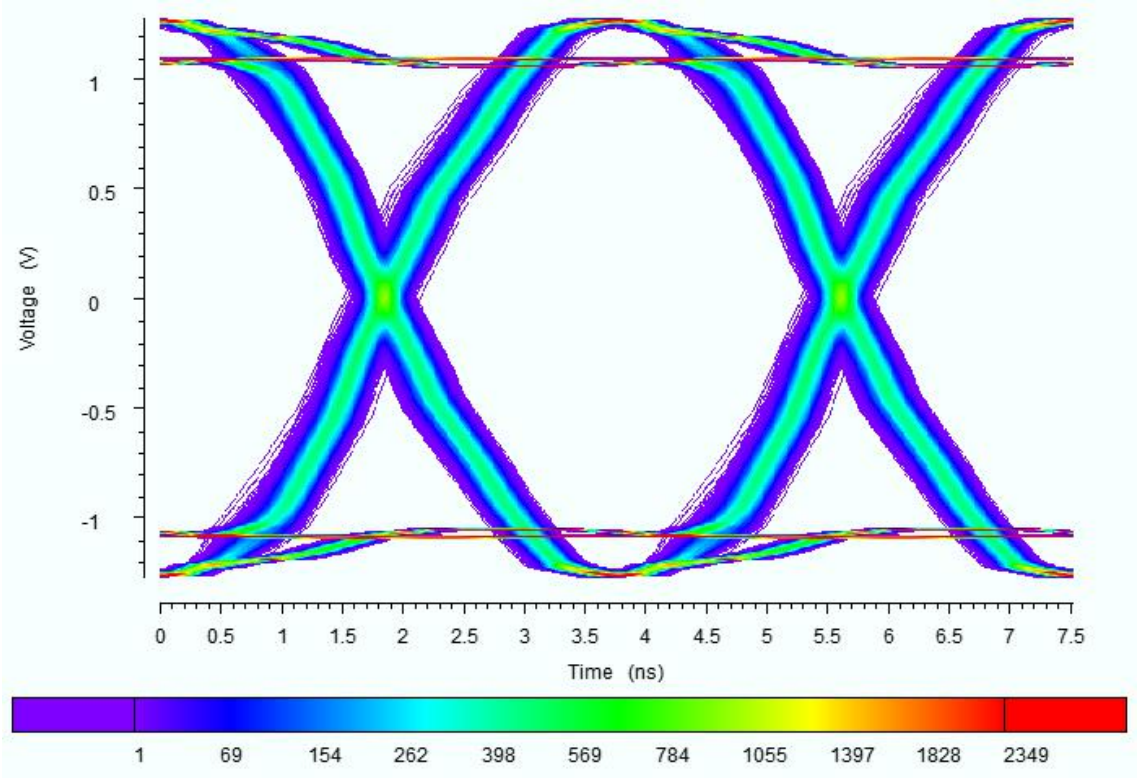


Figure 4.4: Example of an Eye diagram

4.2 Transient Response

As for the Data generation of the transient response, we took advantage of the "Parameter Sweeping" option provided by the eCADSTAR tool to generate many data points in one simulation. The goal was to generate 600 different data points for each of the Single-ended and Differential pair.

However, the maximum number of tasks that can be executed to generate data in 1 parameter sweep simulation is limited to 300. Therefore generating the data had to be done over 2 simulations for each of the Single-Ended and the Differential Pair topologies. Running the simulations for the differential pair in particular was relatively time-consuming and heavy from the computational POV.

In order to match the values generated in the eye-diagram and ensure consistency, the lower boundary of the simulation values was always 50 and the upper bound was 200. This applies to the transmission-line lengths, impedance's values and resistance values.

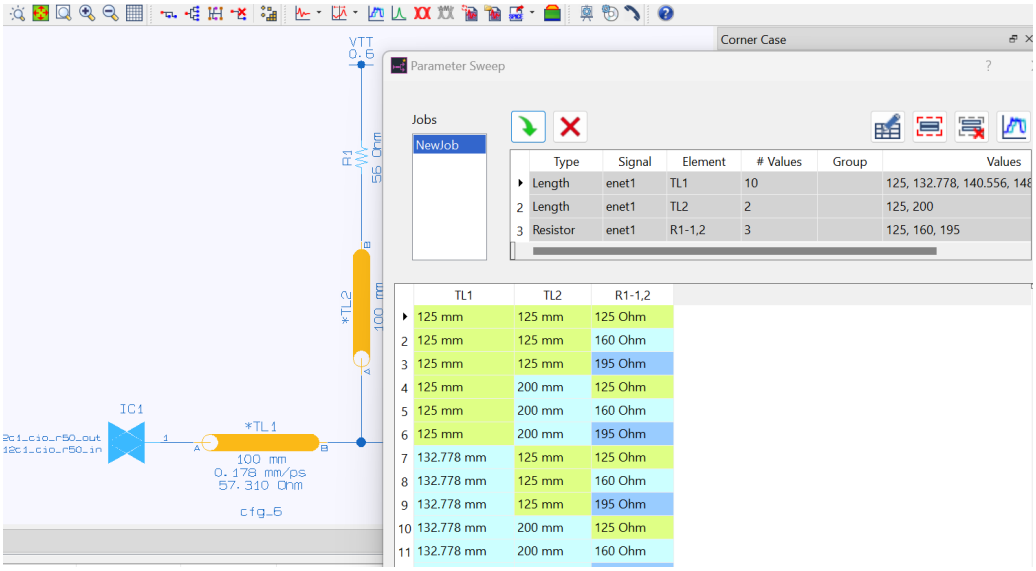


Figure 4.5: Parameter Sweep of e-CADSTAR

The 2 parameter sweep simulations in the Single-ended topology resulted in 1200 data points (600 for IC1 and 600 for IC2). While for the Differential pair, it resulted in 2400 data points (600 for IC1(1), 600 for IC1(2), 600 for IC2(1), and 600 for IC2(2)).

The big challenge was then to organize all these huge amount for data in a way that makes it usable for machine learning purposes. And that what is going to be discussed in the next chapter "Data Preprocessing".

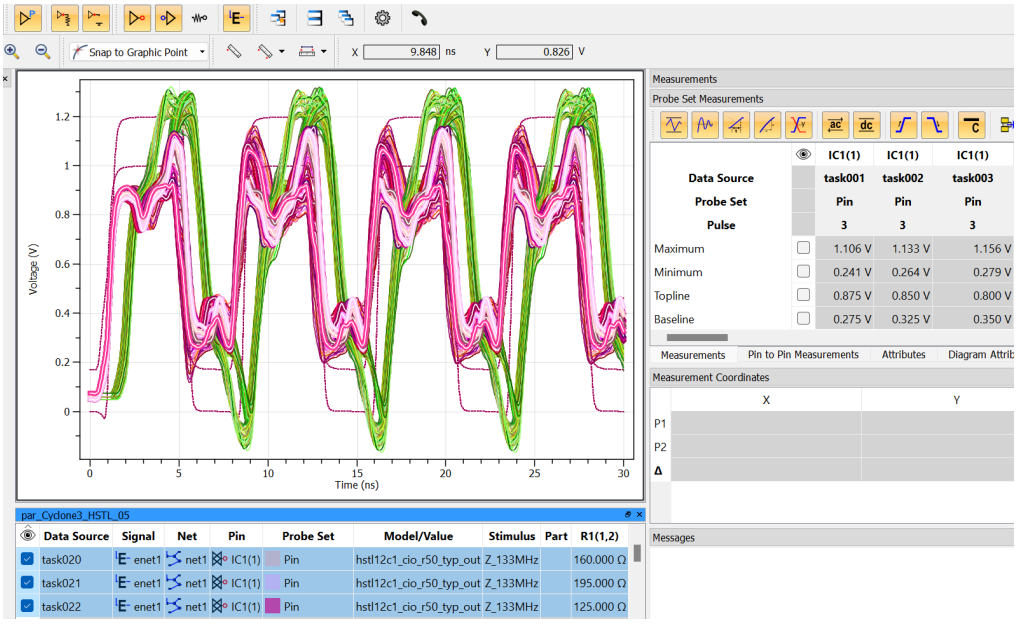


Figure 4.6: Results of a Simulation

5 Data Preprocessing

An essential step in getting datasets ready for machine learning models is data preparation. Preprocessing guarantees that the data is clear, consistent, and prepared for additional analysis in the context of signal integrity analysis for PCB design.

5.1 Preprocessing for Eye Diagram

Adequate preprocessing of the parameter inputs is essential to obtain relevant insights in the pursuit of proper Eye Diagram generation.

5.1.1 Standardization of Units

Parameters like width are recorded in the dataset derived from the simulation results utilising different units, such as picoseconds (ps) and nanoseconds (ns). Consistent analysis is complicated by this unit disparity. One important preprocessing step to address this is to standardise all units to guarantee consistency throughout the dataset. The dataset is made uniform and comprehensible by uniting all width measurements to a single unit, usually nanoseconds (ns). This creates the foundation for more thorough analysis and modelling.

5.1.2 Handling Missing Values

Addressing absent or blank entries in the dataset is a crucial component of data preprocessing. It is not unusual to find eye diagrams in our collection without the height and width variables. But missing data is something that predictive models cannot handle, therefore there needs to be a plan in place for when this happens. Zeros are assigned to missing values in order to fix this problem. This methodology guarantees the dataset's continued completion and suitability for analysis, reducing biases and safeguarding the accuracy of ensuing modelling attempts.

5.1.3 Data Transformation

The dataset for each IC is organised into a single Excel page where related outputs and input parameters are arranged in a deliberate manner next to each other. Every simulation iteration may easily refer to and retrieve the input and output data points thanks to this organised format. When all pertinent data is gathered into a single Excel sheet, it becomes easier to search through the dataset and facilitates a more efficient workflow for further analysis and machine learning modelling. The data feeding procedure into machine learning models is made simpler when input parameters and outputs are aligned within the same Excel page. We can easily extract and feed the dataset straight into ML algorithms without requiring any data modification or restructure because inputs and outputs are combined in adjacent columns. This simplified method ensures the integrity and correctness of the ML analysis by reducing the possibility of data mismatch or errors during the phase of preprocessing.

In conclusion, in order to prepare datasets for predictive models , data preprocessing is essential. We make sure the dataset is clean, consistent, and prepared for additional research by standardising units, managing missing values, and using data transformation tools. In order to maximise PCB design and improve our knowledge of signal integrity, these preprocessing processes are crucial for improving model performance and enabling more precise predictions.

5.2 Transient Response

After generating 1200 data points for single-endend and 2400 for differential pair, organizing the data was done on multiple steps.

The first step was to copy the values from the e-CADSTAR tool and paste in in our excel file. The second step was to divide the excel feels properly into sheets, and assign each pin to a sheet (e.g. Sheet for IC1, sheet for IC2). The third step was to drop the unnecessary measurements (according to the supervisor's instructions).

Afterwards, the measurements units of all data points needed to be unified (e.g. some were V/ns and some were V/s). Last but not least, rows (data points) with an empty column (measurement) were dropped. As for (IC2)s there were rarely any empty columns, but for (IC1)s there were often some missing measurements here and there.

	A	B	C	D	E	F
1	Data Source	TL1 Length (mm)	TL2 Length (mm)	R1 (Ohm)	TL1 Impedance (Ohm)	TL2 Impedance (Ohm)
2	task001	50	50	50	57.31	57.31
3	task002	50	50	85	57.31	57.31
4	task003	50	50	120	57.31	57.31
5	task004	50	57.7778	50	57.31	57.31
6	task005	50	57.7778	85	57.31	57.31
7	task006	50	57.7778	120	57.31	57.31
8	task007	50	65.5556	50	57.31	57.31
9	task008	50	65.5556	85	57.31	57.31
10	task009	50	65.5556	120	57.31	57.31
11	task010	50	73.3333	50	57.31	57.31
12	task011	50	73.3333	85	57.31	57.31
13	task012	50	73.3333	120	57.31	57.31
14	task013	50	81.1111	50	57.31	57.31
15	task014	50	81.1111	85	57.31	57.31
16	task015	50	81.1111	120	57.31	57.31
17	task016	50	88.8889	50	57.31	57.31
18	task017	50	88.8889	85	57.31	57.31
19	task018	50	88.8889	120	57.31	57.31
20	task019	50	96.6667	50	57.31	57.31
21	task020	50	96.6667	85	57.31	57.31
22	task021	50	96.6667	120	57.31	57.31
23	task022	50	104.444	50	57.31	57.31

ushmitha | Transient Single | IC1 | IC2 | Transient Diff. | IC1(1) | IC2(1)

Figure 5.1: Overview of the Input Data

	A	B	C	D	E	F	G	H
1	Pin	Data Source	Maximum (V)	Minimum (V)	Overshoot 1st Rising (V)	Rise Time 20/80 (ns)	Fall Time 20/80 (ns)	Slew Rate Setup Rising (V/ns)
2	IC2(1)	task001	1.039	0.128	0.04071	0.505192	0.512208	1.259
3	IC2(1)	task002	1.088	0.067	0.018909	0.546411	0.544652	1.221
4	IC2(1)	task003	1.14	0.013	0.036667	0.558346	0.556878	1.117
5	IC2(1)	task004	1.045	0.123	0.046017	0.503467	0.509843	1.268
6	IC2(1)	task005	1.084	0.072	0.014729	0.555798	0.553319	1.197
7	IC2(1)	task006	1.135	0.018	0.032078	0.573477	0.571603	1.083
8	IC2(1)	task007	1.049	0.118	0.050789	0.50247	0.508419	1.274
9	IC2(1)	task008	1.078	0.078	0.008374	0.564327	0.561442	1.176
10	IC2(1)	task009	1.132	0.019	0.029186	0.590378	0.586762	1.05
11	IC2(1)	task010	1.053	0.114	0.054779	0.501929	0.507702	1.277
12	IC2(1)	task011	1.083	0.077	0.014008	0.569357	0.567553	1.157
13	IC2(1)	task012	1.142	0.012	0.038701	0.605	0.6	1.022
14	IC2(1)	task013	1.058	0.109	0.059737	0.501046	0.50722	1.277
15	IC2(1)	task014	1.091	0.072	0.021217	0.570007	0.572196	1.143
16	IC2(1)	task015	1.154	0.004	0.050805	0.608	0.607	1.005
17	IC2(1)	task016	1.061	0.106	0.062612	0.500244	0.50692	1.273
18	IC2(1)	task017	1.093	0.07	0.023449	0.568603	0.574313	1.148
19	IC2(1)	task018	1.157	0.002	0.054412	0.599	0.609	1.02
20	IC2(1)	task019	1.065	0.102	0.066183	0.499212	0.506561	1.273
21	IC2(1)	task020	1.092	0.071	0.02314	0.567631	0.578337	1.149
22	IC2(1)	task021	1.157	0.001	0.054389	0.591	0.623	1.025

ushmitha | Transient Single | IC1 | IC2 | Transient Diff. | IC1(1) | IC2(1) | IC1(2) | IC2(2)

Figure 5.2: Overview of the Output Data

The last step before starting with the ML part was to combine the inputs values (of the simulation) and the output values (obtained) in one sheet for each pin (IC1, IC2). The motivation behind that was to facilitate the loading of data into our ML code (which will be discussed in depth later).

	A	B	C	D	E	F	G	H	I	J	K
1	TL1 Length (mm)	TL2 Length (mm)	R1 (Ohm)	TL1 Impedance (Ohm)	TL2 Impedance (Ohm)	Maximum (V)	Minimum (V)	Overshoot 1st Rising (V)	Rise Time 20/80 (ns)	Fall Time 20/80 (ns)	Slew Rate Setup Risir
2	50	50	50	57.31	57.31	1.039	0.128	0.04071	0.505192	0.512208	1.259
3	50	50	85	57.31	57.31	1.088	0.067	0.018909	0.546411	0.544652	1.221
4	50	50	120	57.31	57.31	1.14	0.013	0.036667	0.558346	0.556878	1.117
5	50	57.7778	50	57.31	57.31	1.045	0.123	0.046017	0.503467	0.509843	1.268
6	50	57.7778	85	57.31	57.31	1.084	0.072	0.014729	0.555798	0.553319	1.197
7	50	57.7778	120	57.31	57.31	1.135	0.018	0.032078	0.573477	0.571603	1.083
8	50	65.5556	50	57.31	57.31	1.049	0.118	0.050789	0.50247	0.508419	1.274
9	50	65.5556	85	57.31	57.31	1.078	0.078	0.008374	0.564327	0.561442	1.176
10	50	65.5556	120	57.31	57.31	1.132	0.019	0.029186	0.590378	0.586762	1.05
11	50	73.3333	50	57.31	57.31	1.053	0.114	0.054779	0.501929	0.507702	1.277
12	50	73.3333	85	57.31	57.31	1.083	0.077	0.014008	0.569357	0.567553	1.157
13	50	73.3333	120	57.31	57.31	1.142	0.012	0.038701	0.605	0.6	1.022
14	50	81.1111	50	57.31	57.31	1.058	0.109	0.059737	0.501046	0.50722	1.277
15	50	81.1111	85	57.31	57.31	1.091	0.072	0.021217	0.570007	0.572196	1.143
16	50	81.1111	120	57.31	57.31	1.154	0.004	0.050805	0.608	0.607	1.005
17	50	88.8889	50	57.31	57.31	1.061	0.106	0.062612	0.500244	0.50692	1.273
18	50	88.8889	85	57.31	57.31	1.093	0.07	0.023449	0.568603	0.574313	1.148
19	50	88.8889	120	57.31	57.31	1.157	0.002	0.054412	0.599	0.609	1.02
20	50	96.6667	50	57.31	57.31	1.065	0.102	0.066183	0.499212	0.506561	1.273
21	50	96.6667	85	57.31	57.31	1.092	0.071	0.02314	0.567631	0.578337	1.149
22	50	96.6667	120	57.31	57.31	1.157	0.001	0.054389	0.591	0.623	1.025
23	50	104.444	50	57.31	57.31	1.067	0.101	0.067974	0.498711	0.506687	1.278
24	50	104.444	85	57.31	57.31	1.091	0.071	0.021291	0.570382	0.581411	1.146
25	50	104.444	120	57.31	57.31	1.159	0	0.056097	0.598	0.635	1.007
26	50	112.222	50	57.31	57.31	1.068	0.1	0.06916	0.498194	0.506929	1.281
27	50	112.222	85	57.31	57.31	1.095	0.069	0.02541	0.575139	0.582385	1.133

Figure 5.3: Overview of final data after preprocessing

6 Machine Learning

6.1 ANN Regression for Eye Diagram Measurements

Neural network, in simple terms, is a Machine Learning technique or algorithm that tries to mimic the working of neurons in the human brain for learning. A neural network with more than one hidden layer is a Deep Neural Network and the learning method is called deep learning. Each layers have an arbitrary number of nodes. The inputs are multiplied with weights in the network and biases are added in each layer in every node. An activation function f(z) such as sigmoid, linear, tan hyperbolic etc., is added before transferring it to any further layer. Initially, the model has the wrong values of biases and weights, resulting in high losses/errors. So, we train the model by finding a suitable cost function and update the weights and biases and try to minimise the errors. Some common methods for this are backpropagation and gradient descent.

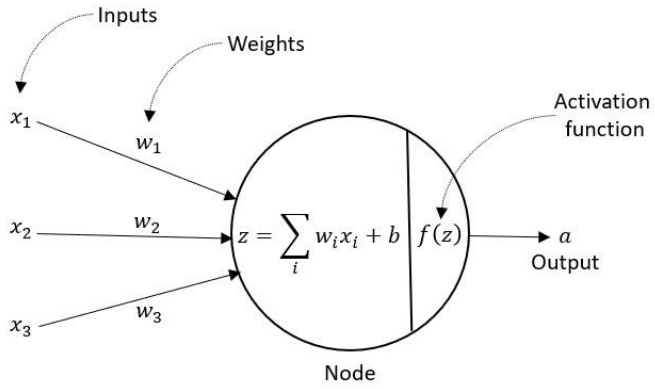


Figure 6.1: A Single Node in a Neural Network

There are many libraries available on Python to implement a ML network and they can be easily imported and accessed in Google Colab. Some of these are Keras, TensorFlow, NumPy, Pandas etc. For our implementation, we mainly require the following libraries:

- numPy for handling numerical values and float data.
- matplotlib for plotting scatter plots and graphs.

• pandas for handling data analysis and reading the dataset.

• tensorflow to implement and access the neural network.

The very first step is to upload the data set. There are many ways to do this: we can either mount Google Drive and provide the link to access the corresponding data, or we can manually upload it to Google Colab by importing the library known as files. The data set that is used can be in either .csv format or .xlsx(excel) format. In our implementation of SI Analysis, eye diagram data was generated using Zuken eCADSTAR software. The obtained values of Eye diagram measurements are stored in a .xlsx file. The data is generated for both Differential pair and Single-ended configurations.

The data used for training and testing contains the following information:

• DL1/TL1 Length (mm)

• DL1/TL1 Impedance (Ohm)

• DL2/TL2 Length (mm)

• DL2/TL2 Impedance (Ohm)

• R9 and R8/R1 (Ohm)

The above parameters are considered as predictors or input parameters. The output parameters or target values are: Zero level(V), One level(V), Mean level(V), Amplitude(V), Height(V), Width(ns), Opening Factor, Signal-to-noise ratio and Crossing Percentage. The data set also contains Rise Time and Fall Time values in nanoseconds. Since our aim is to purely focus on eye opening and train the model to predict the same, we do not consider the rise times and fall times. All of the above-mentioned parameters are recorded for two points in the circuit: IC1 and IC2.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	DL1 Length (n	DL1 Impedanc	DL2 Length (n	DL2 Impedanc	R9 & R8 (Ohm	Zero level (V)	One Level (V)	Mean Level (V	Amplitude (V)	Height (V)	Width (ns)	Opening Fact	Signal to Nois	Crossing Per	Rise Time 10/	Fall Time 10/9	Rise Time 20/	Fall Time 20/80 (ns)	
2	200	123.852	100	101.043	100	-0.856	0.859	0.002	1.715	0.763	1.988	0.815	5.403	49.732	0.342	0.381	0.857	0.863	
3	100	123.852	100	101.043	100	-0.836	0.838	0.001	1.674	0.64	2.428	0.794	4.854	49.697	0.121	0.047	0.715	0.716	
4	200	123.852	200	101.043	100	-0.861	0.866	0.002	1.726	0.923	1.916	0.845	6.447	49.707	0.946	1.001	0.748	0.751	
5	200	123.852	200	101.043	200	-0.846	0.845	-0.001	1.69	0.248	1.07	0.716	3.517	49.62	0.362	0.381	0.865	0.866	
6	200	123.852	100	101.043	200	-0.832	0.83	-0.001	1.661	0	1.293	0.643	2.804	49.642					
7	100	123.852	100	101.043	200	-0.851	0.849	-0.001	1.7	0.114	2.383	0.689	3.215	49.682			0.465	0.405	
8	100	123.852	200	101.043	200	-0.777	0.774	-0.001	1.551	0	1.449	0.565	2.299	49.634					
9	100	123.852	200	101.043	100	-0.806	0.81	0.002	1.616	0.4	2.041	0.749	3.987	49.722			0.805	0.81	
10	200	123.852	50	101.043	100	-0.856	0.858	0.001	1.714	2.186	0.814	5.376	49.735	49.735	0.037	0.085	0.809	0.809	
11	200	123.852	50	101.043	50	-0.865	0.878	0.006	1.743	1.535	2.446	0.96	25.177	49.89	1.084	1.084	0.716	0.709	
12	100	123.852	50	101.043	100	-0.892	0.894	0.001	1.786	1.039	2.475	0.861	7.18	49.677	1.188	1.146	0.722	0.722	
13	100	123.852	50	101.043	50	-0.817	0.825	0.004	1.642	1.496	2.465	0.97	33.709	49.789	0.979	0.978	0.641	0.636	
14	100	123.852	50	101.043	200	-0.948	0.948	0	1.896	0.825	2.49	0.812	5.312	49.647	0.793	0.75	1.746	1.735	
15	200	123.852	50	101.043	200	-0.828	0.826	-0.001	1.654	0	1.854	0.637	2.753	49.669			0.036	0.03	
16	200	123.852	50	101.043	150	-0.84	0.839	0	1.68	0.194	2.001	0.705	3.391	49.689			0.697	0.708	
17	100	123.852	50	101.043	150	-0.928	0.929	0	1.857	0.898	2.484	0.828	5.811	49.656	0.999	0.959	0.741	0.754	
18	50	123.852	200	101.043	100	-0.838	0.841	0.001	1.679	0.689	2.353	0.803	5.087	49.709	1.387	1.21	1.026	1.046	

Figure 6.2: Data set for Eye Measurements

An algorithm as shown in Figure 6.3 was created defining the complete procedure of the code for easier interpretation.

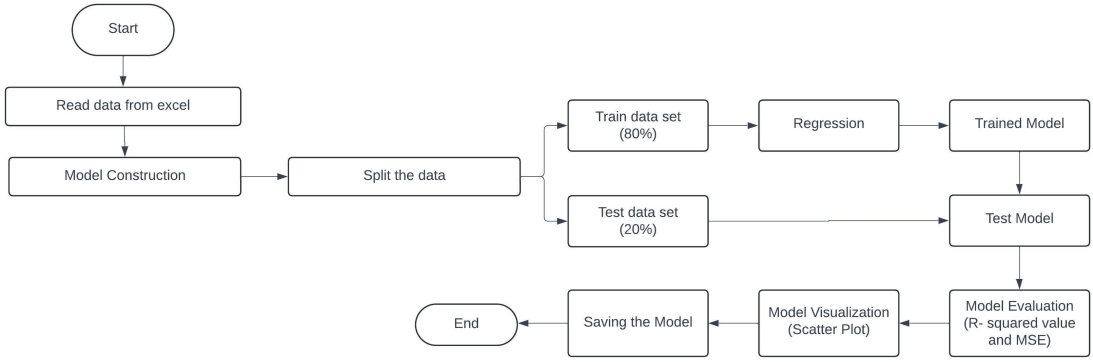


Figure 6.3: Algorithm for ANN Regression

Firstly, the dataset is loaded into a data frame. It is then read by using the pandas library. Since IC1 and IC2 values are contained in different sheets in the excel file, they can be accessed simply by entering the corresponding sheet name. We then split the columns as inputs and outputs and load them into separate variables X and Y.

The next step is to split the data into training and testing data. The standard norm that is followed is to split the data into 70% training data and 30% testing data. It can also be 80% training and 20% testing. However, it has to be noted that the training data must always be significantly more. Therefore, we import train_test_split library. In addition to this, we will also need to normalize the data. So, we import another library called StandardScaler. We convert the data to a matrix format and then denormalize it by using inverse_transform. This results in the original scale of the values. This step is performed to reduce losses.

```
# Normalization of input
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Normalization of output
scaler_y = StandardScaler()
y_normalized = scaler_y.fit_transform(y_ic1)

# Spliting the data into training and testing sets
X_train_ic1, X_test_ic1, y_normalized_train, y_normalized_test = train_test_split(X_normalized, y_normalized,
                                                                                  test_size=0.2, random_state=42)
```

Figure 6.4: Normalization and Data Split

The above steps are for data handling and preparation for the neural network. The next step is to define the layers of the model, compile it and fit it accordingly. From the keras library under tensorflow, we import Sequential and Dense, which are well-suited to handle deep learning architectures. Then we add the number of units, dimensions, activation and kernel initializer accordingly. The same is done for the other layers as well. Finally, we

add the output layer and then compile the model by choosing a suitable optimizer and loss function. Typically, we can go for mean squared error loss and AdaM (Adaptive Moment) optimizer. We can even resort to Gradient Descent optimizer. Then we fit the ANN model to the training data by choosing a suitable number of epochs and batch size. 1 Epoch = Full forward data propagation along with back propagation.

```
# Model Construction
model_ic1 = keras.Sequential([
    keras.layers.Dense(256, activation='relu', input_shape=(5,), name='input_layer'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(128, activation='relu', name='hidden_layer1'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(256, activation='relu', name='hidden_layer2'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(64, activation='relu', name='hidden_layer3'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(256, activation='relu', name='hidden_layer4'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation='relu', name='hidden_layer5'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(9, name='output_layer_ic1')
], name='FFN_model_ic1')
```

Figure 6.5: Model Construction

After this, we can predict the data and denormalize it to plot the graphs using matplotlib. The graphs can be analysed and we can infer how well the model fitting process has been carried out. Then we can run the model by using only the testing data and it gives a list of possible outputs. These can be plotted on scatter plots or regular 2D plots to analyse them better. The model is further evaluated by calculating the r-squared and mean square error value.

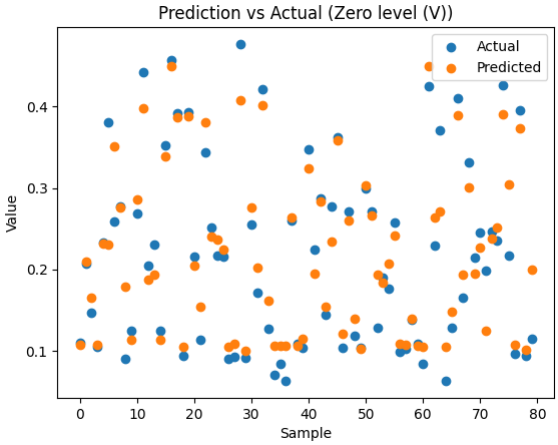


Figure 6.6: Scatter plot for eye diagram results

- R-squared (R2 score): 0.8118504959874896
- Mean Squared Error (MSE): 3.2143073

6.2 ANN Regression for Transient Measurements

6.2.1 Single Ended

As for the single ended, it was decided to built two separate ML models, one for IC1 and the other for IC2. Since both models are almost identical, only the IC2 model will be discussed as an example here in depth.

Implementation

Importing Libraries First, we import the necessary libraries. These libraries provide functionalities for data manipulation, preprocessing, model building, training, and evaluation.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Loading the Data We load the dataset from a CSV file. This file contains the input features and output labels necessary for training and testing the ANN model.

```
data = pd.read_csv('/content/Single IC2 test.csv')
```

Data Preparation We separate the input features (X) and output labels (y) from the dataset. Then, we split the data into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

```
X = data.iloc[:, :5].values # First 5 columns as input features
y = data.iloc[:, 5:].values # Last 6 columns as output labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
```

Data Preprocessing To improve the performance of the neural network, we scale the input and output data. Scaling ensures that the data has a mean of zero and a standard deviation of one, which helps the neural network converge faster during training.

```
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
y_train_scaled = scaler_y.fit_transform(y_train)
X_test_scaled = scaler_X.transform(X_test)
y_test_scaled = scaler_y.transform(y_test)
```

Building the ANN Model We construct the ANN model using the Sequential API from Keras. The model consists of two hidden layers with 64 neurons each and ReLU activation functions. The output layer has 6 neurons, corresponding to the 6 output values.

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(64, activation='relu'),
    Dense(6) # Output layer with 6 neurons for 6 output values
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Training the Model Train the model using the training data. The training process runs for 300 epochs with a batch size of 32. A validation split is used to monitor the model's performance on a subset of the training data that is not used for training. The Training-Testing split was the standard 80-20 ratio.

```
history = model.fit(X_train_scaled, y_train_scaled, epochs=300, batch_size=32,
    validation_split=0.2)
```

Model Evaluation Evaluate the model's performance on the test data using Mean Squared Error (MSE) and Mean Absolute Error (MAE) as metrics.

```
mse, mae = model.evaluate(X_test_scaled, y_test_scaled)
```

Making Predictions Use the trained model to make predictions on the test data. The predicted values are then inverse transformed to their original scale.

```
predictions_scaled = model.predict(X_test_scaled)
predictions = scaler_y.inverse_transform(predictions_scaled)
```

Visualizing the Results Plot the training and validation loss to visualize the model's performance over the epochs.

```
import matplotlib.pyplot as plt

# Extract the training and validation loss from the history
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot the training and validation loss
epochs = range(1, len(train_loss) + 1)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Plot the predicted and actual values for each output variable to visualize the model's performance:

```
# Example for a scatter plot
plt.scatter(y_test[:, 0], predictions[:, 0], label='Maximum (V)')
plt.scatter(y_test[:, 1], predictions[:, 1], label='Minimum (V)')
plt.scatter(y_test[:, 2], predictions[:, 2], label='Overshoot 1st Rising (V)')
plt.scatter(y_test[:, 3], predictions[:, 3], label='Rise Time 20/80 (ns)')
plt.scatter(y_test[:, 4], predictions[:, 4], label='Fall Time 20/80 (ns)')
plt.scatter(y_test[:, 5], predictions[:, 5], label='Slew Rate Setup Rising (V/ns)
    ')
plt.xlabel('Ground Truth')
plt.ylabel('Predicted')
plt.legend()
plt.show()
```

Evaluation Metrics Calculate and print evaluation metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R-squared score (R2) to quantitatively assess the model's performance.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Calculate the evaluation metrics
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
```

```
r2 = r2_score(y_test, predictions)

# Print the metrics
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
```

Results

The performance of the Artificial Neural Network (ANN) model was evaluated using several metrics, including Loss, Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2) score. Below are the results and their interpretations:

- **Training Metrics**:
  - **Training Loss**: 0.0378
  - **Training MAE**: 0.0653

  These values indicate that the model has a reasonably good fit on the training data, with the predictions closely matching the actual values.

- **Validation Metrics**:
  - **Validation Loss**: 0.0608
  - **Validation MAE**: 0.0798

  The validation metrics are slightly higher than the training metrics, suggesting some level of overfitting. However, the model's performance on the validation data is still acceptable.

- **Test Metrics**:
  - **Test Loss**: 0.0168
  - **Test MAE**: 0.0631

  The lower test loss and comparable test MAE indicate that the model generalizes well to unseen data, performing better on the test set than on the training and validation sets.

- **Additional Evaluation Metrics**:
  - **Mean Squared Error (MSE)**: 0.004322

– **Mean Absolute Error (MAE)**: 0.0200

– **R^2 Score**: 0.9781

The low MSE and MAE values confirm that the model's predictions are very close to the true values. The high R^2 score of approximately 0.978 indicates that the model explains a significant proportion of the variance in the test data.

Overall, the evaluation metrics suggest that the ANN model is effective in predicting the output variables, demonstrating good performance and generalization capabilities.

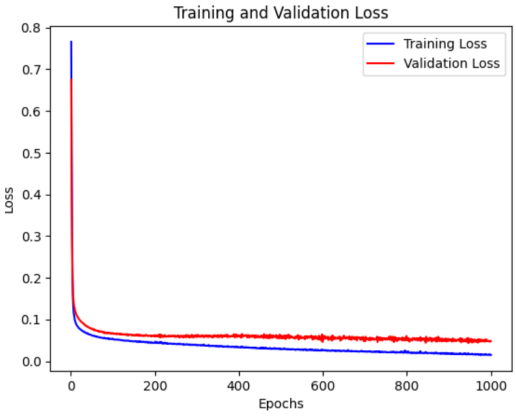


Figure 6.7: Overview of the Output Data

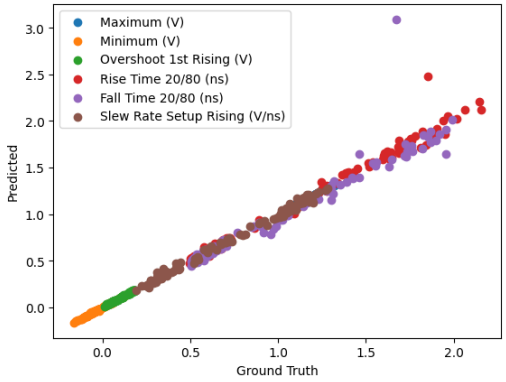


Figure 6.8: Model Evaluation 2

Figure 6.9: Model Performance

6.2.2 Differential Pair

Regression analysis is an essential component of predictive modelling since it offers valuable insights into the correlations between various variables. Specifically for regression problems, we build, train, assess, and demonstrate a neural network model, and we explore deeply into its details. Our goal is to build a model that accurately predicts output variables from input attributes, with the trustworthy TensorFlow and Keras frameworks at.

The first step in our procedure is importing the libraries necessary for data processing, mathematical operations, model construction, and visualisation. Pandas is used to process the data after it has been extracted from an Excel file. After that, the data is normalised and the input and output features are extracted using StandardScaler. The neural network model built with Keras is composed of an input layer with six neurons and four hidden layers, each with 64 neurons that are activated by ReLU , as illustrated in fig 6.10. The model is compiled using the Adam optimizer and the MSE loss function.

```
model_ic1_1 = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(6,),
                       name='input_layer'),
    keras.layers.Dense(64, activation='relu', name='hidden_layer1'),
    keras.layers.Dense(64, activation='relu', name='hidden_layer2'),
    keras.layers.Dense(64, activation='relu', name='hidden_layer3'),
    keras.layers.Dense(64, activation='relu', name='hidden_layer4'),
    keras.layers.Dense(5, name='output_layer_ic1')
], name='FFN_model_ic1')

model_ic1_1.compile(optimizer='adam', loss='mse')

X_train_ic1_1, X_test_ic1_1, y_ic1_1_train, y_ic1_1_test =
train_test_split(X_scaled, y_ic1_1_scaled, test_size=0.2, random_state=42)

history_ic1_1 = model_ic1_1.fit(X_train_ic1_1, y_ic1_1_train,
                  epochs=200, validation_data=(X_test_ic1_1, y_ic1_1_test))
```

Figure 6.10: Model Architecture

After that, the dataset is split into training (80%) and testing (20%) sets to allow for a comprehensive evaluation of the algorithm's generalisation abilities. Validation data is used for monitoring while an extensive training regimen spanning 200 epochs is employed to create the model on the training set. After training, the model's efficiency on testing data is evaluated using indicators like MSE, MAE, and R2 score, which provide important insights into expected accuracy. The model's behaviour can be better understood using graphical aids, such as the comparison of actual and expected values for each output variable in fig. 6.11.

Evaluation metrics, in fig. 6.12, offer valuable insights into the model's effectiveness and serve as reliable gauges of both empirical integrity and prediction accuracy. Visualisations provide a greater understanding of the algorithm's predictive power by elucidating discrepancies between expected and actual values for a variety of output variables. The model can be considered to be a good fit because the MAE and MSE are too low and the R-squared value is almost equal to 1. Applying the developed neural network model to regression analysis tasks yields promising results. Regression model creation and evaluation now has a strong framework due to our systematic method, which combines data preprocessing, training, evaluation, and visualisation.

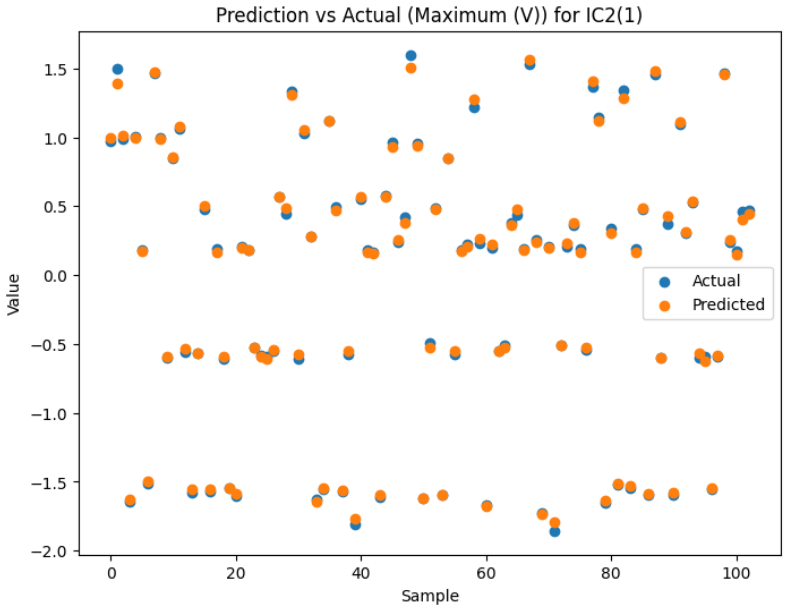


Figure 6.11: Scatter Plot Example (Maximum Voltage for IC2(1))

```
Mean Squared Error (MSE): 0.009496744
4/4 [==============================] - 0s 3ms/step
Mean Absolute Error (MAE): 0.03892922
4/4 [==============================] - 0s 4ms/step
R-squared (R2) Score: 0.9899527592395225
```

Figure 6.12: Evaluation Metrics - MSE, MAE and R2 Scores for IC1(2)

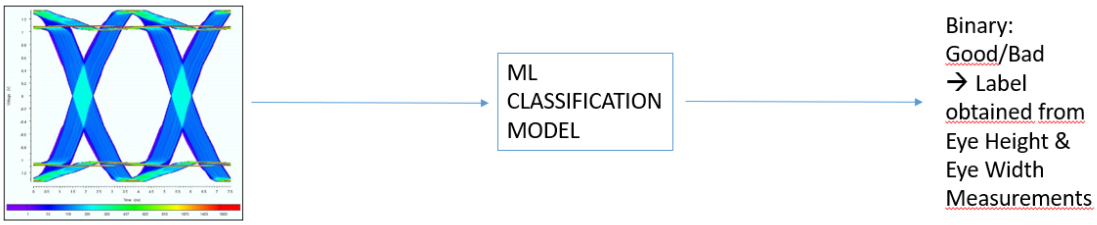
6.3 Classification for Eye Diagram Images

In this section, we present the development and implementation of a Convolutional Neural Network (CNN) designed for the binary classification of eye-diagram images. The objective of this model is to classify eye diagrams based on whether they meet predefined performance thresholds, specifically a height threshold of 0.5V and a width threshold of 1.88 ns.

6.3.1 Model Architecture

The CNN architecture comprises multiple layers designed to capture and analyze the intricate patterns within eye-diagram images. The architecture includes:

1. Input Layer: Accepts eye-diagram images of a fixed size.



2. Convolutional Layers: Extract feature maps using a series of convolutional operations with ReLU activation functions.
3. Pooling Layers: Downsample the feature maps to reduce dimensionality and computation, while retaining critical features.
4. Fully Connected Layers: Flatten the pooled feature maps and pass them through dense layers to learn higher-level representations.
5. Output Layer: A sigmoid activation function is applied to produce a binary classification output indicating whether the eye diagram meets the set thresholds.

Total number of layers: 17

6.3.2 Model Training

The model was initially trained without early stopping. However, the validation loss did not decrease, indicating potential overfitting. To address this, we introduced the EarlyStopping callback to halt training when the validation accuracy did not improve for 15 consecutive epochs.

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# For extracting more features, I am adding more number of convolutional layers.
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flatten converts the image matrix to a 1-D vector.
model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
# Below dense function manages the output neuron.
model.add(Dense(1, activation='sigmoid'))

# Configuring the model
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Figure 6.13: Model Architecture

```
# EarlyStopping callback to prevent overfitting
callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    min_delta=0,
    patience=15,
    verbose=1,
    mode='max',
    restore_best_weights=True)

# Model training with EarlyStopping callback
history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size,
    callbacks=[callback])
```

Figure 6.14: Introduction of Callback function

6.3.3 Data Preparations

The dataset for training and validation was generated through simulation, comprising eye-diagram images labeled according to the specified thresholds. Images that met both the height and width thresholds were labeled as '1' (acceptable), while those that did not were labeled as '0' (unacceptable).

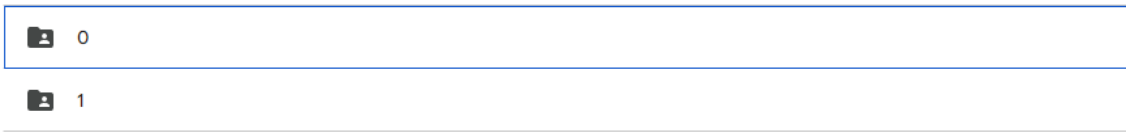


Figure 6.15: Data preparation for 1 and 0

6.3.4 Layers in Model

1. Conv2D Layer: Convolutional layer with 32 filters and a 3x3 kernel size.
2. BatchNormalization Layer: Normalizes the output of the previous layer.
3. MaxPooling2D Layer: Pooling layer with a 2x2 pool size.
4. Dropout Layer: Dropout layer with a dropout rate of 0.25.
5. Conv2D Layer: Convolutional layer with 64 filters and a 3x3 kernel size.
6. BatchNormalization Layer: Normalizes the output of the previous layer.
7. MaxPooling2D Layer: Pooling layer with a 2x2 pool size.
8. Dropout Layer: Dropout layer with a dropout rate of 0.25.
9. Conv2D Layer: Convolutional layer with 128 filters and a 3x3 kernel size.
10. BatchNormalization Layer: Normalizes the output of the previous layer.
11. MaxPooling2D Layer: Pooling layer with a 2x2 pool size.
12. Dropout Layer: Dropout layer with a dropout rate of 0.25.
13. Flatten Layer: Flattens the input.
14. Dense Layer: Fully connected layer with 512 units and ReLU activation.
15. BatchNormalization Layer: Normalizes the output of the previous layer.
16. Dropout Layer: Dropout layer with a dropout rate of 0.5.

17. Dense Layer: Fully connected layer with 1 unit and sigmoid activation (output layer).

6.3.5 Hidden Layers

Hidden layers are those layers that are neither input nor output layers. They process the data between input and output layers.

- Input Layer: Implicitly defined by the `input_shape` in the first Conv2D layer.
- Hidden Layers: Layers 1 to 16 (all layers except the last Dense layer).
- Output Layer: The last Dense layer (Dense(1, activation='sigmoid')).

Hidden Layers Breakdown

- Conv2D Layers: 3
- BatchNormalization Layers: 3
- MaxPooling2D Layers: 3
- Dropout Layers: 3
- Flatten Layer: 1
- Dense Layer: 1 (the first Dense layer before the output layer)
- Additional BatchNormalization and Dropout layers within the fully connected part: 1 each

Total Hidden Layers: 15

Summary

- Total Layers: 17
- Hidden Layers: 15

6.3.6 Training and Validation

The model was trained using a labeled dataset, with a portion of the data set aside for validation. The training process involved optimizing the CNN parameters to minimize binary cross-entropy loss, a suitable metric for binary classification tasks. Additionally,

various data augmentation techniques were employed to enhance the robustness of the model.

The training was conducted under three different scenarios:

1. Single-Ended Pair Images: Training exclusively on eye-diagram images from single-ended pairs.
2. Differential Pair Images: Training exclusively on eye-diagram images from differential pairs.
3. Combined Images: Training on a combined dataset of both single-ended and differential pair images.

Figure 6.16: Data set per each model

6.3.7 Results

Upon completion of the training process, the performance of the CNN model was evaluated based on its accuracy and loss metrics on both the training and validation datasets for each of the three scenarios. The results are as follows:

- Training Accuracy (Combined): 0.9708
- Validation Accuracy (Combined): 0.9667
- Training Loss (Combined): 0.0795
- Validation Loss (Combined): 0.1012

```
=================================================================
Total params: 19037121 (72.62 MB)
Trainable params: 19035649 (72.62 MB)
Non-trainable params: 1472 (5.75 KB)
_________________________________________________________________
Found 1527 images belonging to 2 classes.
Found 71 images belonging to 2 classes.
Epoch 1/30
76/76 [==============================] - 140s 2s/step - loss: 0.4816 - accuracy: 0.8520 - val_loss: 46.0982 - val_accuracy: 0.4833
Epoch 2/30
76/76 [==============================] - 133s 2s/step - loss: 0.3130 - accuracy: 0.8852 - val_loss: 47.7400 - val_accuracy: 0.4500
Epoch 3/30
76/76 [==============================] - 135s 2s/step - loss: 0.2604 - accuracy: 0.9071 - val_loss: 24.3950 - val_accuracy: 0.4667
Epoch 4/30
76/76 [==============================] - 135s 2s/step - loss: 0.2739 - accuracy: 0.9051 - val_loss: 18.8119 - val_accuracy: 0.4333
Epoch 5/30
76/76 [==============================] - 134s 2s/step - loss: 0.2428 - accuracy: 0.9038 - val_loss: 2.5791 - val_accuracy: 0.5667
Epoch 6/30
76/76 [==============================] - 136s 2s/step - loss: 0.2122 - accuracy: 0.9297 - val_loss: 1.7023 - val_accuracy: 0.5667
Epoch 7/30
76/76 [==============================] - 141s 2s/step - loss: 0.2351 - accuracy: 0.9171 - val_loss: 0.4805 - val_accuracy: 0.7500
Epoch 8/30
76/76 [==============================] - 137s 2s/step - loss: 0.2274 - accuracy: 0.9197 - val_loss: 0.7335 - val_accuracy: 0.7167
Epoch 9/30
76/76 [==============================] - 137s 2s/step - loss: 0.2215 - accuracy: 0.9151 - val_loss: 0.4540 - val_accuracy: 0.8333
Epoch 10/30
76/76 [==============================] - 136s 2s/step - loss: 0.2054 - accuracy: 0.9171 - val_loss: 0.4402 - val_accuracy: 0.7833
Epoch 11/30
76/76 [==============================] - 137s 2s/step - loss: 0.1993 - accuracy: 0.9263 - val_loss: 0.3974 - val_accuracy: 0.8000
Epoch 12/30
76/76 [==============================] - 142s 2s/step - loss: 0.1925 - accuracy: 0.9210 - val_loss: 0.1917 - val_accuracy: 0.9167
Epoch 13/30
76/76 [==============================] - 144s 2s/step - loss: 0.1750 - accuracy: 0.9363 - val_loss: 0.3100 - val_accuracy: 0.8167
```

Figure 6.17: Evaluation Metrics - Training loss/acc. and Val. loss/acc

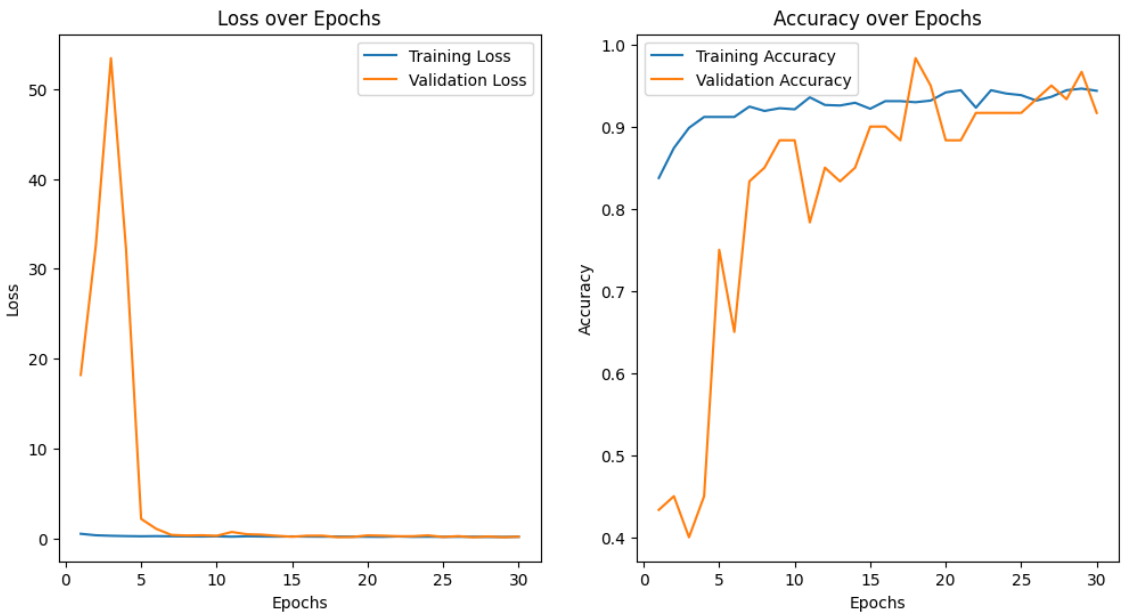


Figure 6.18: Training over 30 Epoch

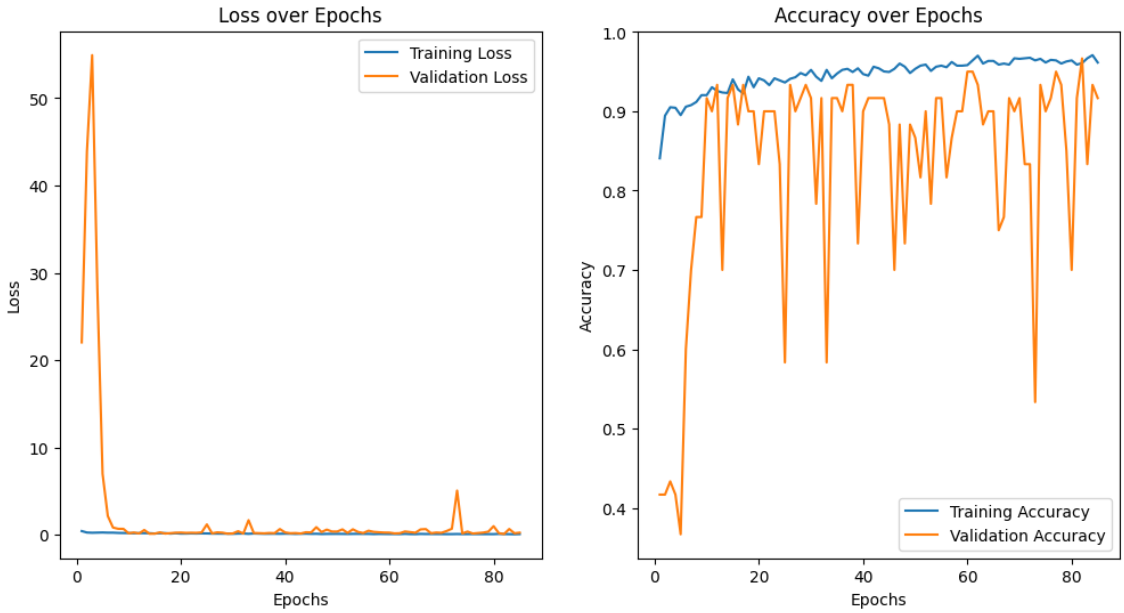


Figure 6.19: Training Over 85 epoch

The model's performance was evaluated based on training and validation accuracy and loss. The EarlyStopping callback significantly improved the model's generalization by preventing overfitting. The results are as follows:

- Training Accuracy (Combined): 0.9509
- Validation Accuracy (Combined): 0.9667
- Training Loss (Combined): 0.1357
- Validation Loss (Combined): 0.1048

```
Total params: 19037121 (72.62 MB)
Trainable params: 19035649 (72.62 MB)
Non-trainable params: 1472 (5.75 KB)
_________________________________________________________________
Found 1527 images belonging to 2 classes.
Found 71 images belonging to 2 classes.
Epoch 1/30
76/76 [==============================] - 140s 2s/step - loss: 0.4816 - accuracy: 0.8520 - val_loss: 46.0982 - val_accuracy: 0.4833
Epoch 2/30
76/76 [==============================] - 133s 2s/step - loss: 0.3130 - accuracy: 0.8852 - val_loss: 47.7400 - val_accuracy: 0.4500
Epoch 3/30
76/76 [==============================] - 135s 2s/step - loss: 0.2604 - accuracy: 0.9071 - val_loss: 24.3950 - val_accuracy: 0.4667
Epoch 4/30
76/76 [==============================] - 135s 2s/step - loss: 0.2739 - accuracy: 0.9051 - val_loss: 18.8119 - val_accuracy: 0.4333
Epoch 5/30
76/76 [==============================] - 134s 2s/step - loss: 0.2428 - accuracy: 0.9038 - val_loss: 2.5791 - val_accuracy: 0.5667
Epoch 6/30
76/76 [==============================] - 136s 2s/step - loss: 0.2122 - accuracy: 0.9297 - val_loss: 1.7023 - val_accuracy: 0.5667
Epoch 7/30
76/76 [==============================] - 141s 2s/step - loss: 0.2351 - accuracy: 0.9171 - val_loss: 0.4805 - val_accuracy: 0.7500
Epoch 8/30
76/76 [==============================] - 137s 2s/step - loss: 0.2274 - accuracy: 0.9197 - val_loss: 0.7335 - val_accuracy: 0.7167
Epoch 9/30
76/76 [==============================] - 137s 2s/step - loss: 0.2215 - accuracy: 0.9151 - val_loss: 0.4540 - val_accuracy: 0.8333
Epoch 10/30
76/76 [==============================] - 136s 2s/step - loss: 0.2054 - accuracy: 0.9171 - val_loss: 0.4402 - val_accuracy: 0.7833
Epoch 11/30
76/76 [==============================] - 137s 2s/step - loss: 0.1993 - accuracy: 0.9263 - val_loss: 0.3974 - val_accuracy: 0.8000
Epoch 12/30
76/76 [==============================] - 142s 2s/step - loss: 0.1925 - accuracy: 0.9210 - val_loss: 0.1917 - val_accuracy: 0.9167
Epoch 13/30
76/76 [==============================] - 144s 2s/step - loss: 0.1750 - accuracy: 0.9363 - val_loss: 0.3100 - val_accuracy: 0.8167
Epoch 14/30
76/76 [==============================] - 142s 2s/step - loss: 0.1771 - accuracy: 0.9310 - val_loss: 0.1513 - val_accuracy: 0.9500
Epoch 15/30
76/76 [==============================] - 144s 2s/step - loss: 0.1579 - accuracy: 0.9409 - val_loss: 0.1632 - val_accuracy: 0.9167
Epoch 16/30
76/76 [==============================] - 134s 2s/step - loss: 0.1728 - accuracy: 0.9356 - val_loss: 0.1301 - val_accuracy: 0.9500
Epoch 17/30
76/76 [==============================] - 136s 2s/step - loss: 0.1825 - accuracy: 0.9283 - val_loss: 0.4526 - val_accuracy: 0.7833
```

Figure 6.20: Evaluation Metrics - Training loss/acc. and Val. loss/acc (with callback)

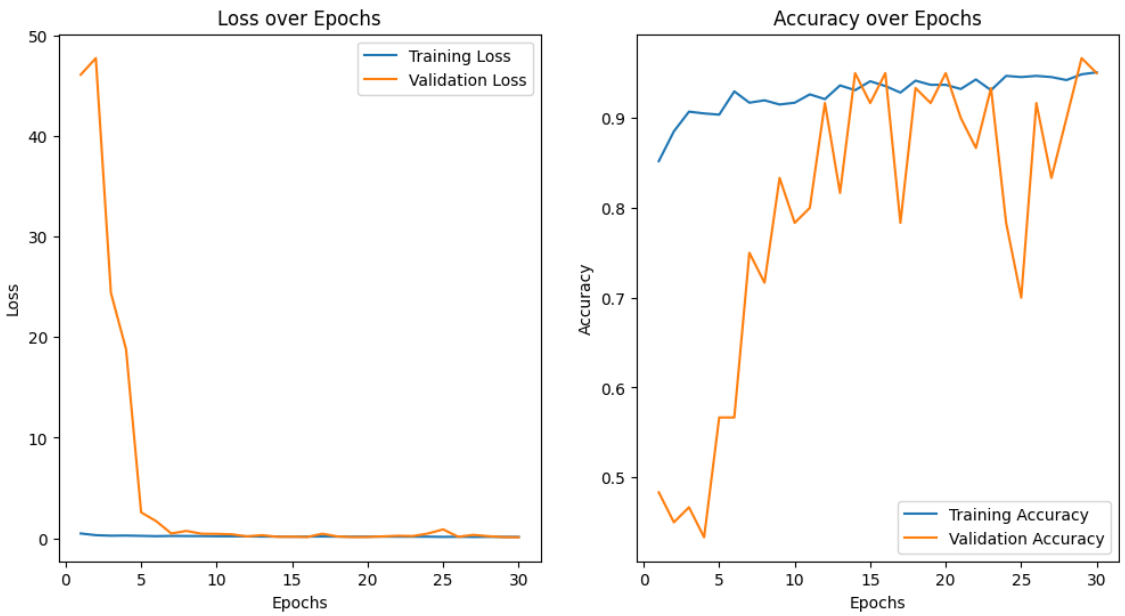


Figure 6.21: Training over 30 epoch and 15 patience

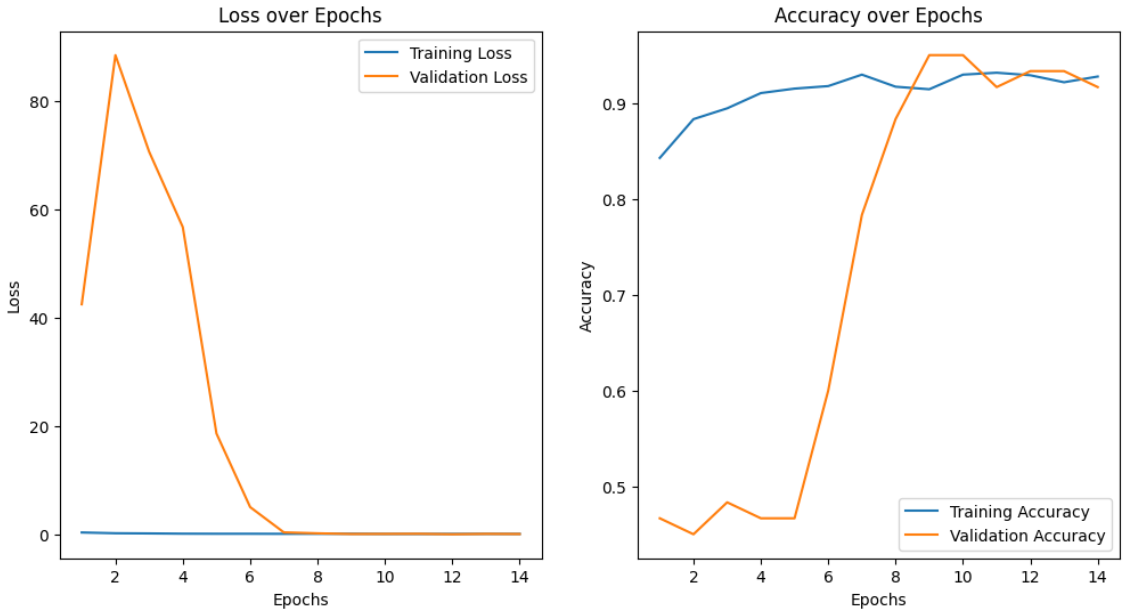


Figure 6.22: Training over 30 epoch and 5 patience

6.3.8 Conclusion

The implemented CNN model demonstrates a high level of accuracy in classifying eye-diagram images according to the specified thresholds. This tool provides a rapid and automated method for assessing the signal integrity of high-speed digital circuits, thereby aiding in the efficient design and validation of these systems. The integration of this CNN model into the design workflow has the potential to significantly streamline the evaluation process, ensuring that circuits meet the necessary performance standards. Training the model under different scenarios (single-ended, differential, and combined) allows for a comprehensive analysis of its performance across various types of signals, ensuring its robustness and versatility in real-world applications. The use of the EarlyStopping callback proved to be effective in preventing overfitting and ensuring the model generalizes well on unseen data. This approach can be extended to other image classification tasks, providing a robust method for automated diagnostics in various fields.

7 Conclusion

The results of this project highlight the effectiveness of using regression and CNN models in the context of high-speed digital circuits. The regression model provides precise predictions of transient responses, facilitating the design and optimization of circuits. Meanwhile, the CNN model offers reliable classification of eye diagrams, enabling quick and accurate assessments of signal integrity. Training the CNN model under different scenarios allowed for a thorough evaluation of its performance, ensuring it can handle a variety of signal types.

Overall, this project contributes significantly to the advancement of automated signal integrity analysis. By leveraging machine learning techniques, it provides tools that can greatly enhance the efficiency and accuracy of high-speed digital circuit design and validation. The integration of these models into the design workflow has the potential to streamline the development process, reduce time-to-market, and improve the performance and reliability of digital systems.

References

[1] Werner John, Julian Withöft, Emre Ecik, Ralf Brüning, and Jürgen Götze. A practical approach based on machine learning to support signal integrity design. In *2022 International Symposium on Electromagnetic Compatibility–EMC Europe*, pages 623–628. IEEE, 2022.

[2] Tianjian Lu, Ju Sun, Ken Wu, and Zhiping Yang. High-speed channel modeling with machine learning methods for signal integrity analysis. *IEEE Transactions on Electromagnetic Compatibility*, 60(6):1957–1964, 2018.

[3] Kallol Roy, Majid Ahadi Dolatsara, Hakki M Torun, Riccardo Trinchero, and Madhavan Swaminathan. Inverse design of transmission lines with deep learning. In *2019 IEEE 28th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 1–3. IEEE, 2019.

[4] Huan Huan Zhang, Zhao Sheng Xue, Xin Yi Liu, Ping Li, Lijun Jiang, and Guang Ming Shi. Optimization of high-speed channel for signal integrity with deep genetic algorithm. *IEEE Transactions on Electromagnetic Compatibility*, 64(4):1270–1274, 2022.