STUDY OF GENES INTERACTION RELATIONSHIPS USING REGRESSION CONVOLUTIONAL NEURAL NETWORKS WITH HYPOTHESIS TESTING ON LARGE-SCALE SELF-SIMULATED GENE PROFILES WITH EMBEDDED PHYLOGENETIC TREE STRUCTURES

by

Zesheng Jia

Co-supervised by Dr. Hong Gu and Dr. Toby Kenney

Submitted in partial fulfillment of the requirements
for the degree of Combined Honours in Statistics and Mathematics

at

Dalhousie University
Halifax, Nova Scotia
April 2024

This thesis is dedicated to my parents and my partner, whose unwavering support guided me through the most challenging times, especially when I was diagnosed with permanent vision loss. Even if in moments when I would no longer be able to see, their love and encouragement will serve as my guiding light, allowing me to navigate the world with clarity and purpose. Thank you for your unconditional love.

Table of Contents

List of Tables

List of Figures

Abstract

In this study, we evaluated multiple deep learning models, including a simple feed-forward neural network, a Convolutional Neural Network (CNN), and the ResNet with and without a bottleneck structure, for identifying gene interaction types using 5 genes profiles image that is embedded with phylogenetic tree structures. We combined these models with hypothesis testing, analyzed the ROC curve, and compared the performance of the evolCCM and CNN models. A pipeline was developed to determine the minimum dataset size for training, and model robustness was confirmed against genes' order changes in the gene profile using the same trained model checkpoint. The False Discovery Rate of the predictions was calibrated using the Benjamini-Hochberg Procedure. Our findings indicated that the ResNet with a bottleneck structure and 34/50 layers yielded the smallest Mean Square Error and Root Mean Square Error for predicting phylogenetic rates of 5 genes along with the training time trade-off. The CNN model proved to be unaffected by changing the phylogenetic tree structures across different data points during training and very little inference time during predictions. We also conduct experiments when increase the number of genes to 10 genes, 20 genes, and 40 genes. Future research will scale up training data with more gene profiles. We hope this study can contribute to understanding interaction types between genes pairs using deep learning models, large-scale data simulation, and statistical analysis.

Acknowledgements

Special thanks to my supervisors, Professor Gu, for her guidance and support throughout this research, and Professor Kenney for his advice and encouragement.

Computations were performed on the Niagara and Mist supercomputer at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario Research Fund: Research Excellence; and the University of Toronto.

Chapter 1

Introduction

A trait is a well-defined, measurable characteristic of organisms, typically assessed at the individual level and used for comparative analysis across different species [Dawson et al., 2021]. Interactions between multiple genes significantly influence these traits. Previous studies [Liu et al., 2022], show that Genes may display similar trends of presence and absence across a collection of genomes due to factors such as involvement in a shared biochemical pathway, physical linkage, or colocalization on a mobile genetic element like a plasmid ([Bowers et al., 2004]; [Fraser et al., 2004]). Studying these patterns can provide valuable insights into related functions. A widely used method to depict presence and absence patterns among genes involves creating phylogenetic profiles. These are binary vectors that encapsulate the presence and absence of genes across a set of genomes, effectively considering each gene as an individual trait ([Pellegrini, 2012]; [Niu et al., 2017]).

In previous studies [Liu et al., 2022], Professor Gu and her team developed a new model called CCM to study the evolutionary relationships between genes to reveal organismal traits' characteristics. CCM model uses the maximum likelihood method with a dedicated phylogenetic tree structure for each gene group to estimate the interaction rates between gene pairs. They demonstrated that their model is more efficient and fits real data better than other methods, resulting in higher likelihood scores with fewer parameters. However, the previous model suffers from the computational time of larger tree structure size and the time and resources required for computing the eigendecomposition of the Q matrix of $2^n \times 2^n$, especially when the number of genes 'n' is larger than 10.

To overcome the difficulties in the CCM model, we think that deep learning models can be a good alternative to studying the evolutionary relationships between genes. Deep learning models have been widely used in various fields, such as computer vision, natural language processing, and supervised learning tasks.

[Gorishniy et al., 2023] are testing multiple deep learning models for tabular data. [Kumar et al., 2023] are using deep learning models for computational biology for DNAs. However, deep learning models often require a large amount of data for training. Hence, we developed procedure to apply data simulation method to generate large-scale data for training the deep learning models in Chapter 2. In Chapter 3, we propose a new model to study the evolutionary relationships between genes using Deep Regression Convolutional Neural Networks (including a vanilla CNN and ResNet [He et al., 2015]) on Gene's Phylogenetic Profiles and we surveyed the simple neural network for comparison. In Chapter 4, we did multiple experiments on Hypothesis Testing, and model comparison between the CNN model and the CCM model. We did various experiments on the model's input data and output results for a better understanding of the model's performance and the potential to improve our predictions. In Chapter 5, we tried to scale up our model to more challenging tasks. We increased the number of tree tips/leaves and the number of genes to 10 genes, 20 genes, and 40 genes. Eventually, We show that the CNN model is immune to the change of the phylogenetic tree structure for different groups of genes that have different tree structures. We also show that the CNN model is more efficient when we make inferences on the evolutionary relationships between genes with a good performance. We will break down those questions and tasks in the following chapters and include their own backgrounds at the beginning of each section to provide a better understanding of the study.

Chapter 2

Data Simulation

2.1 Data Simulation Background: The CCM Model

For deep learning models, the size of the training data and validation data is crucial for the model's performance. In this study, we used the data simulation method from the previous study of the CCM model [Liu et al., 2022] to generate large-scale data of the genes' profiles and rates for different numbers of genes and tree leaves by settings. In the CCM model, Professor Gu and her team evaluate the potential associations between two or more genes within a given phylogenetic tree. Specifically, they examine whether the transition (gain or loss) of any gene within a group is influenced by the current states of its other members. Genes can have positive associations if the genes tend to be gained and lost concurrently, and negative associations if the gain of certain genes in a set seems to correspond with the loss of others within the same set. they refer to gene sets that exhibit signs of associations as a "community" in the CCM model.

[Liu et al., 2022] formulate the transition rate τ for one gene as a function of its intrinsic rate of gain and loss μ. The association factor ω depends on the current states of all other genes in the community.

$$\tau = \mu \times \omega$$

Here, we define the following notations for further usage:

- n: The total number of genes
- m_{tips}: The total number of phylogenetic tree leaves or tips
- $S = \{S_1, S_2, ..., S_{2^n}\}$: The state space of the groups of n genes' representation
- $S_i = \{x_{i,k}; k = 1, 2, ..., n\}$: A vector of one group of n genes. $x_{i,k}$ is the state of

the k-th gene in the i-th group. $x_{i,k} = -1$ means the gene is not present in the group, and $x_{i,k} = 1$ means the gene is present in the group.

- $\mathcal{B} = \{\beta_{h,k}; k, h = 1, 2, ..., n\}$: A symetrix $n \times n$ matrix of the interaction between the genes. $\beta_{h,k}$ is the interaction between the h-th gene and the k-th gene. Its off-diagonal entries are the coefficients of interaction between every pair of genes and diagonal entries are the half the difference between the gain and loss rates of each gene. We will mainly focus on predicting the diagonal entries of the $\mathcal{B}$ matrix.

- $\mathcal{A} = \{\alpha_1, \alpha_2, ..., \alpha_n\}$: A vector of the intrinsic rates of the groups of n genes. α_i is the rate of the i-th group of n genes. The value is the mean of gain and loss rates for each gene.

The instantaneous transition rate for one gene k in the state of representation S_i is defined in the log scale as:

$$\log(\tau_{i,k}) = \alpha_k - \beta_{kk} \cdot x_{i,k} - \sum_{h \neq k}^{n} \beta_{hk} \cdot x_{i,k} \cdot x_{i,h} \tag{2.1}$$

By our definition, a positive β_{hk} means the h-th gene is positively associated with the gain of the k-th gene. That is, $(x_{i,k} = x_{i,h} = 1)$; h-th genes and k-th genes are both present in the group. Hence, the last term in equation (2.1), $\{\sum_{h \neq k}^{n} \beta_{hk} \cdot x_{i,k} \cdot x_{i,h}\}$ will reduce the rates of change $\tau_{i,k}$ for gene k. For the negative β_{hk}, it will increase the rates of change.

Now, we take the exponential of both sides of equation (2.1) to get the transition rate $\tau_{i,k}$ for gene k in the state of representation S_i:

$$\tau_{i,k} = \underbrace{\exp\left(\alpha_k - \beta_{kk} \cdot x_{i,k}\right)}_{\mu} \cdot \underbrace{\exp\left(- \sum_{h \neq k}^{n} \beta_{hk} \cdot x_{i,k} \cdot x_{i,h}\right)}_{\omega} \tag{2.2}$$

As we defined before, the μ is the intrinsic rate of gain and loss for the k-th gene, and the ω is the association factor that depends on the current states of all other genes in the community. The ω represents the influence of the community.

[Liu et al., 2022] model the gene state changes as a continuous-time Markov process with a specific phylogenetic tree, and assume the instantaneous rate for all transitions involving more than one gene is 0. The transition rate matrix $\mathcal{Q}$ is defined as:

$$\mathcal{Q} = \{q_{i,j} : i, j = 1, 2, \ldots, 2^n\}$$
$$q_{i,j} = \begin{cases} \tau_{i,k} & \text{if } i \neq j \textbf{ and } S_i - S_j = \pm 2e_k \\ -\sum_{i' \neq i} q_{i,i'} & \text{if } i = j \\ 0 & \text{Otherwise} \end{cases} \tag{2.3}$$

where $\{\ e_k; k-1, 2, ..., n\}$ is the standard basis vectors of all 0's except the kth element as 1. That is, $S_i - S_j = \pm 2e_k$ indicates that only the kth gene changes states.

To compare the CCM model with the deep learning models, we also briefly introduce the CCM model's maximum likelihood estimation method. [Liu et al., 2022] assume that transitions on separate branches of the phylogenetic tree are independent. It means that the distribution of the state at the end of a given branch depends only on the starting state of that branch. However, the computation cost could be expensive when summing all the possible combinations of the states at each internal node in the phylogenetic tree. They use Felsenstein's pruning algorithm [Felsenstein, 1973], which is a dynamic programming method to calculate the likelihood recursively. We define the modified likelihood function L in Figure 2.1(a) for demonstration as:

$$\begin{aligned} L(\Theta; T, X) = &\sum_{s_0} \Bigg[\left(\sum_{s_1} P_{s_0,s_1} \cdot (b_1) \cdot P_{s_1,s_3}(b_3) P_{s_1,s_4}(b_4) \right) \\ &\times \left(\sum_{s_2} P_{s_0,s_2} \cdot (b_2) \cdot P_{s_2,s_5}(b_5) P_{s_2,s_6}(b_6) \right) \Bigg] \end{aligned} \tag{2.4}$$

It reduces the computational complexity to linear in the number of leaves of the phylogenetic tree. For the final estimation, they use the negative log-likelihood function $-L(\Theta; T, X)$ that is minimized to get the maximum likelihood estimates of the parameters by using the quasi-Newton optimizer [Paradis et al., 2004].

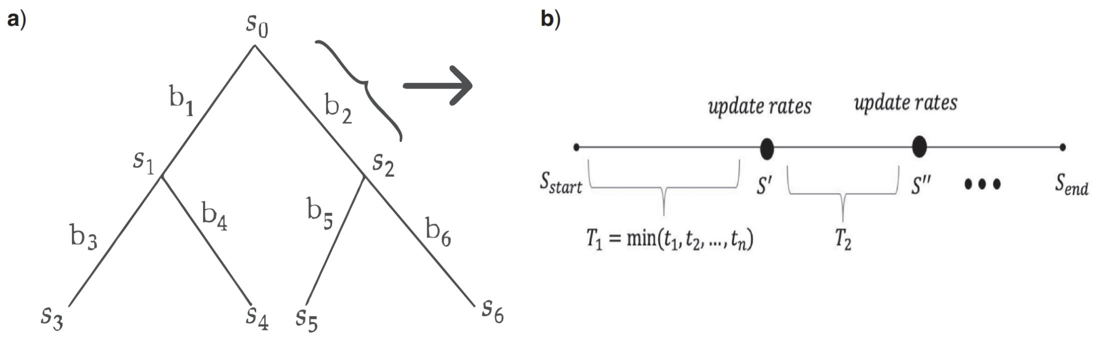


Figure 2.1: [Liu et al., 2022]'s CCM model phylogenetic tree structure and illustration of the data simulation process. *(a)* A phylogenetic tree structure with 4 tips: $\{s_i; i = 1, 2, ..., 6\}$ denotes the state at each node and $\{b_i; i = 1, 2, ..., 6\}$ denotes the branch length. *(b)* The data simulation process of the CCM model. S is the community state and T is the time that there is a transition out of the current state. The process ends when the total transition time is longer than the branch length.

In order to perform the Hypothesis test, the CCM model has two ways to calculate the standard error of estimates. The parametric bootstrap method and the Hessian matrix method.

- Parametric bootstrap: This approach simulates a substantial number of profiles using a fixed range of estimated parameters. Such as let $\mathcal{B} \in \{-1.5, 1.5\}$. Then estimate the rates based on the simulated profiles. Then computes the standard deviation of these estimates based on the bootstrap samples.

- Hessian matrix: This method calculates the standard error of the estimates by the inverse of the Hessian matrix of the negative log-likelihood function at the maximum likelihood estimates. Such that, $SE = \sqrt{diag(H^{-1})}$.

When the tree and community size increase, the MLE procedure is potentially overshooting. The CCM also adds $L2$ regularization to the likelihood function to prevent overfitting. Hence, the final version of the CCM model's likelihood function is:

$$-\log\left(\mathcal{L}(\Theta; T, X)\right) + \lambda \cdot \left(||\Theta||_2^2\right) \tag{2.5}$$

2.2 Data Simulation Procedure

In order to model the interaction relationships among genes, we utilize the CCM framework. This framework posits that the transition rates of a particular gene are influenced by the states of other genes within the same community. The process for simulating the evolution of a gene community of size 'n' or the number of genes on a single branch can be outlined as follows in Algorithm 1:

Algorithm 1: Community State Simulation

```
Input: The starting state of the community $S$, a user-defined coefficient
       matrix $\mathcal{B}_{n\times n}$, user-defined intrinsic rates $A_0$ and branch length $b$
Output: The new state of the community $S'$
1  while $b > 0$ do
2      for $h = 1$ to $n$ do
3          Calculate the current transition rate for gene $\tau_h$ using Formula (2.2);
4          Sample the transition time for gene $\tau_h$ from the exponential
             distribution, $\tau_h \sim Exp(\tau_h)$;
5      end
6      Find the gene $k$ with the minimum transition time,
         $T_{\min} = \min\{t_1, t_2, ..., t_n\}$;
7      if $T_{min} \leq b$ then
8          Update the state of gene $k$ in $S$ with the opposite state;
9      end
10     Update the branch length $b = b - T_{\min}$;
11 end
```

Figure 2.1(b) provides a visual representation of the evolutionary process on a single branch in [Liu et al., 2022]. The final state, denoted as S_{end}, then serves as the initial state for the subsequent adjacent branches. This same procedure is sequentially applied to all branches, starting from the root and extending to the tips. An example of this simulation, involving six genes divided into two groups of size 3, is depicted in Figure 2.2. The interaction matrix used in this example has within-group interaction coefficients set to 1.5, signifying strong relationships, and between-group interaction coefficients set to 0, indicating independent evolution.

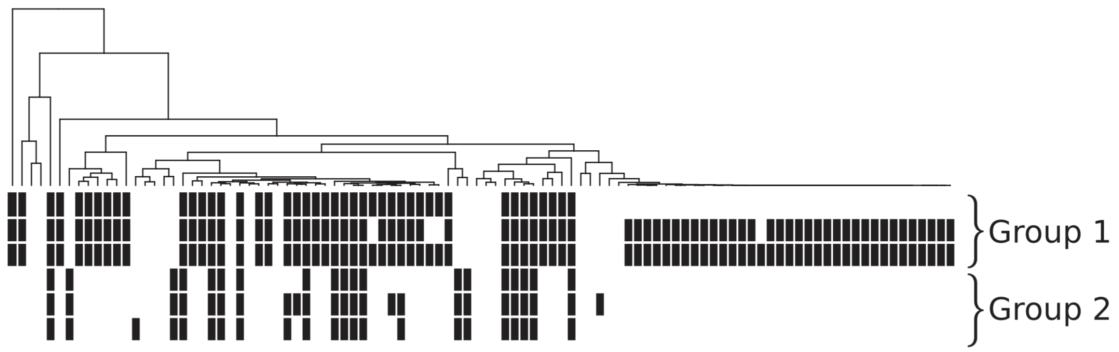


Figure 2.2: Demonstration in [Liu et al., 2022] of 2 groups of correlated profiles of size 3 by the simulation procedure. The Black strip shows the gene is present in the group and the white strip shows the gene is not present in the group. The interaction matrix is set to 1.5 for within-group interactions and 0 for between-group interactions.

By using Algorithm 1, we have the ability to simulate the evolution of gene communities of varying sizes and interaction strengths. This simulation process is crucial for generating the training and validation datasets for the deep learning models. The generated datasets will be used to train the models to predict the phylogenetic rates of the genes and to identify the interaction types between genes.

Here, we provide a high-level overview of the simulation workflow in Algorithm 2. The simulation workflow generates a random tree of size t_s and rescales the branch lengths. Then we set the parameters for the simulation and create folders for storing the simulation data. For each simulation, the workflow runs Algorithm 1's simulation procedure and saves the data. Based on the data simulation method from the CCM model, we developed a robust code base in R that can generate up to 8 million genes' profiles and rates for different numbers of genes and tree leaves by settings. The random seed is for the reproducibility of the simulation data. And especially when we use clusters, we can generate multiple data simulation tasks with different

Algorithm 2: Simulation Workflow

```
Input: random seeds, tree size/number of tips $t_s$, number of genes, number
       of data points
Result: simulated data saved in specified folders
1 Set random seed;
2 Generate a random tree structure by [Paradis et al., 2004] of size $t_s$ and
    rescale branch lengths;
3 Set number of genes;
4 Set parameters for simulation: $\mathcal{B}_{n\times n}$, $\mathcal{A}_{1\times n}$ ;
5 Create folders for simulation data;
6 foreach Number of data points do
7     Run Algorithm 1's simulation procedure and save data;
8 end
```

random seeds. Then combine the data afterwards. The R code is available in the Github repository.

2.3 Data Simulation Results

For the parameters and Genes Rates, we have $\mathcal{A}_5, \mathcal{B}_{5\times 5}$, and the number of tips of the tree structure. As shown in Figure 2.4, we sample $\{\mathcal{A}_i; i = 1, 2, ..., 5\}$ and $\{\mathcal{B}_{i,i}; i = 1, 2, ..., 5\}$ from a uniform distribution from $(-0.5, 0.5)$. And sample $\{\mathcal{B}_{i,j}; i \neq j$ **and** $i, j = 1, 2, ..., 5\}$ from a Gaussian distribution with $\mu = -0.5$ and $\sigma = 1.5$. We set the number of tips of the tree structure to 400. We simulate 4 Million data points of 5 genes' profiles and rates. The data simulation process is completed in around 10 hours in the Niagara cluster. We demonstrate the outcome profiles from the data simulation procedure of 5 genes in Figure 2.3. The black strip indicates that the gene is present in the group, while the white area indicates that the gene is not present in the group.

The Niagara cluster, referenced in [Ponce et al., 2019], is a supercomputer system that forms part of the Compute Canada network. This substantial cluster comprises

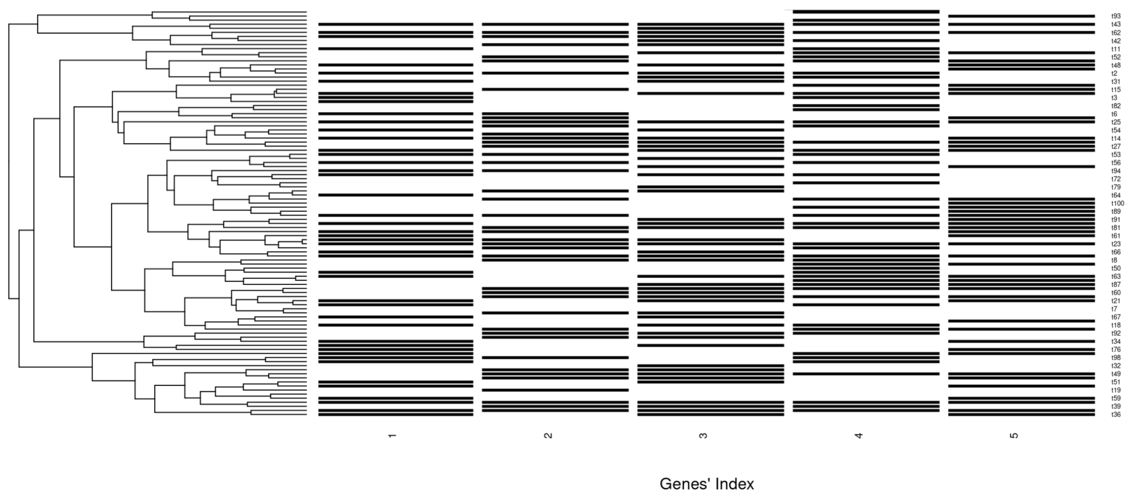


Figure 2.3: Demonstration of 5 Genes' simulated Profiles. The Black strip shows the gene is present in the group and the white area shows the gene is not present in the group.

2,024 Lenovo SD530 servers, each equipped with either 40 Intel "Skylake" cores running at 2.4 GHz (1548 nodes) or 40 Intel "CascadeLake" cores operating at 2.5 GHz (476 nodes). The Niagara cluster is housed at the University of Toronto and is utilized for high-performance computing tasks. Niagara cluster belongs to SciNet datacenter [Loken et al., 2010]. In our study, we utilized one Node from the Niagara cluster to generate the data simulation. One node contains around 200GB of memory and 80 cores of CPUs. We employ the "parallel" package in R, a powerful tool for parallel computing, to concurrently distribute 80 processes. This approach significantly reduces computation time. As a built-in feature referenced in [R Core Team, 2024], the "parallel" package facilitates the efficient distribution of processes across multiple cores of a CPU.

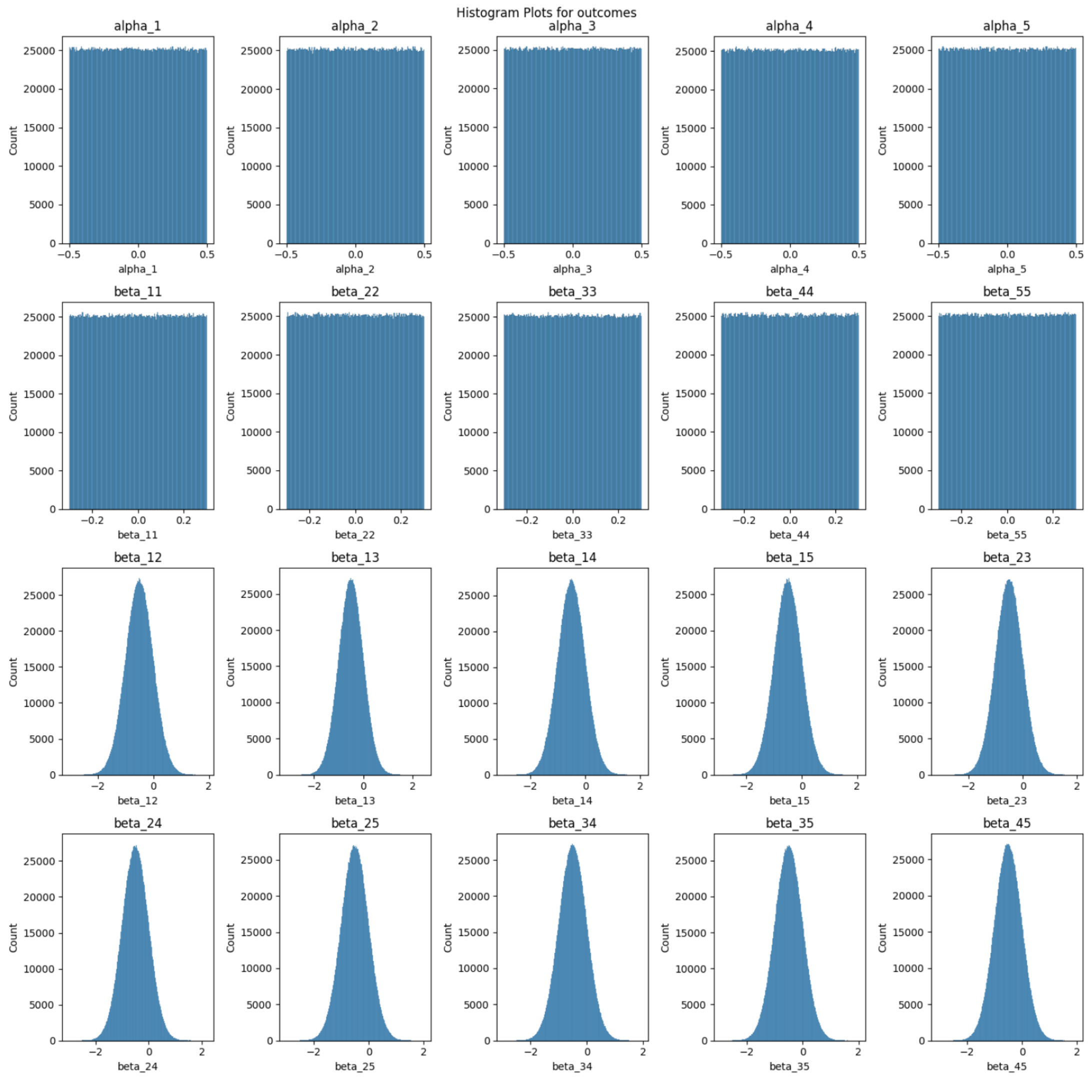


Figure 2.4: Histogram of 5 Genes' Rates. $\{\mathcal{A}_i; i = 1, 2, ..., 5\}$, $\{\mathcal{B}_{i,i}; i = 1, 2, ..., 5\}$ from a uniform distribution from $(-0.5, 0.5)$, and $\{\mathcal{B}_{i,j}; i \neq j$ **and** $i, j = 1, 2, ..., 5\}$ from a Gaussian distribution with $\mu = -0.5$ and $\sigma = 1.5$.

Chapter 3

Survey on Deep Learning Models with Genes's interaction Relationships

3.1 Deep Learning Models Overview

In our data, we need to use the genes' profile to predict multiple outcomes, such as $\{\mathcal{A}_i; i = 1, 2, ..., 5\}$, $\{\mathcal{B}_{i,i}; i = 1, 2, ..., 5\}$, and $\{\mathcal{B}_{i,j}; i \neq j \textbf{ and } i, j = 1, 2, ..., 5\}$ in the same time. Deep learning models are naturally suitable for such tasks. We can adjust the number of the last layers' nodes to fit the requirement of the number of outcomes altogether. There have been multiple attempts in the past from different researchers that use deep learning models to predict supervised learning and tabular tasks. [Borisov et al., 2024] examines multiple deep learning structures on five popular real-world tabular data sets of different sizes and with different learning objectives. [Kumar et al., 2023] shows that deep Convolutional Neural Network is highly accurate in predicting genetic variations and gene expression levels. They use Deep learning techniques for analyzing epigenetic data, including DNA methylation and histone modifications. [Gorishniy et al., 2023] examines two popular deep learning structures; ResNet and Transformers on tabular data, and compares them with Gradient Boosted Decision Trees.

In the recent trend of large-language models, computer vision models, and supervised learning tasks, we found that deep learning models are sensitive to data size and model complexity. Recent practices in deep learning have consistently demonstrated that enhanced performance is achieved through larger model sizes, increased data, and additional computation, which collectively contribute to a reduction in training loss. [Simon et al., 2023] gives theoretical backing to these empirical observations. In our study, we also test and verify that the deep learning models' performance on the genes' profile data is heavily impacted by the training data size with certain thresholds. However, our study fully utilizes the data simulation methods to generate large-scale data for training. Hence, we have the ability to scale up our models' complexity by training with very large-scale data.

In the following sections of this chapter, we will introduce the deep learning models we tested in our study. We will also discuss the model structures, the training process, and the results of the models. We started from the most common and simple deep learning structure, the Feedforward Neural Network, and then moved to the more complex models, such as the Convolutional Neural Network [O'Shea and Nash, 2015] and the ResNet [He et al., 2015].

3.2 Simple Feedforward Neural Network

3.2.1 Model Structure and Training Process

In Figure 3.1, we show a simple structure of the FeedForward Neural Network in our study. To construct Figure 3.1 with a limited number of nodes and edges, we reduce the gene profile data to include only 4 tree tips. This implies that each gene possesses 4 states. With a total of 5 genes, the gene profile becomes a matrix with 5 rows and 4 columns. In the context of a Feedforward Neural Network (FFN), we flatten this matrix into a single linear vector, resulting in $5 \times 4 = 20$ nodes.

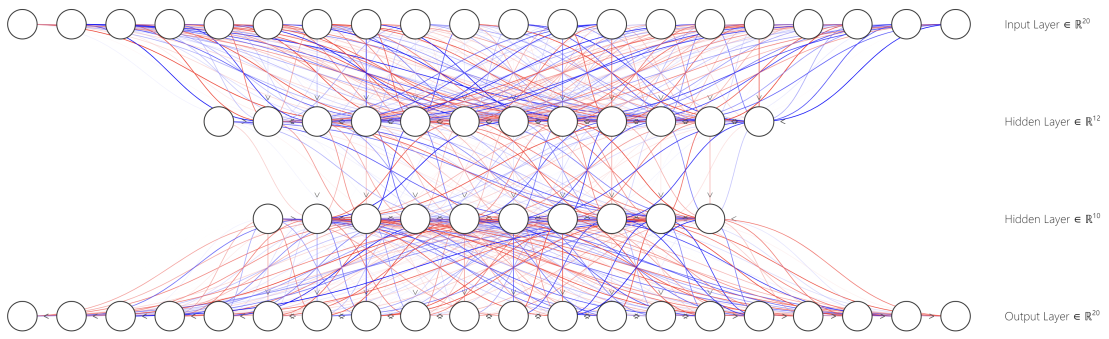

Input Layer ∈ $\mathbb{R}^{20}$
Hidden Layer ∈ $\mathbb{R}^{12}$
Hidden Layer ∈ $\mathbb{R}^{10}$
Output Layer ∈ $\mathbb{R}^{20}$

Figure 3.1: Demonstration of a Simple Feedforward Neural Network with genes' profile settings: tree tips/leaves size = 4, number of genes = 5. The Genes' Rates are the output of the model with 20 parameters. $\{A_i; i = 1, 2, ..., 5\}, \{B_{i,i}; i = 1, 2, ..., 5\}, \{B_{i,j}; i \neq j; (i, j) = 1, 2, ..., 5\}$

.

Table 3.1: FeedForward Neural Network Structure

Layer (type)	Input Nodes	Output Nodes	Param + Bias Nodes#
Layer 1	2000	128	256128
Layer 2	128	64	8256
Layer 3	128	20	1300

For the output nodes, we have 5 nodes to predict the $\{A_i; i = 1, 2, ..., 5\}$, 5 nodes to predict the $\{B_{i,i}; i = 1, 2, ..., 5\}$, and (5 choose 2) = 10 nodes to predict the $\{B_{i,j}; i \neq j; (i, j) = 1, 2, ..., 5\}$. (5 is the number of genes, and 2 is the interaction between $\{B_{(i,j)}; i \neq j\}$). Hence, the total number of outcomes in the model is 20 nodes. For the internal layers, we can choose an arbitrary number of nodes and layers. In the demonstration, we pick the first hidden layer with 12 nodes and the second hidden layer with 10 nodes. For the model training, We use the Mean Square Error MSE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ as the loss function to train the model. The model is trained with the Adam optimizer [Kingma and Ba, 2017] with a learning rate of 0.001 with ReLU as its activation function [Agarap, 2019]. The model is implemented in Python with PyTorch [Paszke et al., 2019].

For the real training data, each gene's profile has 400 tree tips and 5 genes. The flattened vector size is 500 nodes. We added 2 hidden layers between the input and output layers with nodes 128 nodes and 64 nodes. For each layer, we add batch normalization [Ioffe and Szegedy, 2015]. Batch normalization enables the use of significantly higher learning rates and reduces the need for specially designed initialization of network weights. The model is trained with 4 Million data points on the Niagara and Mist cluster with one Nvidia V100 with 32G GPU memory.

3.2.2 Model Prediction and Results

In Figure 3.2, we show the scatter plot of the predictions of the Feedforward Neural Network model. The x-axis is the true value of the genes' rates, the y-axis is the predicted value of the genes' rates, and "$y = x$" by the red line. We notice that the model hardly can predict the $\{A_i; i = 1, 2, ..., 5\ \}$ and $\{B_{i,i}; i = 1, 2, ..., 5\ \}$. However, in our study, we are more interested in the interaction between genes:

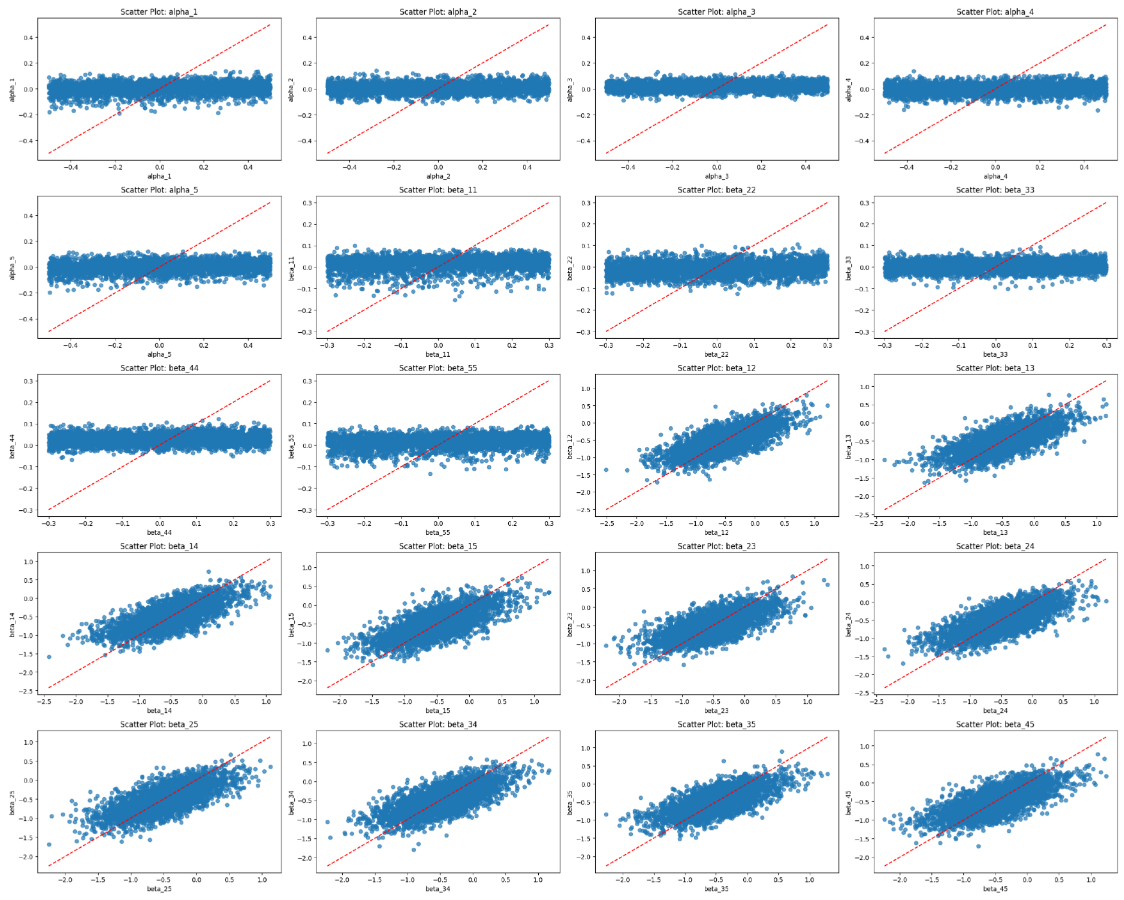


Figure 3.2: Predictions of a Simple Feedforward Neural Network with genes' profile settings: tree tips/leaves size = 400, number of genes = 5. The Genes' Rates are the output of the model with 20 parameters. $\{A_i; i = 1, 2, ..., 5\}, \{B_{i,i}; i = 1, 2, ..., 5\}, \{B_{i,j}; i \neq j; (i, j) = 1, 2, ..., 5\}$

.

$\{B_{i,j}; i \neq j; (i,j) = 1,2,...,5\ \}$, instead of the intrinsic rates $\mathcal{A}_i$ of the genes, and the interaction rates $\mathcal{B}_{(i,i)}$ between the same gene itself. In Table 3.1, we show the detailed Mean Square Error for genes' Rates by the FFN's prediction, and Table 3.2 shows the Root Mean Square Error. We will use those two tables to compare the performance of the other models in the following sections.

Table 3.2: FeedForward Neural Network MSE on 5 Genes' Rates with 400 tree tips

alpha_1	alpha_2	alpha_3	alpha_4	alpha_5
0.083746	0.081373	0.081138	0.082228	0.080709
beta_11	beta_22	beta_33	beta_44	beta_55
0.029672	0.02945	0.03015	0.028924	0.028396
beta_12	beta_13	beta_14	beta_15	beta_23
0.124953	0.11659	0.118786	0.12303	0.127255
beta_24	beta_25	beta_34	beta_35	beta_45
0.126977	0.118641	0.118761	0.121724	0.126617

Table 3.3: FeedForward Neural Network RMSE on 5 Genes' Rates with 400 tree tips

alpha_1	alpha_2	alpha_3	alpha_4	alpha_5
0.289392	0.285259	0.284846	0.286744	0.284097
beta_11	beta_22	beta_33	beta_44	beta_55
0.172249	0.171612	0.173643	0.170070	0.168510
beta_12	beta_13	beta_14	beta_15	beta_23
0.353489	0.341456	0.344653	0.350757	0.356902
beta_24	beta_25	beta_34	beta_35	beta_45
0.356331	0.344448	0.344615	0.348890	0.355831

In Figure 3.2, especially the scatter plot of $\{B_{(i,j)}; (i \neq j)\}$, we still can notice a wide band in the middle of the strip. It shows the variance of the prediction is still significant. The points on the right end of the scatter plot of truth vs prediction are mostly below the line "y = x", which means that the model tends to under-predict the target variable for higher values. In other words, for near the largest true values, the predictions are generally lower than the actual values. In the meantime, at the left end of the scatter plot, for near the smallest true values, the predictions are generally lower than the actual values. This could indicate that the FFN model is not capturing some aspect of the relationship between the genes' profiles and genes' rates for two-tailed values. Because of sampling method of $\{B_{(i,j)}; (i \neq j)\}$ is from a Gaussian distribution with $\mu = -0.5$ and $\sigma = 1.5$, the model may not be able to capture the wide range of the distribution, since the values at two-tailed data are rarely sampled. This phenomenon is observable across all $\{B_{(i,j)}; (i \neq j)\}$ in Figure 3.2. Figure 3.3 shows the training loss and validation loss during 1000 epochs, we can see that the model is not overfitting, and the loss turns to stable.

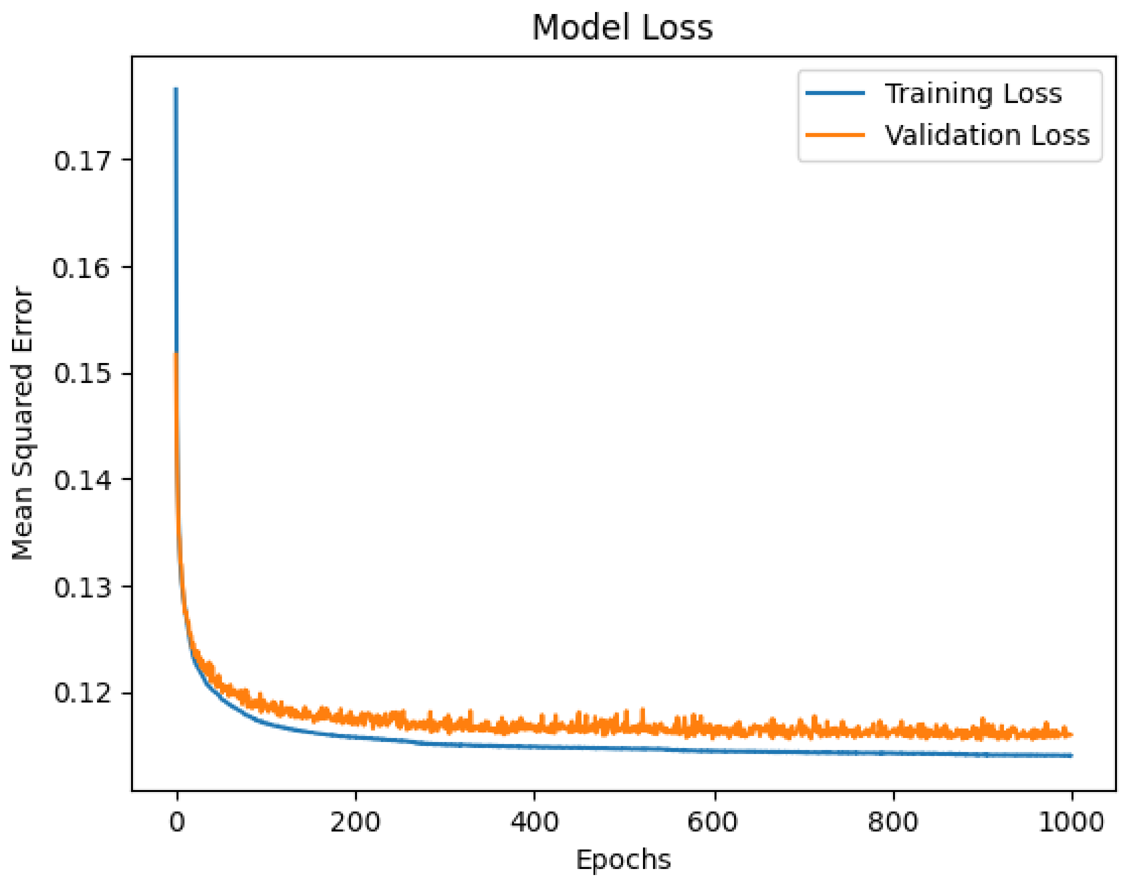


Figure 3.3: Training Loss and Validation Loss of simple Feedforward Neural Network during 1000 epochs.

3.2.3 Advantages and Disadvantages of the Feedforward Neural Network

The Feedforward Neural Network is known for its simplest structure and dynamic design of output nodes. In our case, if the number of tree tips or the number of genes changes, we can easily modify the number of input and output nodes. However, for the simple FFN, the hidden layers design will influence the model performance if the design is far from optimal. The cost of finding such a structure may become a burden if the property of genes' profiles is frequently changed. This procedure was quite common a few years back before the residual block was introduced in [He et al., 2015], which makes the Neural Network can go much deeper and the network structure design become more stable.

3.3 Convolutional Neural Network

3.3.1 Background and Image Proprocessing

In our study, we represent gene profiles as matrices, where the presence or absence of genes within a group is indicated. This representation incorporates both the structures of phylogenetic trees and the interactions between genes. Given its renowned ability to capture the spatial structure of images, the Convolutional Neural Network (CNN) is an ideal choice for processing such data. In this context, we treat the gene representation matrix as the input image for the CNN model. For instance, a black strip in the image signifies the presence of a gene in the group, while a white area denotes its absence. We use 0 and 1 to represent the absence and presence of genes in the group, respectively.

In the meantime, in a standard Convolutional Neural Network (CNN), the size of the input image plays a significant role. Typically, the filter size is either 3×3 or 5×5. However, if the image is too narrow or shallow, the filter may not function as expected within the CNN model, particularly when the number of genes is minimal. For instance, if a gene profile consists of 5 genes and 400 tips, the initial image size would be 400×5. In contrast, the typical image size for a CNN is around

128×128, 512×512, or even larger. Utilizing such a narrow image width could present challenges for CNNs, as they might struggle to effectively learn complex patterns from such constrained spatial information. Here are some potential challenges we might encounter when using a small image size in the CNN model:

- Spatial Dimensions: The height of 5 pixels might be too small for the convolutional layers to effectively capture meaningful patterns. Convolutional layers are designed to detect spatial patterns in images, and with a height of only 5 pixels, there may not be enough spatial information for the model to learn from.
- Pooling Operations: For the max pooling procedure, the small height of 5 pixels could lead to significant information loss during pooling operations, reducing the effectiveness of the model.
- Model Architecture: The architecture of the CNN model might not be suitable for processing such small images. Designing and changing the architecture may become a difficult task.
- Overfitting: With such a small height, the model may be prone to overfitting, especially if the number of parameters in the model is large relative to the amount of data.

To address these challenges, we propose an image preprocessing approach to increase the width of the gene profile image (when the number of genes is small). By duplicating the gene profile matrix, we can expand the width of the image to a more suitable size for the CNN model. For example, if the original gene profile matrix is 400×5, we can duplicate the matrix and append them to the right-hand side to create an image with a height of 400×200, as shown in Figure 3.4. This approach allows the CNN model to effectively learn spatial patterns from the gene profile data. In Figure 3.5, we demonstrate the CNN model structure with an input gene size of 400×200 and an output gene rate size with 10 nodes, as the interaction rates between genes 1 to 5: $\{B_{i,j}, i \neq j; (i, j) = 1, 2, ..., 5\ \}$. For the specific settings of image width

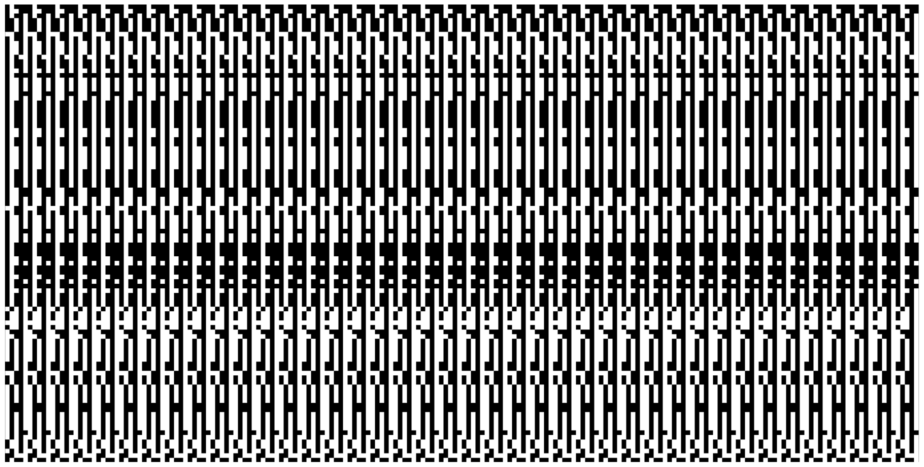


Figure 3.4: Demonstration of Genes' representation with duplicates in the gene profile to construct the image input.

after duplicate, we test multiple width lengths from 50 to 200. We found that for different tasks, the length of width will affect the training speed slightly, but not by much on the prediction performance. In the following sections, we will mainly focus on the width length of 200 or 100 for the CNN model.

3.3.2 Model Structure and Training Process

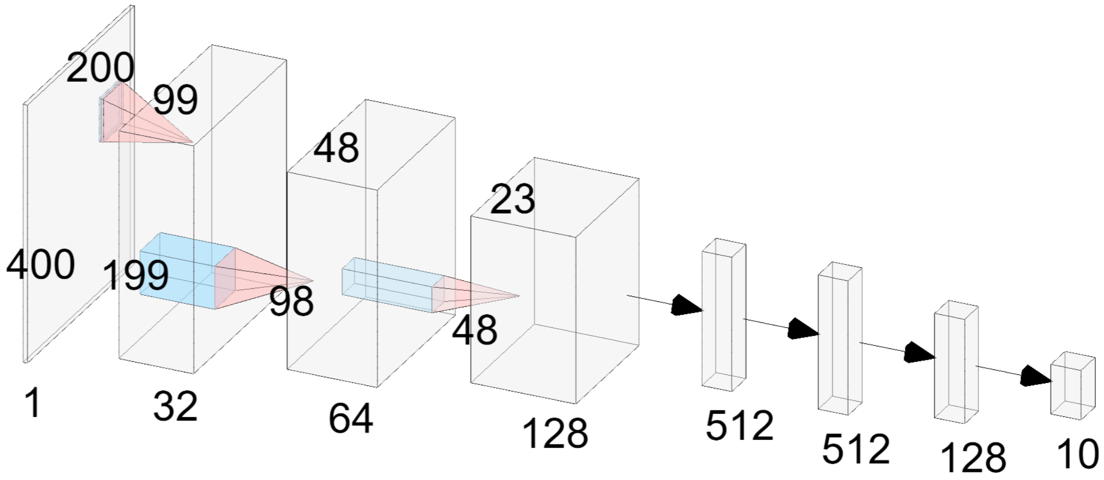


Figure 3.5: Demonstration of CNN model structure with input gene size 400×200 and output genes' rates size with 10 nodes.

In our study, the output of the CNN model should be numerical values of interaction rates between different genes. Hence, the task that we are trying to solve is a regression task that requires numerical outputs. In our case, for the 5 genes data, as shown in Figure 3.5, the original input is a 3D matrix with a shape “$400 \times 200 \times 1$”. 400 is the number of tree tips, 200 is the width of the image after duplicate, and 1 is the channel of the image (represents that the color is only black and white without RGB channel). We add 3 convolutional layers with a kernel size of 3×3 and a stride of 1. The first convolutional layer has 32 filters, the second convolutional layer has 64 filters, and the third convolutional layer has 128 filters. We add a max-pooling layer with a kernel size of 2×2 and a stride of 2 after each convolutional layer.

After the CNN layers, we flatten the nodes to one extra hidden linear layer that contains all the nodes from the CNN model (Due to the size of this layer, we didn’t show it in Figure 3.5), and we connect this hidden layer to a linear layer with 512 nodes. Then, we add 2 fully connected layers with 512 nodes and 128 nodes, respectively. The output layer has 10 nodes to predict the interaction rates between different genes. We use the Mean Square Error as the loss function to train the model. The model is trained with the Adam optimizer with a learning rate of $1e^{-3}$ with ReLU as its activation function. The regression task is solved by using the MSE loss function and the last output layer with no activation function.

In the data preprocessing phase of our study, we initially generated gene profile data comprising 400 tree tips and 5 genes. To increase the width of the image to 200, we duplicate the gene profile matrix. As a standard procedure, we normalize the data to a range of [0, 1] by dividing it by the maximum value in the gene profile matrix (we will incorporate more gene conditions in subsequent chapters). We partition the data into training and validation sets at an 80:20 ratio. The data is then converted into PyTorch tensors and reshaped into a 4D tensor with a shape of “$n \times 400 \times 200 \times 1$”, where ’n’ represents the image batch size in a single training step. Given the large size of these images, we employ a mini-batch method to train the model. We first divide the 4 million data points into 50 batches, each containing 80,000 data points. The model is then trained with 50 batches of data at a time. For each batch, we set the image training batch size to 128 images in the PyTorch dataloader. We still use

Nvidia V100 with 32G GPU memory on the Mist server to train this model.

3.3.3 Model Prediction and Results

Table 3.4: CNN Models RMSE on 5 Genes' Rates with 400 tree tips

beta_12	beta_13	beta_14	beta_15	beta_23
0.234079	0.226208	0.250063	0.241371	0.226899
beta_24	beta_25	beta_34	beta_35	beta_45
0.241046	0.221328	0.235981	0.233438	0.248801

For the CNN model, we only train with 10 output nodes with the interaction rates between different genes. As Table 3.4 and Table 3.3 show, the CNN model's RMSE (Root Mean Square Error) is better than the Simple Feedforward Neural Network by around 0.1, which is a crucial difference for the model performance and Hypothesis Testing in the following chapters.

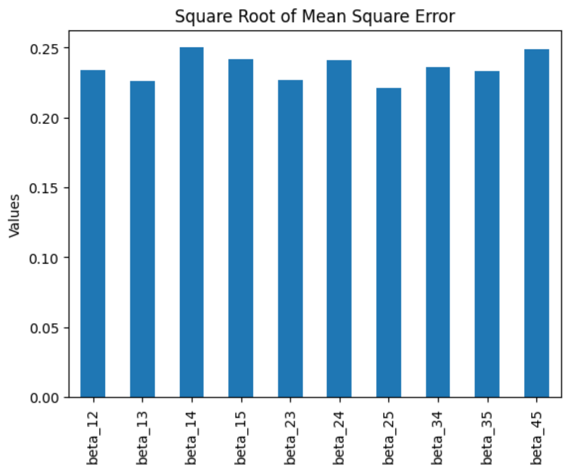


Figure 3.6: CNN model RMSE on 5 Genes' Rates with 400 tree tips.

In Figure 3.6, we found that the RMSE for the CNN model for different $B_{i,j}$ values are similar. The RMSE for $B_{i,j}$ values is around 0.23 to 0.25. This indicates that the CNN model is stable and consistent in predicting the interaction rates between different genes.

3.3.4 Advantages and Disadvantages of the Convolutional Neural Network

In our study, we investigate the performance of the Convolutional Neural Network (CNN) model on gene profile data. The CNN model is well-suited for processing image data, as it can effectively capture spatial patterns and relationships within the gene phylogenetic profile data. By treating the gene profile matrix as an image, we can leverage the CNN model's ability to learn complex spatial patterns from the data. The CNN model's architecture is designed to detect spatial patterns in images, making it an ideal choice for processing gene profile data. However, for the vanilla CNN model, we still need to design the CNN model structure, filter size, depth, pooling settings, and fully connected layers. This process can be time-consuming and may require significant expertise in designing CNN models. Especially when the number of genes is changed, along with the number of output nodes changes. Those may require a redesign of the model structure or fine-tuning the structure and layers. In the next section, we will introduce the ResNet model, which simplifies the design of CNN models and improves the model's performance.

3.4 ResNet, Residual Connection, and Bottleneck

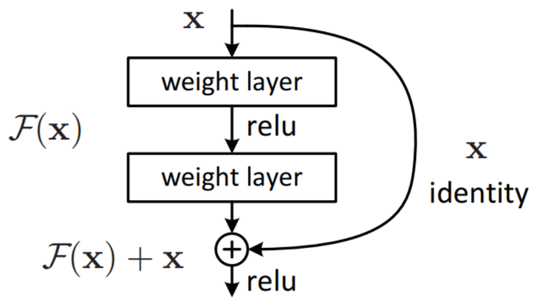


Figure 3.7: Residual Connection Block from [He et al., 2015]

ResNet, or Residual Network, is a convolutional neural network (CNN) architecture designed to tackle the degradation problem often encountered in very deep networks. This issue arises when the network's accuracy plateaus and then rapidly deteriorates as the network depth increases. Most of the time, when the deep learning model has more layers and is deeper, the performance of the model may have better results. Hence, ResNet addresses this problem by introducing an innovative "residual learning" framework. This framework employs skip connections or shortcuts, enabling the network to learn residual functions with reference to the layer inputs, rather than learning unreferenced functions. This method allows for the training of extremely deep networks by facilitating the direct backpropagation of gradients to earlier layers. There are two main components of ResNet: the Residual Connection and the Bottleneck Block. As Figure 3.7 shows, the Residual Connection, a crucial component of ResNet, refers to the skip connection used in the network's architecture. This connection allows for the direct backpropagation of the gradient to earlier layers, mitigating the vanishing gradient problem and enabling the training of deeper networks. In mathematical terms, a residual block is defined as $F(x)+x$, where $F(x)$ is the output of a series of layers, and x is the input to those layers. The addition operation is performed element-wise.

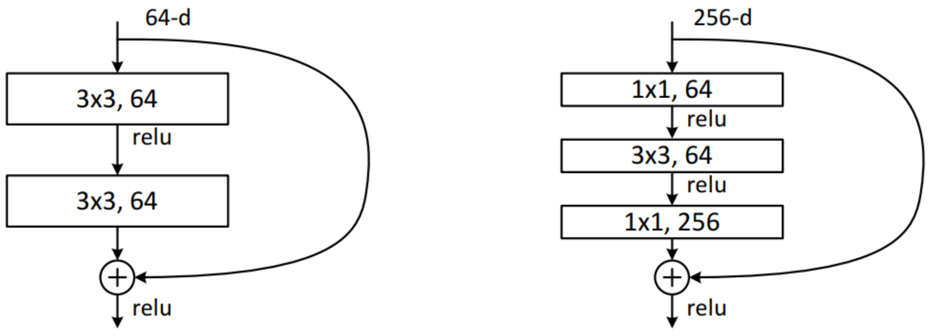


Figure 3.8: Left image: Building block for ResNet-34. Right image: BottleNeck Block from [He et al., 2015] in ResNet-50/101/152.

Within the context of ResNet, a Bottleneck that is shown in Figure 3.8, refers to a specific type of block used in the network architecture to reduce computational complexity in very deep networks. A bottleneck block consists of three convolutional layers: a 1x1 convolution for dimensionality reduction, a 3x3 convolution as the main convolution operation, and another 1x1 convolution for dimensionality restoration. This design helps reduce the number of parameters and computations while maintaining the network's representational power. Bottleneck blocks are commonly used in deeper versions of ResNet, such as ResNet-50 and ResNet-101, to improve efficiency without sacrificing accuracy. By using ResNet, we can simplify the design of the CNN model and improve the model's performance. The ResNet model is designed to address the degradation problem encountered in very deep networks, making it an ideal choice for processing gene profile data. In the following sections, we will compare the results with the vanilla CNN model, the ResNet model, and the CCM model.

Chapter 4

Hypothesis Testing, Model Comparison, and Other Experiments

4.1 Hypothesis Testing

Hypothesis testing is a concept in statistics used to make inferences about population parameters based on sample data. It involves two opposing hypotheses: the null hypothesis (H_0) and the alternative hypothesis (H_1). The null hypothesis represents the default assumption, representing that there is no difference between groups, while the alternative hypothesis is the hypothesis that we are trying to verify its difference. In our case, it is that the interaction indeed exists between two genes. The Hypothesis Testing contains a test statistic and a p-value. The test statistic measures how much the sample data deviates from what would be expected under the null hypothesis. The p-value is the probability of obtaining a test statistic as we observed, assuming the null hypothesis is true. We compare the p-value to a pre-determined significance level (α), with a range from 0.01 to 0.2. If the p-value is less than or equal to α, the null hypothesis is rejected, and the alternative hypothesis is accepted, indicating that there is enough evidence to support the alternative hypothesis. If the p-value is greater than α, the null hypothesis is not rejected, indicating that there is not enough evidence to support the alternative hypothesis. Hypothesis testing provides a way to evaluate the strength of evidence in support of a hypothesis. In our study, we use hypothesis testing to evaluate the interaction between genes and compare the performance of different models.

$$\begin{aligned} &H_0 : \text{There is no interaction between gene } i \text{ and gene } j,\ i \neq j \\ &H_1 : \text{There is an interaction between gene } i \text{ and gene } j,\ i \neq j \end{aligned} \tag{4.1}$$

In this phase of our study, we shift our focus from metrics like Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) to explore and discuss the potential interaction between two genes. In our case, we investigate whether there is an interaction between gene 1 and gene 2 to discover their interaction relationships. As shown in Equation 4.1, we employ Hypothesis Testing for this purpose, setting the Null Hypothesis as 'there is no interaction between the two genes', and the Alternative Hypothesis as 'there is an interaction between the two genes'. We use the p-value to test the Null Hypothesis.

$$\begin{aligned} \hat{z} &= \frac{\hat{B}_{ij}}{SE} \\ p &= 2 \times (1 - \Phi(|\hat{z}|)) \end{aligned} \tag{4.2}$$

As shown in Equation 4.2, we use the Standard Error as the denominator and estimated Beta values as the nominator to calculate the $\hat{z}$ scores. We then use the $\hat{z}$ scores to calculate the p-values. If the p-value is less than a certain threshold α, we reject the Null Hypothesis and accept the Alternative Hypothesis. For the Standard Error, we use the same method that is mentioned in Chapter 2. The parametric bootstrap method. We first generate a large number of gene profiles that use the same pre-set Beta values as the estimated parameters. Then we calculate the Standard Deviation of those estimated Beta Values as the Standard Error.

4.2 Network Structures of 5 Genes Data

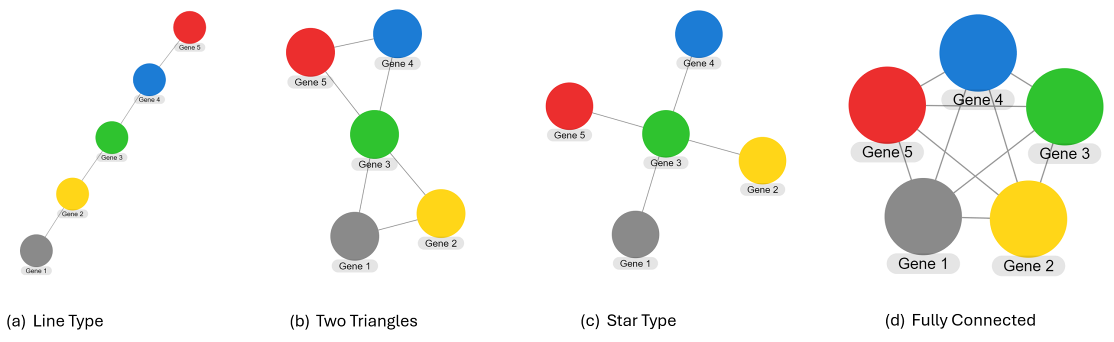


Figure 4.1: Four different types of network structures with 5 genes. From left to Right: (a) Line Type, (b) Two Triangles Type, (c) Star Type, (d) Fully Connected Type.

In our study, we will mainly focus on 4 different Network structures of 5 genes data. They are shown in Figure 4.1, including (a) Line Type, (b) Two Triangles Type, (c) Star Type, and (d) Fully Connected Type. The Line Type network structure is the simplest structure, where each gene is connected to the next gene. The Two Triangles Type network structure consists of two triangles, with each gene connected to the other gene in the same triangle, and Gene 3 is the connection node between two triangles. The Star Type network structure has one central gene (Gene 3) connected to all other genes. The Fully Connected Type network structure is the most complex, with each gene connected to all other genes. We will compare the performance of the CNN model and the EvolCCM model on these four network structures to evaluate their ability to capture the interaction between genes. The line between any two genes represents that there is an interaction between each other.

For our testing purposes, we generate 1,000 test data points for each network structure using the methods outlined in Chapter 2. Specifically, we set the generation parameter settings as follows: if there is no interaction, the interaction rate is set to 0. If there is an interaction, the interaction rate is set to 0.5, as shown in Table 4.1. These settings are chosen to align with those used in the CCM model's paper [Liu et al., 2022].

Table 4.1: Network Structures' Beta values for 5 Genes Data Generation

Network Types / $Beta_{(ij)}$	$\mathcal{B}_{12}$	$\mathcal{B}_{13}$	$\mathcal{B}_{23}$	$\mathcal{B}_{14}$	$\mathcal{B}_{24}$	$\mathcal{B}_{34}$	$\mathcal{B}_{15}$	$\mathcal{B}_{25}$	$\mathcal{B}_{35}$	$\mathcal{B}_{45}$
Line Type	0.5	0	0.5	0	0	0.5	0	0	0	0.5
Two Triangles	0.5	0.5	0.5	0	0	0.5	0	0	0.5	0.5
Star Type	0	0.5	0.5	0	0	0.5	0	0	0.5	0
Fully Connected	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

4.3 ROC Curve and Model Comparison between Vanilla CNN and CCM model

In our study, we applied the CCM model [Liu et al., 2022] and a trained vanilla CNN model checkpoint (From Chapter 3) to four different network structures' test datasets, as outlined in the previous section. For the purpose of hypothesis testing, we iteratively set the α value from 0.01 to 1.0, and Standard Error by using the parametric bootstrap sample method to construct the Receiver Operating Characteristic (ROC) curve, which plots the True Positive Rate (Sensitivity) against the False Positive Rate (1 - Specificity), as shown in Equation 4.3. This curve provides a visual representation of the model's ability to distinguish between two outcomes: interaction (positive) and no interaction (negative) between two genes. An ideal classifier would have an ROC curve that closely follows the top-left corner of the plot, indicating high sensitivity (the ability to correctly identify interactions) and a low false positive rate (the tendency to incorrectly classify non-interactions as interactions). Conversely, a random classifier would yield a diagonal line from the bottom-left to the top-right of the plot, reflecting equal chances of true positives and false positives across all thresholds. The Area Under the ROC curve (AUC) is a key metric used to quantify the overall performance of the model. A higher AUC value suggests better discrimination ability, with an AUC of 1.0 indicating a perfect classifier and 0.5 indicating a classifier that performs no better than random chance. When assessing model performance, we are looking for ROC curves that are both concave and positioned towards the top-left corner, indicating strong discriminatory power. Additionally, a steep curve and a significant distance from the diagonal line further reinforce the model's effectiveness in distinguishing between classes. Hence, the ROC curve and AUC provide valuable insights into the performance of binary classification models. This is particularly useful in our study, where we aim to identify the presence or absence of interactions between genes.

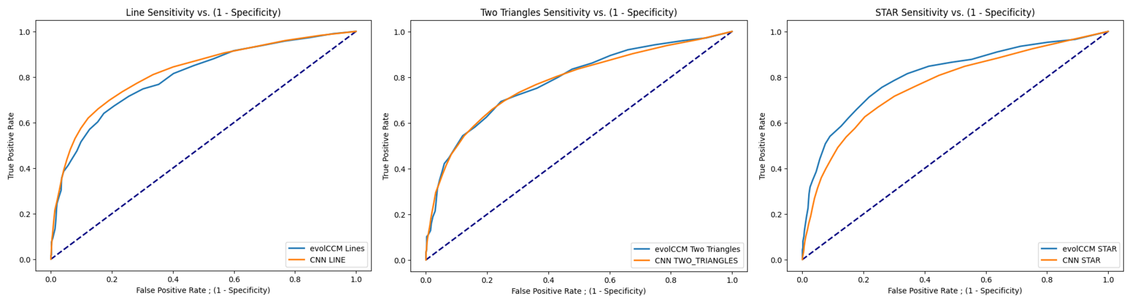


Figure 4.2: ROC Curve of CNN and CCM models on Network Structures with 5 genes. From left to Right: (a) Line Type, (b) Two Triangles Type, (c) Star Type

$$\begin{aligned}\text{False Positive Rate (FPR)} &= \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \\ \text{True Positive Rate (TPR)} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}\end{aligned} \tag{4.3}$$

To compare the performance of the trained vanilla Convolutional Neural Network (CNN) model with the CCM Model ([Liu et al., 2022]), we plotted the Receiver Operating Characteristic (ROC) curves for three network structures. However, due to the formula for False Positive Rate (FPR), and the fact that the fully connected network structure has no False Positives and True Negatives, the ROC curve for the Fully Connected network structure could not be displayed in Figure 4.2. From Figure 4.2, it is evident that the performance of the CNN model is very similar to that of the CCM model. As per [Liu et al., 2022], maximum likelihood methods should yield optimal performance. In Figure 4.2, we observed that in Line Type and Two Triangles Type Networks, the CCM model and Vanilla CNN model exhibit very similar performance. However, in the Star Type Network, the CCM model slightly outperforms the Vanilla CNN model. This suggests that in this particular trained Vanilla model checkpoint, the CNN model's performance is slightly worse. To further investigate this phenomenon, we plotted the Boxplot of the estimated Beta values from the Vanilla CNN model and the CCM model in Figure 4.3. We found that the CNN model has a narrow interquartile range (IQR) when the true Beta Values equal to 0, indicating that the middle 50% of the data points are closely clustered. However, there are more outliers.

At this stage, we hypothesize that the data generation sampling method may be affecting the model performance by generating excessive data around 0, using the Gaussian distribution with $\mu = -0.5$ and $\sigma = 1.5$. This could potentially cause the model to have a bias towards the 0 values. In the future section, we change the sampling method from Gaussian to Uniform distribution to investigate this hypothesis. In Figure 4.4, we plot the violin plot of the same estimated Beta values from two models. We can see that the CNN models' estimated Beta values of interacted genes have more spread distribution than the non-interacted genes. To solve this problem, in the next section, we will try to improve the model performance in two

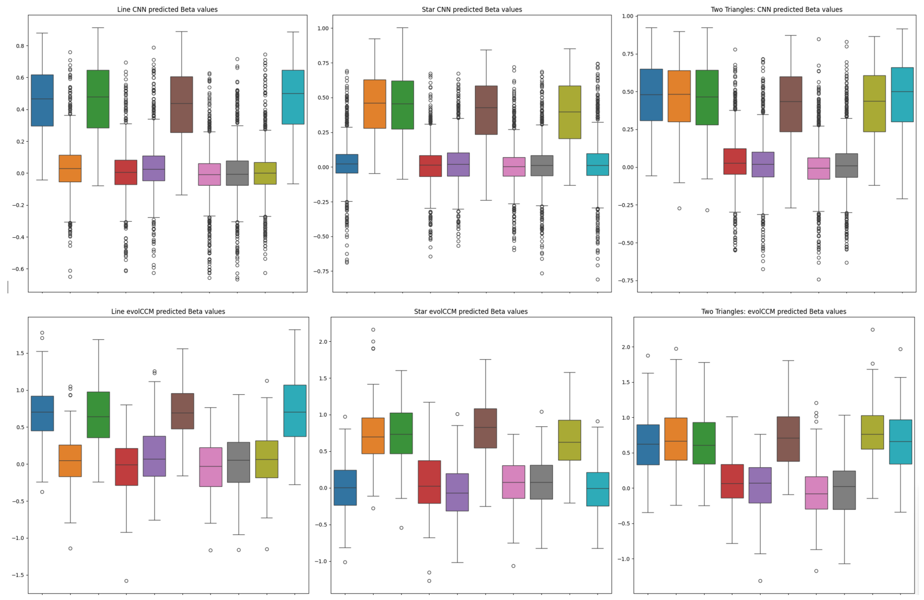


Figure 4.3: Vanilla CNN (Top) and CCM (Bottom) models' estimated Beta values' Boxplot

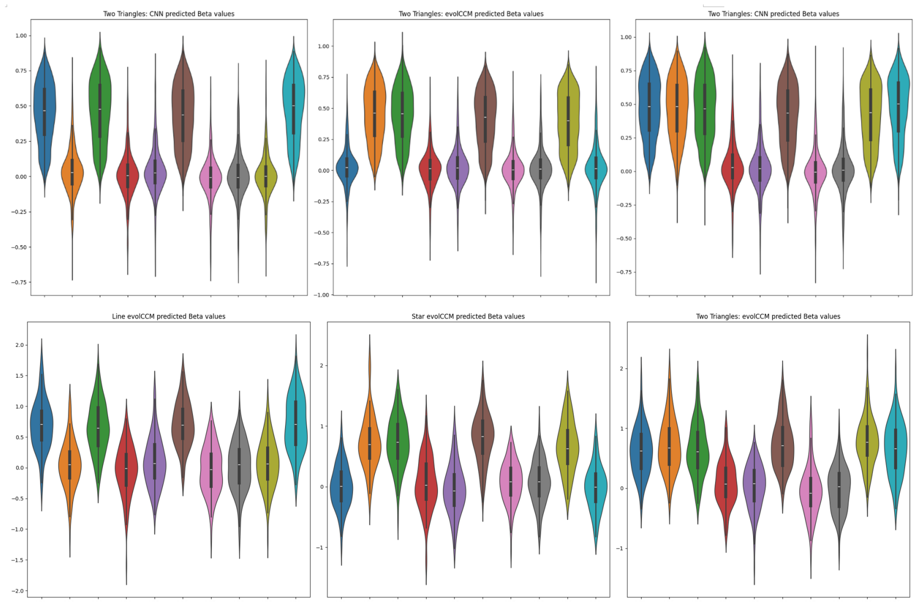


Figure 4.4: Vanilla CNN (Top) and CCM (Bottom) models' estimated Beta values' Violinplot

directions, first by using the uniform distribution to generate the gene rates/$Beta_{(i,j)}$ more evenly, then we will try to find the smallest number of data points that are needed to train the model to achieve the best performance.

4.4 Data Generation with Uniform Distribution Sampling

The CNN model trained in the previous sections utilizes the Gaussian Distribution to sample the Beta Values for data generation, adhering to the original settings from the evolCCM model's code repository. However, observations from the last section led us to hypothesize that the Gaussian Distribution might induce a bias towards the 0 values in the CNN model. To investigate this hypothesis, we modified the data generation sampling method from Gaussian to Uniform Distribution. We generated 200,000 gene rates/$Beta_{(i,j)}$ for training, using the Uniform Distribution within a range from -1.5 to 1.5, while maintaining all other settings. This configuration is referred to as the Pure Uniform Distribution.

Simultaneously, we introduced a Mixture of Uniform Distribution to test whether unbalanced Beta Values near Zero would impact the model performance when we also include more data points across the entire range of -1.5 to 1.5, which compares to the original Gaussian Distribution. In the Mixture Uniform Distribution setting, we initially generated each data point's Beta values from the Uniform Distribution within a range from -1.5 to 1.5. Then, we employed a Bernoulli Distribution with $p = 0.3$ to modify each Beta value in the data point. If the Bernoulli Distribution sampling resulted in 1, we replaced the Beta value with 0. This setting aims to mimic real-world scenarios where most gene interactions are near 0, and only a few gene interactions are strong. We also generated 200,000 gene rates/$Beta_{(i,j)}$ for training under this setting.

From Figures 4.5 to 4.8, we present the boxplot and violinplot of the estimated Beta values from the CNN model trained with both the Pure Uniform Distribution and the Mixture Uniform Distribution. We observed that the Pure Uniform Distribution exhibits a similar range of Interquartile Range (IQR) for both interacted and non-interacted gene relationships, with very few outliers. However, the Mixture

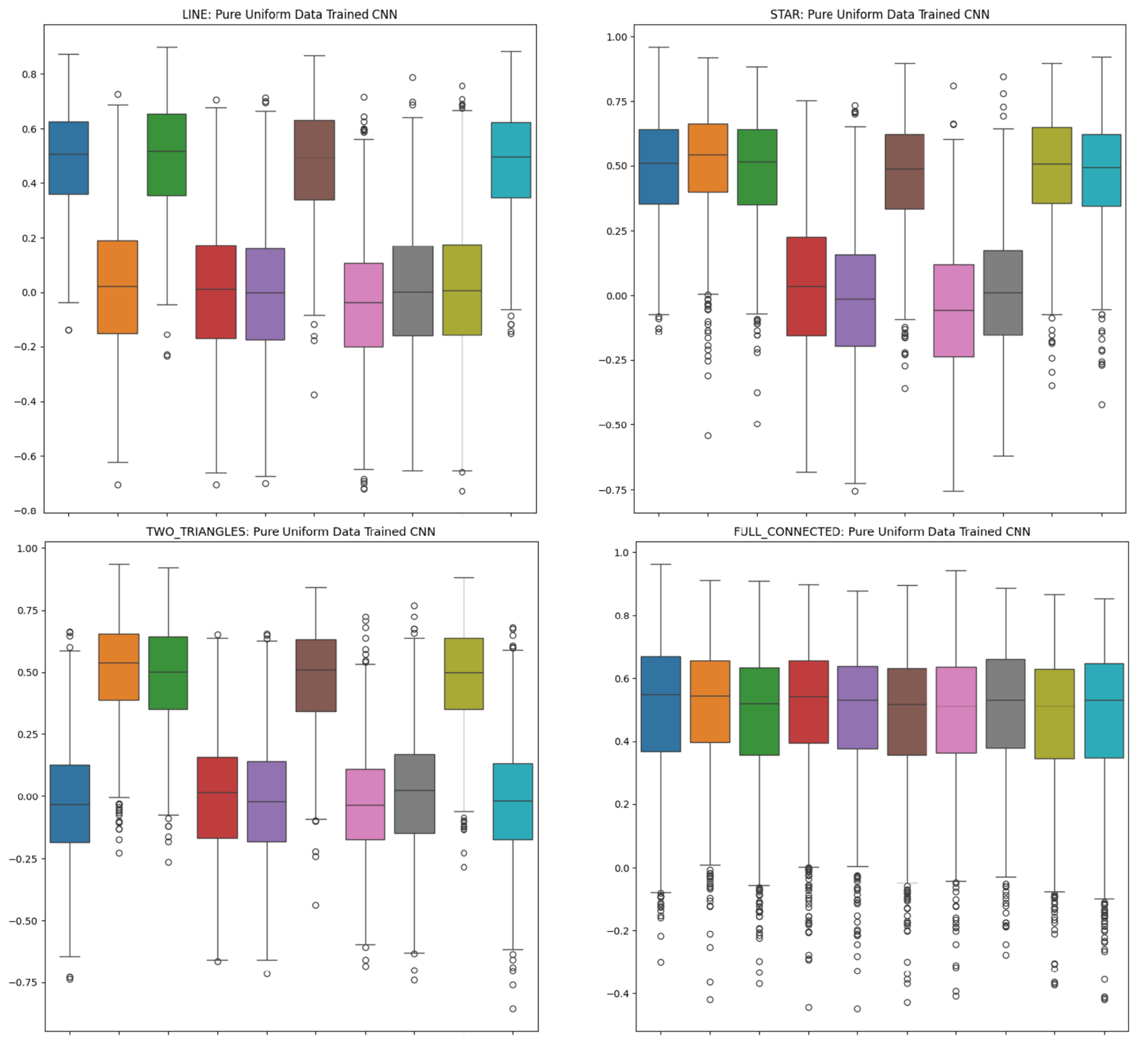


Figure 4.5: Boxplot of the four Network Structures with 5 genes' Beta values from the CNN model trained with Pure Uniform Distribution $\mathcal{B}_{i,j} \sim \mathcal{U}(-1.5, 1.5)$

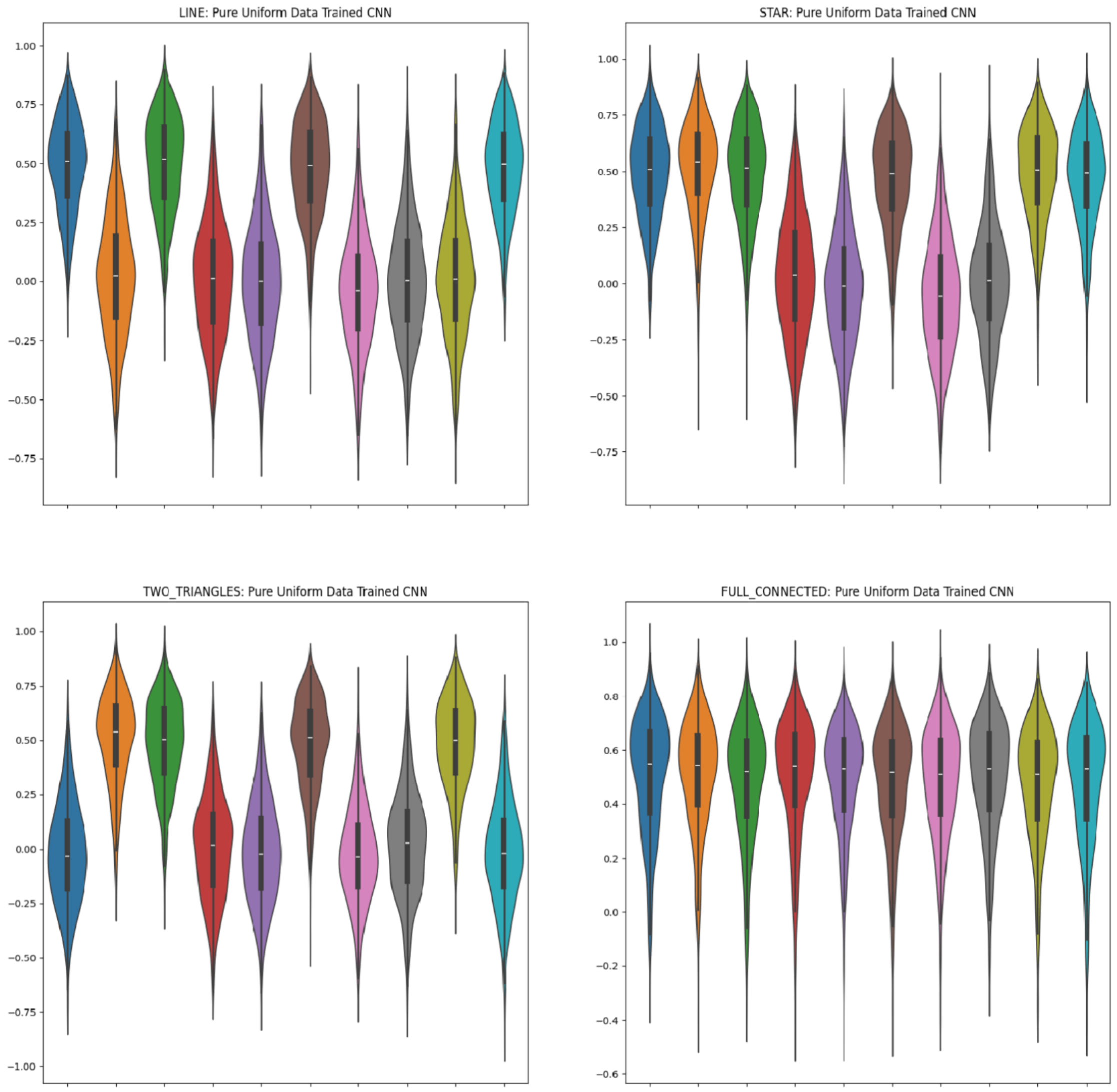


Figure 4.6: Violinplot of the four Network Structures with 5 genes' Beta values from the CNN model trained with Pure Uniform Distribution $\mathcal{B}_{i,j} \sim \mathcal{U}(-1.5, 1.5)$

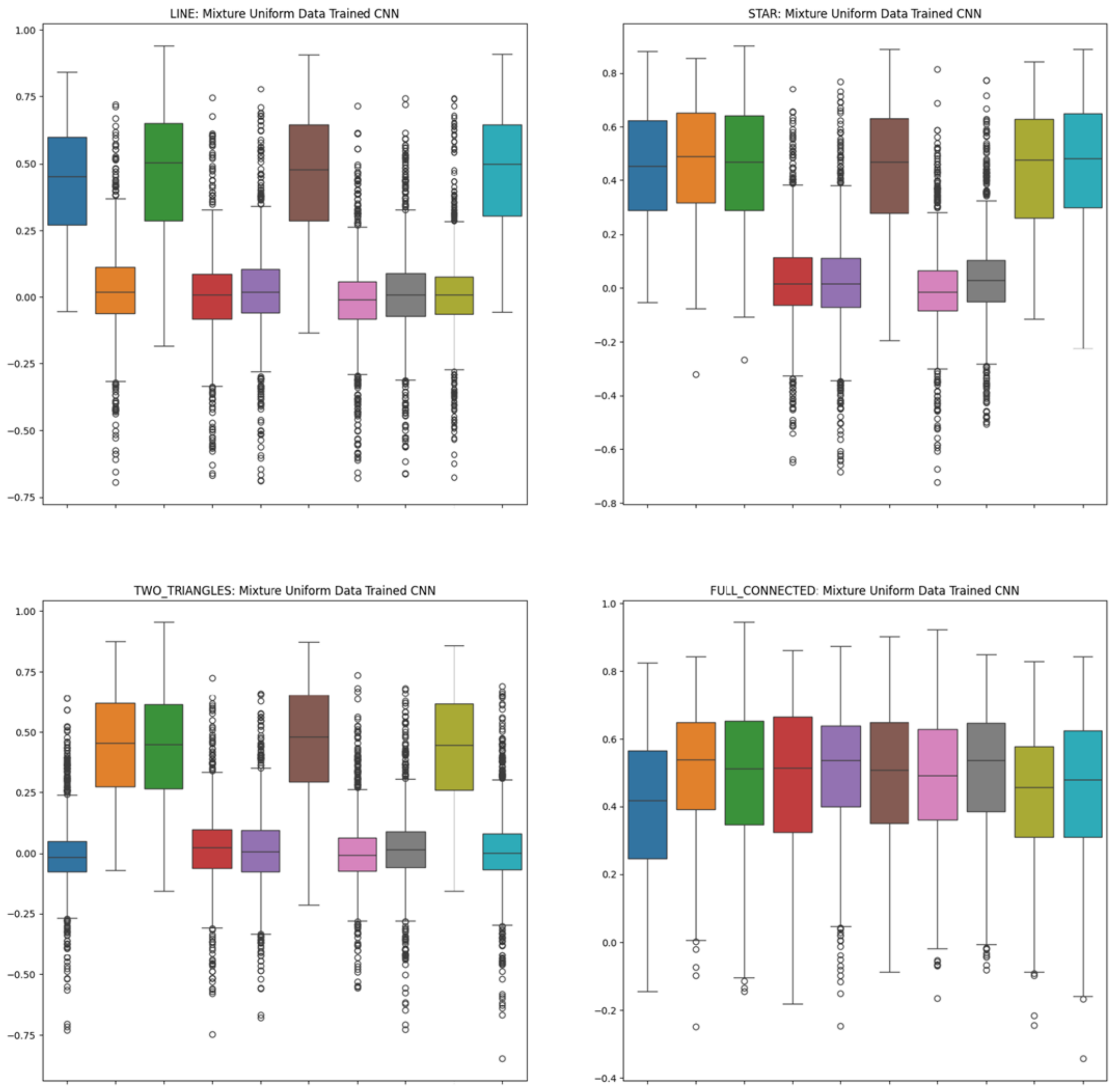


Figure 4.7: Boxplot of the four Network Structures with 5 genes' Beta values from the CNN model trained with Mixture Uniform Distribution $\mathcal{B}_{i,j} \sim \mathcal{U}(-1.5, 1.5)$ with Bernoulli Distribution $p = 0.3$ to replace the $\mathcal{B}_{i,j}$ with 0.

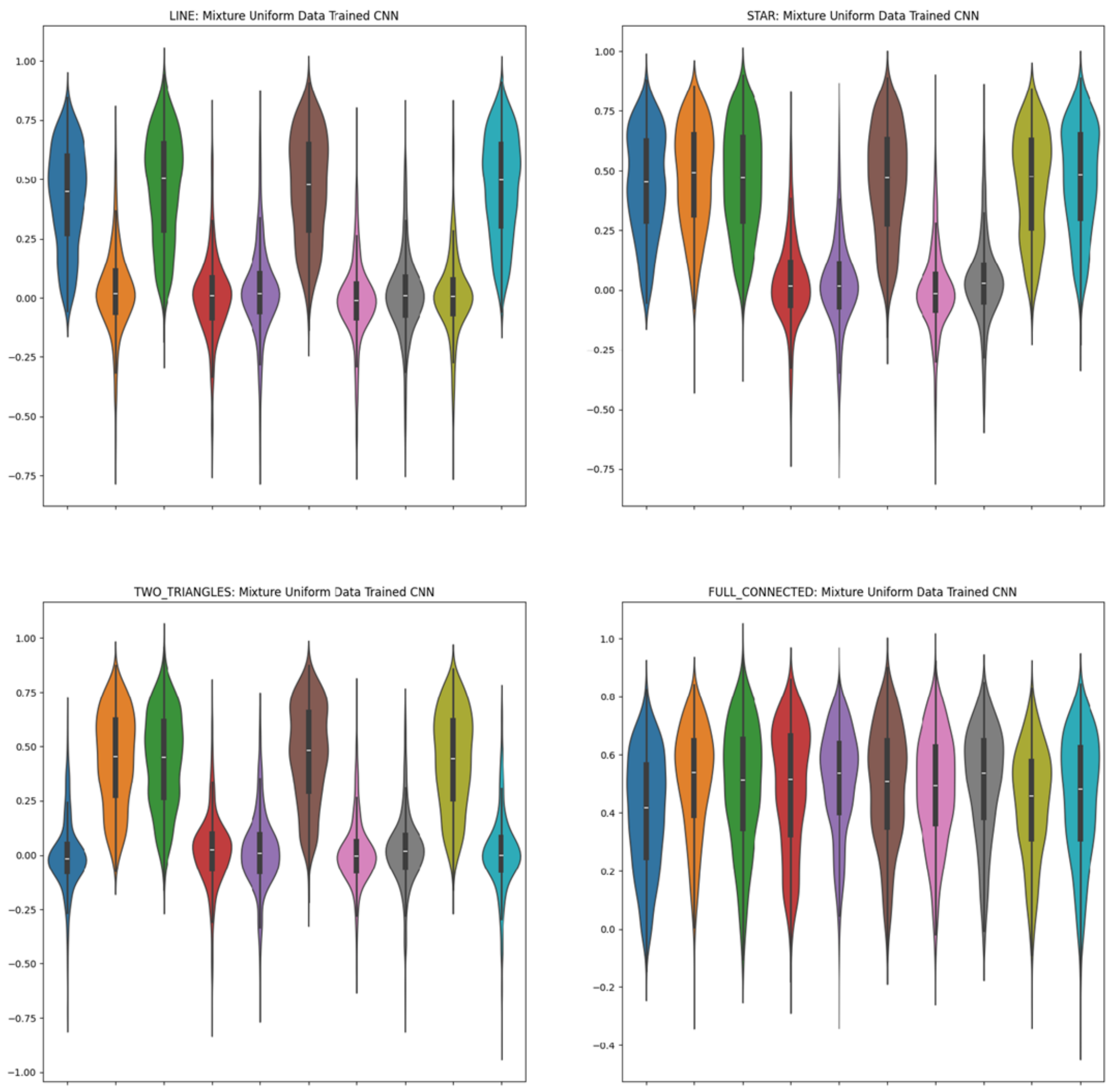


Figure 4.8: Violinplot of the four Network Structures with 5 genes' Beta values from the CNN model trained with Mixture Uniform Distribution $\mathcal{B}_{i,j} \sim \mathcal{U}(-1.5, 1.5)$ with Bernoulli Distribution $p = 0.3$ to replace the $\mathcal{B}_{i,j}$ with 0.

Uniform Distribution displays a phenomenon similar to the Gaussian Distribution, with a narrow range of IQR for the non-interacted gene relationships and a significant number of outliers. This suggests that the Mixture Uniform Distribution may also exhibit a bias towards the 0 values, similar to the Gaussian Distribution.

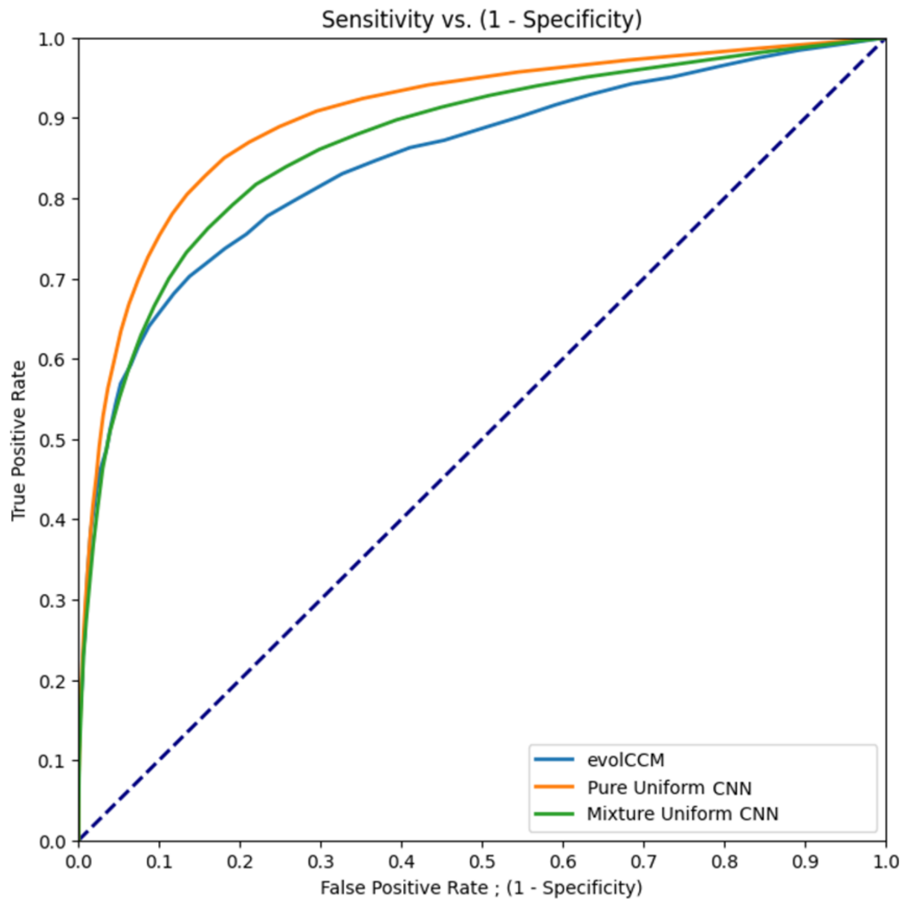


Figure 4.9: ROC curve with Pure and Mixture Uniform Distribution for the CNN model on combined ALL Network Structures with 5 genes.

In Figure 4.9, we plot the Receiver Operating Characteristic (ROC) curve for the CNN model trained with the Pure Uniform Distribution, the Mixture Uniform Distribution, and the EvolCCM model [Liu et al., 2022]. These models were tested on a dataset containing all four network structures. We observed that the CNN model trained with the Pure Uniform Distribution slightly outperforms the model trained with the Mixture Uniform Distribution and the EvolCCM model. That may confirm our hypothesis. Hence, from now on, we will use the Pure Uniform Distribution $\mathcal{B}_{i,j} \sim \mathcal{U}(-1.5, 1.5)$ to generate the gene rates/$Beta_{(i,j)}$ for training the CNN model.

4.5 Finding the Minimum Dataset Size for Training

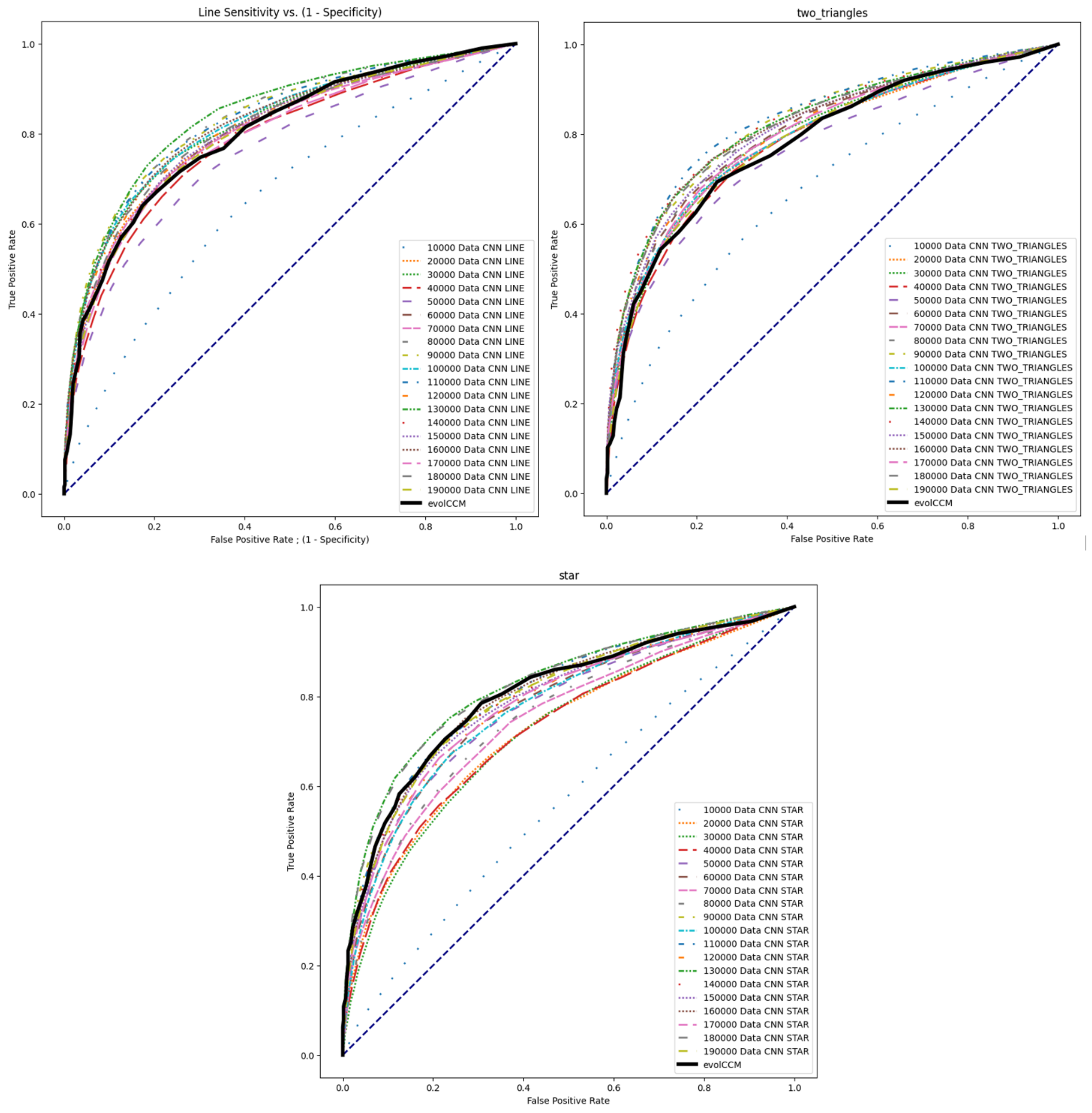


Figure 4.10: ROC Curve Analysis for CNN Model Trained with Pure Uniform Distribution Across Varying Training Set Sizes (10K to 190K)

Based on the previous result, we can see that a properly generated training dataset can indeed increase the model performance. Hence, we are also interested in finding the minimum number of data points needed to train the model to achieve the best performance. To investigate this, we generated 10,000, 20,000, ..., up to 190,000 data points for training the CNN model with the Pure Uniform Distribution. We developed an automated code base for training the CNN model, along with an early stopping mechanism. This mechanism stops the training process when there's no improvement in the model's performance on the validation set over 30 consecutive epochs. Then automatically starts to train the new training set with 10K more data points. We then tested the trained model on the test set with different network structures. We plotted the Receiver Operating Characteristic (ROC) curve for each dataset size on each network structure to compare the model performance. From Figure 4.10, we observed that the model performance improves as the dataset size increases. However, the performance improvement diminishes as the dataset size exceeds 130,000 data points. This suggests that the CNN model trained with the Pure Uniform Distribution requires a minimum of 130,000 data points to achieve a similar performance of the CCM model in 5 genes data.

4.6 Swapping Positions of the Genes in the Gene Profile Image

We also aim to determine whether our model is robust to changes in the positions of genes within the gene profile image. For instance, if the original order of genes is [Gene 1, Gene 2, Gene 3, Gene 4, Gene 5], we rearrange the positions to [Gene 5, Gene 4, Gene 3, Gene 2, Gene 1] and assess whether the performance remains consistent. To achieve this, we first document the changes in gene order from the gene profile, then expand the 5 genes to a 200-image width by repeating them 40 times. We use the pre-trained CNN checkpoint to predict this reordered test set data. It's important to note that such reordering will also alter the positions of the output nodes' values. Consequently, we need to revert the model predictions from the permutated order back to the original order for straightforward comparison. The detailed pseudocode is presented in Algorithm 3.

In Figure 4.11, we draw the Boxplot of the estimated Beta values from one of the

Algorithm 3: Generate Permutations and Permutated Genes Predictions

```
 1: procedure GeneratePermutations( )
 2:     Initialize a list of elements from 0 to 4
 3:     Generate all permutations of the elements
 4:     Convert each permutation from a tuple to a list
 5:     return the list of permutations
 6: end procedure
 7: procedure GetPermutatedPredictions($all\_permutations, ...$)
 8:     Record the Orginal Order as $Beta_{(i,j)}$: $\mathcal{B}_{12}$, $\mathcal{B}_{13}$, $\mathcal{B}_{23}$, $\mathcal{B}_{14}$, $\mathcal{B}_{24}$, $\mathcal{B}_{34}$, $\mathcal{B}_{15}$, $\mathcal{B}_{25}$,
    $\mathcal{B}_{35}$, $\mathcal{B}_{45}$
 9:     Select a permutation from the list of $all\_permutations$ from
    Procedure{GeneratePermutations(...)} using $permutation\_selection\_index$
10:     Repeat the selected permutation 40 times to create a new list called
    “permutated order” to construct the new gene profile image with a width of 200
11:     Reorder the original $X\_test$ based on the new list “permutated order”
12:     Use the reordered $X\_test$ to make predictions with the model
13:     Initialize a list of original gene orders
14:     For each pair of genes in the original order, sort the pair based on the
    selected permutation and their column name, then append it to a new list,
    prefixing it with '$\mathcal{B}$'
15:     Rename the columns of the prediction results using the new list of sorted
    gene pairs
16:     Reorder the columns of the prediction results based on the original order
17:     return the reordered prediction results
18: end procedure
19: Generate all permutations
20: Get permutated predictions using a specific permutation, test data, the trained
    model checkpoint, selected GPU device, loss criterion (MSE), and a batch size
    of 128
```

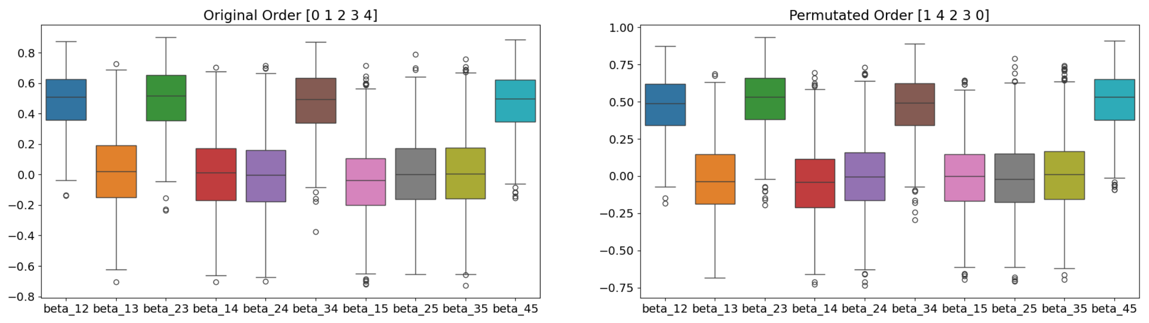


Figure 4.11: Boxplot of the estimated Beta values from Permutated Genes' Profile Order and Original Genes' Profile Order data with the same trained CNN model from the previous section.

Permutated Genes' Profile Order and Original Genes' Profile Order data with the same trained CNN model from the previous section. We can see that their estimated values are very similar, indicating that the model is very likely robust to changes in the positions of genes within the gene profile image. This is a crucial finding, as it suggests that the model can generalize well to different gene orders, which is essential for real-world applications where the gene order may vary. To further investigate this, we involve a size curve for all permutated gene orders in Figure 4.12. The size curve is a graphical representation used in hypothesis testing to illustrate the relationship between the significance level (α) and the false rejection rate. The significance level α represents the probability of incorrectly rejecting the null hypothesis when it is actually true, while the false rejection rate is the proportion of times the null hypothesis is incorrectly rejected in repeated hypothesis tests when it is actually true. Ideally, the size curve should be a straight line, indicating that the false rejection rate is equal to the significance level (α) across all values of α. Deviations from this ideal curve can indicate issues such as test bias or inefficiency in controlling the Type I error rate. Understanding the size curve is crucial as it helps assess the trade-off between the risk of incorrectly rejecting a true null hypothesis (Type I error) and the probability of correctly rejecting a false null hypothesis (power).

Typically, we are particularly interested in the range of significance levels from 0.01 to 0.2. In Figure 4.12, we plot the size curve with significance levels ranging from 0.01 to 1.0 to observe the differences between various permuted orders across

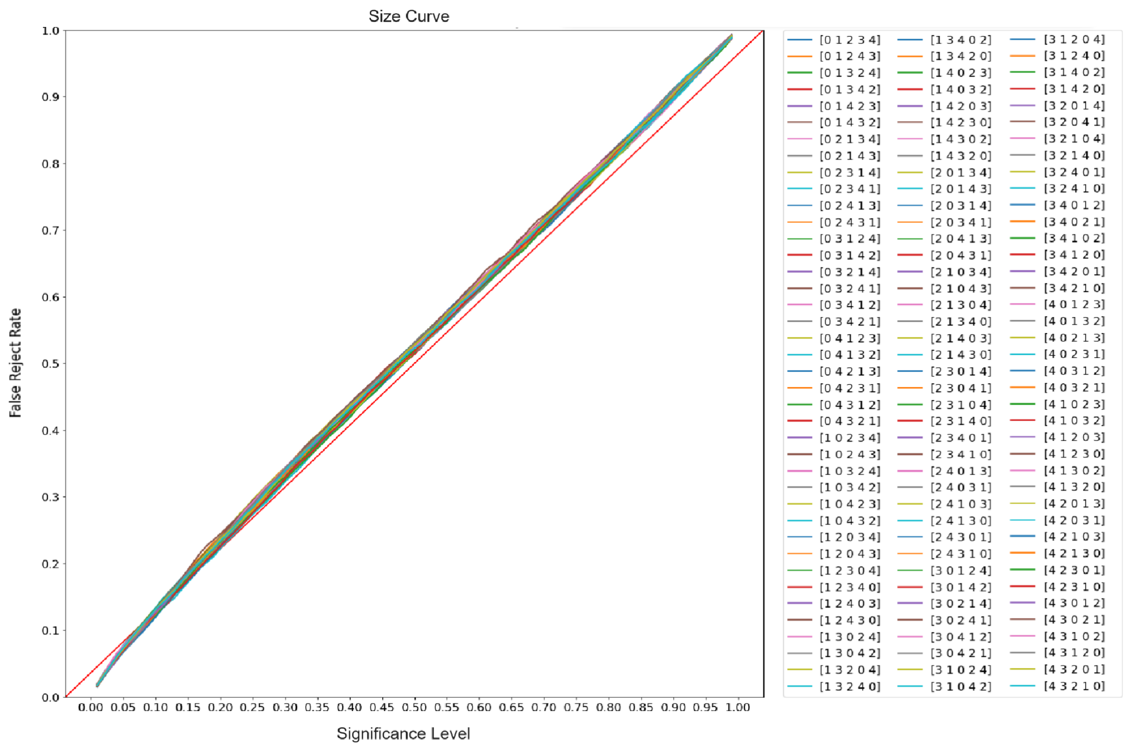


Figure 4.12: Size Curve Analysis for CNN Model with Gene Profile all Permutations Orders

the full range. As can be seen from Figure 4.12, beyond $\alpha = 0.05$, the size curves closely align with the ideal line, and all permuted gene orders exhibit very similar performance. The size curves remain close to the ideal line up to $\alpha = 0.2$. This suggests that the model generalizes well from $\alpha = 0.05$ to 0.2. Beyond $\alpha = 0.2$, we observe that the performance across all permuted gene orders remains remarkably similar. This indicates that the model is robust to changes in the positions of genes within the gene profile image.

4.7 Phylogenetic Tree Structures in the Training Data

Table 4.2: Comparison of RMSE for CNN Models Trained on 5 Genes' Rates with Fixed and Dynamic Phylogenetic Tree Structures in Data Generation Settings

Tree Type / Loss Type	Fixed Tree Structure	Dynamic Tree Structure
RMSE	0.234079	0.236208

When we initially trained our CNN model, we used the same fixed phylogenetic tree structure for all gene profiles in the training data. However, in real-world scenarios, the phylogenetic tree structure may differ across various gene profiles. The EvolCCM model requires the input to include both gene profiles and the phylogenetic tree structure. In contrast, for the CNN model, we only input the gene profile as images, assuming that the phylogenetic tree structure is embedded within the image. Therefore, after a few initial training attempts, we changed our data generation mechanism so that each data point has a unique, randomly generated tree structure. In this setting, we can train the CNN model with a dynamic phylogenetic tree structure. We observed that the MSE of the CNN models trained with fixed and dynamic phylogenetic tree structures is almost identical. This suggests that the CNN model is stable to changes in the phylogenetic tree structure in the training data. It implies that the model can generalize well to different phylogenetic tree structures, which is crucial for real-world applications where the phylogenetic tree structure may vary.

4.8 False Discovery Rate Control with Benjamini-Hochberg Correction

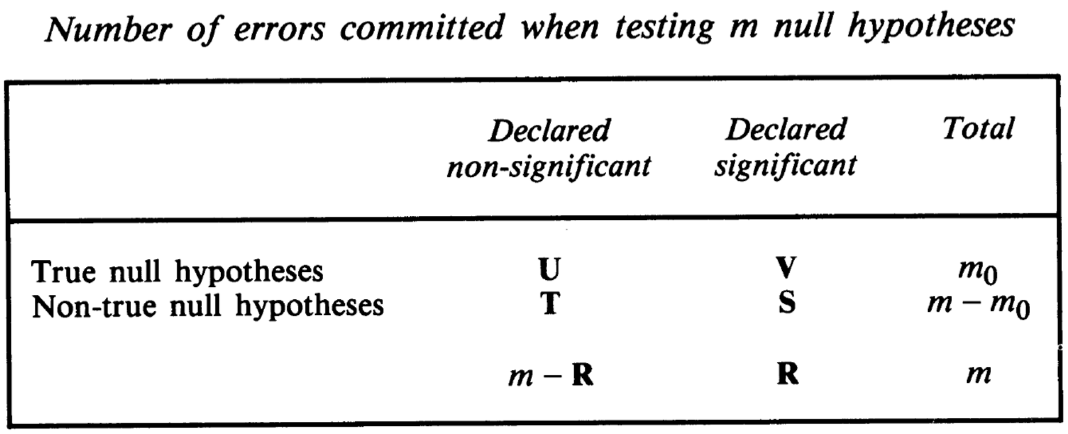
As [Rouam, 2013] stated, the False Discovery Rate (FDR) is a way to handle the problem of multiple comparisons in statistics. It's often used in experiments where a lot of tests are done at once, to fix issues where something might look important just by chance. When we test a null hypothesis to see if a result is statistically significant, we calculate a p-value and compare it to a threshold α. If we test k hypotheses at the same time with a confidence level α, the chance of getting false positives (that is, we say the null hypothesis is wrong when it's actually right) is 1 - (1 - α) $\times$ k. This can make the error rate in the experiment pretty high. So, we use something like the FDR to adjust our confidence levels based on how many tests we're doing. To control the False Discovery Rate, we apply the Benjamini-Hochberg Correction [Benjamini and Hochberg, 1995] on the estimated Beta values from the model predictions. As the last section in this chapter, the concept of False Discovery Rate is crucial in future analysis and is new to our current study stage. Here, we quote and give the entire definition of the False Discovery Rate and a summary of BH correction from [Benjamini and Hochberg, 1995] to illustrate the related works.

4.8.1 False Discovery Rate Definition

From [Benjamini and Hochberg, 1995], we quote the following definition of the False Discovery Rate (FDR).

"Consider the problem of testing simultaneously m null hypotheses, of which m_0 are true. $\mathcal{R}$ is the number of hypotheses rejected. Figure 4.13 summarizes the situation in a traditional form. The specific m hypotheses are assumed to be known in advance. $\mathcal{R}$ is an observable random variable; $\mathcal{U}, \mathcal{V}, \mathcal{S}$, and $\mathcal{T}$ are unobservable random variables. If each individual null hypothesis is tested separately at level α, then $\mathcal{R} = \mathcal{R}(\alpha)$ is increasing in α. We use the equivalent lowercase letters for their realized values.

In terms of these variables, the PCER (per-comparison error rate) is $\mathbf{E}(\mathcal{V}/m)$ and the FWER (the family-wise error rate) is $\mathbf{P}(\mathcal{V} \geq 1)$. Testing individually each



Number of errors committed when testing m null hypotheses

	Declared non-significant	*Declared significant*	*Total*
True null hypotheses	**U**	**V**	m_0
Non-true null hypotheses	**T**	**S**	$m - m_0$
	$m - \mathbf{R}$	**R**	m

Figure 4.13: Number of Errors committed when testing m null hypotheses. From [Benjamini and Hochberg, 1995]

hypothesis at level α guarantees that $\mathbf{E}(\mathcal{V}/m) \leq \alpha$. Testing individually each hypothesis at level α guarantees that $\mathbf{P}(\mathcal{V} \geq 1) \leq \alpha$.

The proportion of errors committed by falsely rejecting null hypotheses can be viewed through the random variable $\mathcal{Q} = \mathcal{V}/(\mathcal{V}+\mathcal{S})$ - the proportion of the rejected null hypotheses which are erroneously rejected. Naturally, we define $\mathcal{Q} = 0$ when $\mathcal{V}+\mathcal{S} = 0$, as no error of false rejection can be committed. $\mathcal{Q}$ is an unobserved (unknown) random variable, as we do not know q or s, and thus $q = v/(v+s)$, even after experimentation and data analysis. We define the FDR $\mathcal{Q}_c$ to be the expectation of $\mathcal{Q}$.

$$\mathcal{Q}_c = \mathbf{E}(\mathcal{Q}) = \mathbf{E}(\mathcal{V}/(\mathcal{V}+\mathcal{S})) = \mathbf{E}(\mathcal{V}/\mathcal{R}) \tag{4.4}$$

Two properties of this error rate are easily shown, yet are very important.

- If all null hypotheses are true, the FDR is equivalent to the FWER: in this case, $s = 0$ and $v = r$, so if $v = 0$ then $\mathcal{Q} = 0$, and if $v > 0$ then $\mathcal{Q} = 1$, leading to $\mathbf{P}(\mathcal{V} \geq 1) = \mathbf{E}(\mathcal{Q}) = \mathcal{Q}_c$. Therefore control of the FDR implies control of the FWER in the weak sense.

- When $m_0 < m$, the FDR is smaller than or equal to the FWER: in this case, if $v > 0$ then $v/r \leq 1$, leading to $\mathcal{X}_{(V \geq 1)} \geq \mathcal{Q}$. Taking expectations on both sides we obtain $\mathbf{P}(\mathcal{V} \geq 1) \geq \mathcal{Q}_c$, and the two can be quite different. As a result, any procedure that controls the FWER also controls the FDR. However, if a procedure controls the FDR only, it can be less stringent, and a gain in power may be expected. In particular, the larger the number of the non-true null hypotheses is, the larger $\mathcal{S}$ tends to be, and so is the difference between the error rates. As a result, the potential for an increase in power is large when more of the hypotheses are non-true."

4.8.2 Benjamini-Hochberg Correction

From [Benjamini and Hochberg, 1995], we summarize the BH correction as follows. BH correction is a method used to control the False Discovery Rate (FDR) in multiple hypothesis testing. It is used to reduce the chance of false positives or Type I errors.

The procedure is as follows:

- Calculate the p-values for each estimated Beta value from the model predictions by using the test statistics of $\mathcal{B}_{(i,j)}/SE$, where SE is using the parametric bootstrap method to calculate the Standard Error.
- Rank the p-values from all tests in ascending order.
- Calculate the adjusted p-values $\hat{p} = \alpha \times i/m$, where α is the significance level, i is the rank of the p-value, and m is the total number of tests.
- If the adjusted p-values are less than the significance level α, reject the null hypothesis.

4.8.3 False Discovery Rate Control with Benjamini-Hochberg Correction on CNN model

In Figure 4.14, we plot the significance level v.s. the False Discovery Rate plot of the CNN model's original p-values of the estimated Beta values for all four network structures, and the CNN model's adjusted p-values' plot with Benjamini-Hochberg Correction. The range of significance level we choose is from 0.01 to 0.2. We can see that the original estimated Beta values' p-values have a higher False Discovery

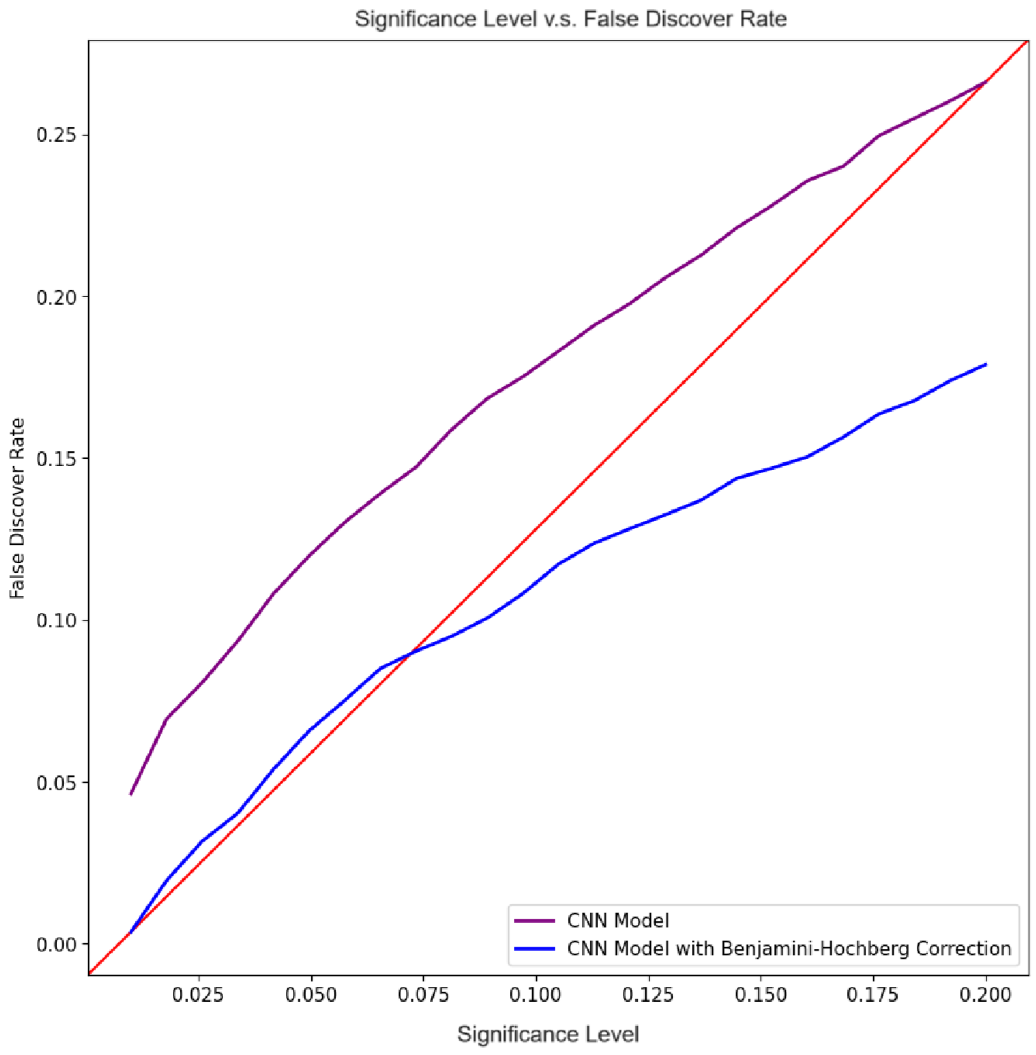


Figure 4.14: False Discovery Rate for CNN Model Comparison with Benjamini-Hochberg Correction

Rate than the adjusted p-values with Benjamini-Hochberg Correction everywhere in Figure 4.14. The False Discovery Rate is well controlled under the significance level $\alpha = 0.01$ to $\alpha = 0.07$ with Benjamini-Hochberg Correction. This suggests that the Benjamini-Hochberg Correction is a useful method to control the False Discovery Rate in our CNN model.

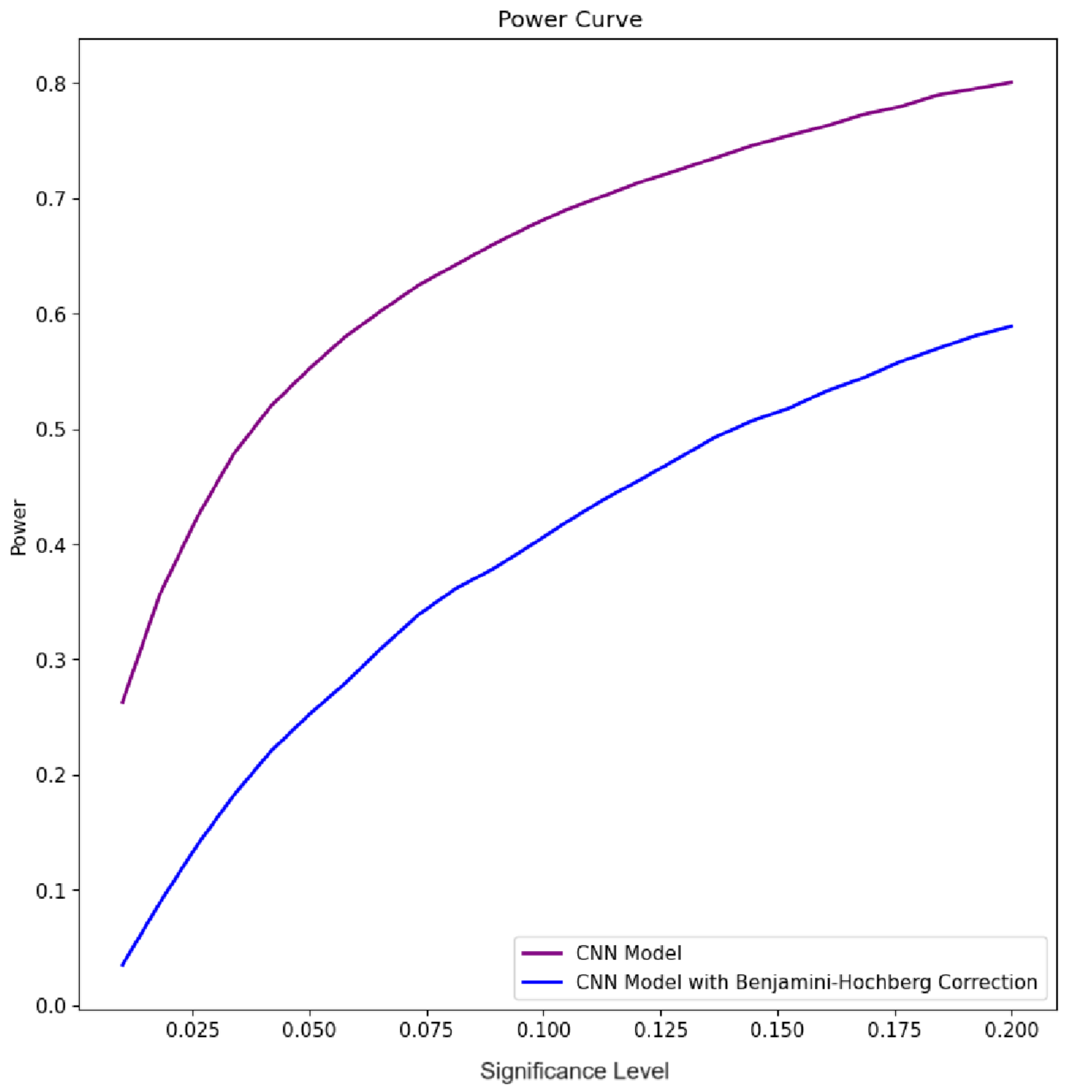


Figure 4.15: power Curve for CNN Model Comparison with Benjamini-Hochberg Correction

In Figure 4.15, we plot the power curve of the CNN model's original p-values of the estimated Beta values for all four network structures, and the CNN model's adjusted p-values' plot with Benjamini-Hochberg Correction. We can see that the power curve of the adjusted p-values with Benjamini-Hochberg Correction is lower than the original p-values of the estimated Beta values for all four network structures. It shows that there is a compromise between the False Discovery Rate and the power of the model. The Benjamini-Hochberg Correction can control the False Discovery Rate well, but it may reduce the power of the model. This is a trade-off that we need to consider when using the Benjamini-Hochberg Correction in our CNN model.

Chapter 5

Scale Up and Future Works

This Chapter contains multiple experiments that are still ongoing. From now on, we switch from the vanilla CNN model to ResNet to incorporate the varied changes in the number of tree tips and the number of genes. By using the ResNet, we have one of the best CNN models currently in the world, and no need to design the model structure if the number of genes or tree tips changes. We can directly use the ResNet model to train the data. The ResNet can automatically adjust the model structure to fit different sizes of the image inputs. Its downsampling layers can reduce the number of parameters in the model even if the input image or the number of output nodes have a very large size. We add a couple of linear layers at the end of ResNet to suit our regression tasks with the MSE loss criterion. Since it is still an ongoing experiment, we only have the preliminary results for Mean Square Error on the test set. We will continue to update the results in the future.

The training time for a very large number of data points and a large image size is very long. We utilize the full power of the Mist cluster with multiple nodes simultaneously that have Nividia 32G GPU by training for more than 1 month with many different settings. The potential of scaling up the CNN model is very exciting. Since the CCM model [Liu et al., 2022] is limited by the computation resources of $\mathcal{Q}$ matrix size, it can estimate less than 10 genes. We were hoping that our new CNN model with large-scale simulated gene data could create a new benchmark for gene rate estimation. By the Advantages of transfer learning, our trained model checkpoints can be used for fine-tuning in other settings in the future.

5.1 Increasing the Number of Tree tips

By the prior work in [Liu et al., 2022], we anticipate that the model performance will increase when there are more tree tips / the number of tree leaves in the gene profile image. Hence, we trained four "1.6 million" datasets with 100, 400, 1000, and

2000 tree tips, respectively on 5 genes' profile images. We used the ResNet-34 model to train the data with the same settings as the previous CNN model. We observed that the Root Mean Square Error (RMSE) on the test set decreases as the number of tree tips increases.

Table 5.1: Comparison of RMSE for ResNet-50 Trained on 5 Genes' Rates with phylogenetic tree tips of 100, 400, 1000, and 2000

Number of Tree Tips Loss Type	100	400	1000	2000
RMSE	0.2525	0.2362	0.2000	0.1760

From Table 5.1, we can confirm that the model performance increases when we increase the number of tips in the phylogenetic tree on a large-scale dataset. Here we list one of the training bash scripts on the Mist cluster for our code base that will be published on GitHub in the future.

```bash
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --gpus-per-node=1
#SBATCH --time=23:59:45
#SBATCH --job-name training_job
#SBATCH --output=/scratch/***/logs/5_genes_1000_tips_3rd_training_output_%j.txt
#SBATCH --mail-user=zs******@dal.ca
#SBATCH --mail-type=ALL

module load anaconda3
source activate genes_env

python ./CNN/scripts/train.py \
    --main_dir ./simulations_data/1000_tips_tree_5_genes_1600000_records \
    --load_model_checkpoint 1 \
    --model_checkpoint_path ./simulations_data/1000_tips_tree_5_genes_1600000_records/model_checkpoints/batch_best_model_0.04.pth \
    --epochs 1000 \
    --batch_size 128 \
    --num_outputs 10 \
    --sub_training_batch 1\
    --input_gene_image_size "1, 1000, 200" \
    --log_file_name "5
_genes_1000_tips_3rd_ResBet50_img_batch_128_sub_batch_1_1e-4" \
    --model_type "ResNet" \
    --ResNet_depth 50 \
    --learning_rate 1e-4\
```

5.1.1 Inference Time of the CNN and CCM Model

We also compared the inference time of the ResNet-50 model and the CCM model with 1000 tree tips and 5 genes' data in Table 5.2. We found that the ResNet-50 model can predict the rates of 5 genes' interactions in 3 seconds, while the CCM model takes 5 hours to predict the same data. This suggests that the ResNet-50 model is much faster than the CCM model in terms of inference time. It overcomes the limitation of the CCM model in terms of computation time.

Table 5.2: Comparison of Inference Time for ResNet-50 and CCM Model with 1000 data points of 5 Genes' data

Model Type / Inference Time	CNN/ResNet Model	EvolCCM Model
Total Time	3 seconds	5 Hours

5.2 Increasing the Number of Genes

The ultimate goal of our study is to estimate the genes' rates for a large number of genes. We trained the ResNet-18, 34, 50, 101, and 152 models on multiple different numbers of genes' datasets. We changed the model hyperparameters, the number of tree tips, the size of training data, and some other settings for searching for the best model performance. So far, we have tested 5 genes, 10 genes, 20 genes, and 40 genes' profiles with their corresponding number of rates from n choose 2. Since when the ResNet model structure goes deeper, the training time will be much longer, we choose to show the results from ResNet-50 which is trained on all different settings. Since the model performance between ResNet-34/50 and ResNet-101/152 is very similar. The model complexity itself won't make a huge difference in the RMSE level.

Table 5.3: Comparison of RMSE for ResNet-50 Trained on Genes' Profile with 5 genes, 10 genes, 20 genes, and 40 genes with 1000 tree tips

Number of Genes / Loss Type	5	10	20	40
RMSE	0.20	0.46	0.65	0.75

From Table 5.2, we can see that when the number of genes increases, the RMSE also increases a bit. This task is very challenging, as the number of output nodes increases rapidly with the number of genes. For example, 5 genes data profile has 10 outputs, 10 genes data has 45 outputs, 20 genes data has 190 outputs, and 40 genes data has 780 outputs. There are many more nodes when we increase the number of genes. It puts tremendous pressure on the model to estimate the rates for all the gene pairs. It is very difficult to learn the model links' weights when the number of output nodes is large. We are continuing to investigate the model performance with more genes currently.

Chapter 6

Conclusion

In our study, we propose a new model to study the interaction relationships between genes using Deep Regression Convolutional Neural Networks on Gene's Phylogenetic Profiles by converting the Genes' Profiles as an image that contains the genes' presentation and it will automatically embed the tree structure into the image. We demonstrate that the CNN model is immune to changes in the phylogenetic tree structure for different gene groups with varying tree structures. For the model performance, the CNN model shows slightly better performance than the CCM model of different types of 5 Genes's interaction network structures with smaller standard errors and smaller bias of phylogenetic rates. By using the Hypothesis Test to reveal the interaction types, the CNN model shows a slightly better ROC curve than the CCM model. We also found that using Benjamini-Hochberg Procedure can control the False Discovery Rate of the CNN model's prediction with a trade-off between the power curve's performance. For the CNN model structure itself, we test and show that the extra linear layers after flattening the Convolutional layers can predict the Genes' Phylogenetic rates directly for the needs of the hypothesis test. In the code base, we used a vanilla CNN model with 3 Convolutional layers and 3 Linear layers to predict the Phylogenetic rates of the Genes. We also modified the ResNet model to fit the Regression tasks and trained with ResNet-18, ResNet-34, ResNet-50, and ResNet-152 on Compute Canadas' Niagara and Mist Cluster with up to 4 Nvidia V100 GPUs. We found that ResNet-34 and 50 have the best training time and model performance leverage. ResNet shows better performance and the data downsampling and bottleneck structures help to make the model deeper but with fewer parameters. The CNN model manages to solve 2000 tree leaves with 5 genes phylogenetic profile with Mean Square Error of 0.031 with Root Mean Square Errors of 0.1760 for 5 different genes co-interaction relationships. It has much less inference time than the CCM model. In this study, we create a proper code base that uses the data simulation method from the previous study of the CCM model to generate up to 8 Million genes' profiles and rates for different numbers of genes and tree leaves by settings. We also created a Python package code to convert the genes profiles to a proper image representation and train the vanilla CNN or ResNet-n model by 1-click running. We managed to create the whole pipeline to simulate, convert, and train the CNN model based on Genes' Profile with the user's preferences. The code base will be available in the GitHub repository soon. For future research, we will continue to find a way to scale up the model to predict the interaction relationships between more genes. We hope this study can contribute to understanding genes interaction types between genes pairs using the CNN model with faster inference speed and a larger number of genes profiles.

Bibliography

[Agarap, 2019] Agarap, A. F. (2019). Deep learning using rectified linear units (relu).

[Benjamini and Hochberg, 1995] Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300.

[Borisov et al., 2024] Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. (2024). Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21.

[Bowers et al., 2004] Bowers, P. M., Pellegrini, M., Thompson, M. J., Fierro, J., Yeates, T. O., and Eisenberg, D. (2004). Prolinks: a database of protein functional linkages derived from coevolution. *Genome Biol.*, 5(5):R35.

[Dawson et al., 2021] Dawson, S. K., Carmona, C. P., González-Suárez, M., Jönsson, M., Chichorro, F., Mallen-Cooper, M., Melero, Y., Moor, H., Simaika, J. P., and Duthie, A. B. (2021). The traits of "trait ecologists": An analysis of the use of trait and functional trait terminology. *Ecol. Evol.*, 11(23):16434–16445.

[Felsenstein, 1973] Felsenstein, J. (1973). Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters. *Systematic Biology*, 22(3):240–249.

[Fraser et al., 2004] Fraser, H. B., Hirsh, A. E., Wall, D. P., and Eisen, M. B. (2004). Coevolution of gene expression among interacting proteins. *Proc. Natl. Acad. Sci. U. S. A.*, 101(24):9033–9038.

[Gorishniy et al., 2023] Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. (2023). Revisiting deep learning models for tabular data.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Kumar et al., 2023] Kumar, S., Guruparan, D., Aaron, P., Telajan, P., Mahadevan, K., Davagandhi, D., and Yue, O. X. (2023). Deep learning in computational biology: Advancements, challenges, and future outlook.

[Liu et al., 2022] Liu, C., Kenney, T., Beiko, R. G., and Gu, H. (2022). The Community Coevolution Model with Application to the Study of Evolutionary Relationships between Genes Based on Phylogenetic Profiles. *Systematic Biology*, 72(3):559–574.

[Loken et al., 2010] Loken, C., Gruner, D., Groer, L., Peltier, R., Bunn, N., Craig, M., Henriques, T., Dempsey, J., Yu, C.-H., Chen, J., Dursi, L. J., Chong, J., Northrup, S., Pinto, J., Knecht, N., and Zon, R. V. (2010). Scinet: Lessons learned from building a power-efficient top-20 system and data centre. *Journal of Physics: Conference Series*, 256(1):012026.

[Niu et al., 2017] Niu, Y., Moghimyfiroozabad, S., Safaie, S., Yang, Y., Jonas, E. A., and Alavian, K. N. (2017). Phylogenetic profiling of mitochondrial proteins and integration analysis of bacterial transcription units suggest evolution of F1Fo ATP synthase from multiple modules. *J. Mol. Evol.*, 85(5-6):219–233.

[O'Shea and Nash, 2015] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks.

[Paradis et al., 2004] Paradis, E., Claude, J., and Strimmer, K. (2004). Ape: analyses of phylogenetics and evolution in r language. *Bioinformatics*, 20(2):289–290.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[Pellegrini, 2012] Pellegrini, M. (2012). Using phylogenetic profiles to predict functional relationships. In *Bacterial Molecular Networks*, Methods in molecular biology (Clifton, N.J.), pages 167–177. Springer New York, New York, NY.

[Ponce et al., 2019] Ponce, M., van Zon, R., Northrup, S., Gruner, D., Chen, J., Ertinaz, F., Fedoseev, A., Groer, L., Mao, F., Mundim, B. C., Nolta, M., Pinto, J., Saldarriaga, M., Slavnic, V., Spence, E., Yu, C.-H., and Peltier, W. R. (2019). Deploying a top-100 supercomputer for large parallel workloads: the niagara supercomputer. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC '19, New York, NY, USA. Association for Computing Machinery.

[R Core Team, 2024] R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Rouam, 2013] Rouam, S. (2013). *False Discovery Rate (FDR)*, pages 731–732. Springer New York, New York, NY.

[Simon et al., 2023] Simon, J. B., Karkada, D., Ghosh, N., and Belkin, M. (2023). More is better in modern machine learning: when infinite overparameterization is optimal and overfitting is obligatory.