HYPERPARAMETER PREDICTION IN ANN AND RNN: A CASE STUDY IN HOTEL DOMAIN

Yasantha Subash Samarasekara

(199484N)

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

April 2024

HYPERPARAMETER PREDICTION IN ANN AND RNN: A CASE STUDY IN HOTEL DOMAIN

Wijesooriya Ralalage Yasantha Subash Samarasekara

(199484N)

Thesis/Dissertation submitted in partial fulfilment of the requirements for the degree Master of Science

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

April 2024

Declaration

I declare that this is my work, and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or Diploma in any other University or institute of higher learning. To the best of my knowledge and belief, it does not contain any material previously published or written by another person except where acknowledgement is made in the text.

Also, I hereby grant the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part, in print, electronic, or other mediums. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

…… ……09/05/2024……
Mr. W.R.Y.S. Samarasekara

The above candidate has carried out research for the Master thesis/dissertation under my supervision.

Signature of the supervisor: Date:

19/08/2024

…… ……
Dr. A.T.P. Silva

Abstract

Deep Neural Networks are a subfield of the subsymbolic paradigm of Artificial Intelligence. Usually, when one wants to use artificial neural networks for a specific task, the neural network should be configured with optimal hyperparameters. To conduct such a task, the user should know about neural networks, as finding the optimal set of hyperparameters is time-consuming when manually configuring. This matter is prominent in data science because the field's requirements are directed toward deep learning and LLMs with autoML platforms. Automated hyperparameter prediction and optimization results in knowing about configuration and turning neural networks entirely or partially eliminated.

This research starts its journey to achieve hyperparameter prediction in a naval way by considering the selection of independent and dependent features. Hyperparameter generation happens by using neural networks and training 34 predictors and classifiers. Mapping between different feature sets and the existing model configurations, achieved by a naval general frame, has been introduced, using natural language processing to arrange feature names in similar columns. This nominal feature set is so diverse in such a way that it has 2-3 100% identical feature names per 40 feature name rows. So, naval natural language processing encoding was introduced as a solution. When the complete feature set is inserted into the model, the set of classifiers and the predictors achieve the configuration in RNN and ANN—the solution given the name HPPGeneral model. Then, the resultant configurations are applied to select problems in the hotel domain to evaluate the approach. This research has delivered several naval research outputs, such as general frame and natural language encoding, besides a model that can be used for both ANN and RNN. Finally, the hyperparameter prediction achieved by this approach also gives results almost similar to those of manual hyperparameter prediction.

Keywords: hyperparameter prediction, ANN, RNN, general frame, NLP encoding, general model

Dedication

To my lovely family

Acknowledgement

First of all, I’m so grateful to my supervisor, Dr. Thushari, for her kind guidance and demonstration of support. Without her, this research would never have been possible. Also, thank you for the kind guidance provided by the professor. Karunananda, through the comments on the evaluations. Because of it, I got the opportunity to excel my research experience in Artificial General Intelligence and hyperparameter prediction. Because of them, I found this opportunity to do such beautiful research. Also, I am grateful for the comments the course coordinator and evaluators provided to make this research and the thesis a success.

Additionally, I would be grateful to my friend Dr. Haishchndra, an engineering faculty member at the University of Peradeniya, for his guidance. This type of research topic is not easy to do, and research has been selected due to the leave granted by the Sabaragamuwa University of Sri Lanka. It is a great pleasure to work with them.

The support given by my family, especially my wife and her mother, in tackling and bearing difficult situations greatly supported me in this research. Also, my two children, even at their more minor ages, who least disturbed me, allowed me to focus on the research while working at home. Last but not least, I sincerely thank anyone who helped me enormously to make this research successful.

Table of contents

Declaration iii

Abstract iv

Dedication v

Acknowledgement vi

Table of contents vii

List of Figures xiii

List of tables xxii

List of abbreviations xxiii

CHAPTER 1 INTRODUCTION 1

1.1 Prolegomena 1

1.2 Aim and Objectives 2

1.3 Background and Motivation 3

1.4 Problem definition 5

1.5 General model 6

1.5.1 Input 6

1.5.2 Output 7

1.5.3 Process 7

1.5.4 Users 7

1.5.5 Features 8

1.6 Resource requirements 8

1.7 Structure of the Thesis 8

1.8 Summary 8

CHAPTER 2 DEVELOPMENT AND ISSUES 9

2.1 Introduction 9

2.2 Early developments in ANN, RNN, and hyperparameter Research with hotel classification and review prediction 9
2.3 hyperparameter prediction, optimization in neural networks 13
2.4 New Trends in hyperparameter prediction and optimization 23
2.5 Analysis of Literature 26
2.6 Problem definition 38
2.7 Summary 39
CHAPTER 3 TECHNOLOGIES ADOPTED 40
3.1 Introduction 40
3.2 Panda data frame 40
3.3 TensorFlow 41
3.4 Mathematical Model of ANN 43
3.4.1 Mathematical model of multilayer ANN 44
3.4.1.1 Backpropagation algorithm 47
3.4.1.2 Other learning rules 49
3.5 Long Short-Term Memory 49
3.6 Dataset building with existing problems. 52
3.7 Application programming interface 54
3.8 Summary 54
CHAPTER 4 APPROACH 55
4.1 Introduction 55
4.2 Hypothesis 56
4.3 Input 56
4.4 Output 56
4.5 Process 56
4.6 Users 57

4.7 Features 58

4.8 Summary 58

CHAPTER 5 DESIGN 59

5.1 Introduction 59

5.2 Module 01 & Module 02: Preprocessing 60

5.3 Module 03: ANN-RNN Classifier 60

5.3.1 General frame 60

5.3.2 Encoding general data frame 64

5.3.3 NLP Encoding 65

5.3.3.1 NLP encoding method 01 65

5.3.3.2 NLP encoding method 02 65

5.3.4 Label encoding with random fit 65

5.3.5 The neural network for ANN-RNN classifier 66

5.4 Module 04: Hyperparameter predictor module 66

5.4.1 Layer Number Predictor 67

5.4.2 Optimizer classifier 68

5.4.3 Lost function classifier 68

5.4.4 Lost metrix classifier 68

5.4.5 Number of epochs predictor 68

5.4.6 Batch size predictor 68

5.4.7 Layer type classifier 69

5.4.8 Number of neurons predictor 69

5.4.9 Dropout predictor 69

5.4.10 Activation function classifier 70

5.4.11 Embeding Layer-Input dimension predictor 70

5.4.12 Embeding Layer -Output dimention predictor 70

5.5 Module 05: General module 70
5.5.1 Create a general model 71
5.5.2 ANN & RNN Layer Predictors 71
5.5.3 ANN & RNN layer restrictors 72
5.5.4 ANN & RNN layer creators 72
5.5.5 Configuration loader 72
5.5.6 Model runner 72
5.5.7 Evaluator and Predictor 72
5.5.8 User process for HPPGeneral model 73
5.6 Summary 74
CHAPTER 6 IMPLEMENTATION 75
6.1 Introduction 75
6.2 Module 01 & Module 02: Preprocessing 75
6.3 Module 03: ANN-RNN Classifier 76
6.3.1 General frame 76
6.3.1.1 Sequence Row 77
6.3.1.2 Transfer dependent variables into the dependent section(dependent variables) 82
6.3.2 Encoding general data frame 85
6.3.2.1 NLP encoding method 01 86
6.3.2.2 NLP encoding method 02 90
6.3.2.3 Label encoding with random fit 92
6.3.3 The neural network 95
6.4 Module 05: Hyperparameter prediction module 97
6.4.1 Layer Number Predictor 97
6.4.2 Optimizer classifier 98

6.4.3 Lost function classifier 100
6.4.4 Lost metrix classifier 102
6.4.5 Number of epochs predictor 104
6.4.6 Batch size predictor 106
6.4.7 Layer type classifier 107
6.4.7.1 Layer type classifier for layer 01 108
6.4.7.2 Layer type classifier for layer 02 109
6.4.7.3 Layer type classifier for layer 03 112
6.4.7.4 Layer type classifier for layer 04 113
6.4.7.5 Layer type classifier for layer 05 115
6.4.7.6 Layer type classifier for layer 06 117
6.4.8 Number of neurons predictor 119
6.4.9 Dropout predictor 126
6.4.10 Activation function classifier 134
6.4.10.1 Activation function classifier for layer 01 135
6.4.10.2 Activation function classifier for layer 02 137
6.4.10.3 Activation function classifier for layer 03 139
6.4.10.4 Activation function classifier for layer 04 141
6.4.10.5 Activation function classifier for layer 05 142
6.4.10.6 Activation function classifier for layer 06 144
6.4.11 Embedding Layer - Input dimension predictor 146
6.4.12 Embedding Layer - Output dimension predictor 147
6.4.13 Final Hyperparameter Predictor Module 148
6.5 Module 05: General module 149
6.5.1 Create a general model 149
6.5.2 ANN & RNN Layer Predictors 150

6.5.2.1 ANN Layer Predictor 150
6.5.2.2 RNN Layer Predictor 151
6.5.3 ANN & RNN layer restrictors 153
6.5.3.1 ANN layer restrictors 153
6.5.3.2 RNN layer restrictors 154
6.5.4 ANN & RNN layer creators 155
6.5.5 Configuration loader 157
6.5.6 Model runner 160
6.5.7 Evaluator and Predictor 163
6.5.8 User process for HPPGeneral model 163
6.6 Summary 168
CHAPTER 7 EVALUATION 169
7.1 Introduction 169
7.2 Training results comparison table 170
7.3 Summary 175
CHAPTER 8 CONCLUSIONS AND RECOMMENDATIONS 176
8.1 Introduction 176
8.2 Other Important Research Outputs 176
8.3 Recommendations 177
8.4 Summary 178
References 179
Appendices 188

List of Figures

Figure 2.1 ANN network architectures 10

Figure 2.2 Quad—arrangement of nodes as a basic representational unit 22

Figure 3.1 Architecture of a typical vanilla LSTM block 50

Figure 5.1 Overview of the Design diagram of the general model 59

Figure 5.2 Row adding process to the general frame. 61

Figure 5.3 Process of adding feature row to the general frame. 62

Figure 5.4 Transfer dependent variable from independent side to dependent side 63

Figure 5.5 Encorder class 64

Figure 5.6 General Frame 65

Figure 5.7 Training and prediction process of the classifier 66

Figure 5.8 Structure of the hyperparameter predictor 67

Figure 5.9 General module class 71

Figure 5.10 General Model Creation 73

Figure 6.1 Converting camelCase feature with spaces with lowercase 75

Figure 6.2 Code to introduce lowercase with spaces 76

Figure 6.3 Code for creating general frame structure. 76

Figure 6.4 Code for adding the first row directly otherwise sequence the items. 77

Figure 6.5 Code to add to the general frame. 77

Figure 6.6 Code to compare the maximum comparison value 78

Figure 6.7 The output created by the code. 78

Figure 6.8 Code for the action when the value has already existed 79

Figure 6.9 Output when a higher comparison value is found 79

Figure 6.10 Code for checking candidate values to consider for a shift 80

Figure 6.11 Visual output when shifting occurs. 80

Figure 6.12 Code for adding the item to the sequence list by available list. 81

Figure 6.13 Visual output from removing an item and adding to the sequence list. 81
Figure 6.14 Code for rearranging the feature items in the order 81
Figure 6.15 Output of the rearranged list according to the order 82
Figure 6.16 Appearance of the frame after adding 15 rows. 82
Figure 6.17 General frame with empty cells in dependent variables side 82
Figure 6.18 Code to find the first empty row in the dependent side 83
Figure 6.19 Set of codes for transferring the dependent variables 84
Figure 6.20 Output after transferring dependent variables to the dependent side. 84
Figure 6.21 Code for saving the output as CSV or loading it if it already existed. 84
Figure 6.22 Add column method to add a column to the general frame. 85
Figure 6.23 Add one or more columns to the general frame at once. 85
Figure 6.24 Final general frame after generating all the columns. 85
Figure 6.25 Code to check whether a frame is saved as a CSV. 86
Figure 6.26 Output after encoding for the independent side by NLP encoding 01 86
Figure 6.27 Code sees whether the general frame is already saved as a CSV 87
Figure 6.28 The code compares the values with an empty string 87
Figure 6.29 Output after encoding the dependent side. 88
Figure 6.30 After encoding the Nature of the neural network column 88
Figure 6.31 Correlated matrix for the NLP encoding method 01 89
Figure 6.32 Output of the neural network trained: encoded values(the lost) 89
Figure 6.33 Output of the neural network trained: encoded values(the accuracy) 90
Figure 6.34 Output for the second time training of the neural network. 90
Figure 6.35 This is the code for NLP encoding method 02. 90
Figure 6.36 Encoding output for NLP method 02. 91
Figure 6.37 This is the correlation matrix for the NLP encoding method 02. 91
Figure 6.38 The output for the training by NLP method 02. 92

Figure 6.39 Accuracy of the neural network after training: encoding method 02. 92
Figure 6.40 This is the code for the label encoding for the entire table. 93
Figure 6.41 Output after applying label encoding with the random fit. 93
Figure 6.42 This is the correlation matrix after label encoding. 94
Figure 6.43 Output: neural network after applying encoded values for the training 94
Figure 6.44 This is the accuracy after the training by label encoding. 95
Figure 6.45 Prediction accuracy for the NLP encoding method01. 95
Figure 6.46 Predicting accuracy for the label encoding. 95
Figure 6.47 Value transfer through NumPy and min-max Scaler. 96
Figure 6.48 Traning testing split by sklearn. 96
Figure 6.49 Neural network configuration for the binary classifier. 96
Figure 6.50 Neuralnerwork configuration number of layers predictor 97
Figure 6.51 Loss function number of layers predictor 97
Figure 6.52 Actual value and predicted value of the number of layers 98
Figure 6.53 Neural network for optimizer classifier 98
Figure 6.54 Lost function for optimizer classifier 99
Figure 6.55 Accuracy of optimizer classifier 99
Figure 6.56 Evaluation and prediction of optimizer classifier 100
Figure 6.57 Predicted values of optimizer classifier 100
Figure 6.58 Neural network for lost function classifier 101
Figure 6.59 Lost function for lost function classifier 101
Figure 6.60 Accuracy of lost function classifier 101
Figure 6.61 Evaluation and Prediction of Lost Function Classifier 102
Figure 6.62 Predicted values of lost function classifier 102
Figure 6.63 Neural network for matrix classifier 102
Figure 6.64 Lost function for matrix classifier 103

Figure 6.65 Accuracy of matrix Classifier 103

Figure 6.66 Evaluation and prediction of matrix classifier 103

Figure 6.67 Predicted values of matrix classifier 104

Figure 6.68 Neural network for epochs predictor 104

Figure 6.69 Lost function for epochs predictor 105

Figure 6.70 Evaluation and prediction of epochs Predictor 105

Figure 6.71 Predicted values of epochs predictor 105

Figure 6.72 Neural network for batch size Predictor 106

Figure 6.73 Lost function for batch size predictor 106

Figure 6.74 Evaluation and prediction of batch size Predictor 107

Figure 6.75 Predicted values of batch size predictor 107

Figure 6.76 Neural network for layer type classifier for layer one 108

Figure 6.77 Lost function for layer type classifier for layer one 108

Figure 6.78 Accuracy of layer type classifier for layer one 108

Figure 6.79 Evaluation of layer type classifier for layer one 109

Figure 6.80 Predicted values of layer type classifier for layer one 109

Figure 6.81 Neural network for layer type classifier for layer two 109

Figure 6.82 Lost function for layer type classifier for layer two 110

Figure 6.83 Accuracy of layer type classifier for layer two 110

Figure 6.84 Evaluation and prediction of layer type classifier for layer two 111

Figure 6.85 Predicted values of layer type classifier for layer two 111

Figure 6.86 Neural network for layer type classifier for layer three 112

Figure 6.87 Lost function for layer type classifier for layer three 112

Figure 6.88 Accuracy of layer type classifier for layer three 112

Figure 6.89 Evaluation and prediction of layer type classifier for layer three 113

Figure 6.90 Predicted values of layer type classifier for layer three 113

Figure 6.91 Neural network for layer type classifier for layer four 113
Figure 6.92 Lost function for layer type classifier for layer four 114
Figure 6.93 Accuracy of layer type classifier for layer four 114
Figure 6.94 Evaluation and prediction of layer type classifier for layer four 115
Figure 6.95 Predicted values of layer type classifier for layer four 115
Figure 6.96 Neural network for layer type classifier for layer five 115
Figure 6.97 Lost function for layer type classifier for layer five 116
Figure 6.98 Accuracy of layer type classifier for layer five 116
Figure 6.99 Evaluation and prediction of layer type classifier for layer five 116
Figure 6.100 Predicted values of layer type classifier for layer five 117
Figure 6.101 Neural network for layer type classifier for layer six 117
Figure 6.102 Lost function for layer type classifier for layer six 117
Figure 6.103 Accuracy of layer type classifier for layer six 118
Figure 6.104 Evaluation and prediction of layer type classifier for layer six 118
Figure 6.105 Predicted values of layer type classifier for layer six 118
Figure 6.106 Neural network for number of neurons predictor for layer one 119
Figure 6.107 Lost function for the number of neurons predictor for layer one 119
Figure 6.108 Evaluation and prediction, prediction: No of neuron predictor L one 120
Figure 6.109 Neural network for number of neurons predictor for layer two 120
Figure 6.110 Lost function for number of neurons predictor for layer two 120
Figure 6.111 Evaluation & prediction, prediction: No of neuron predictors L-two 121
Figure 6.112 Neural network for number of neurons predictor for layer three 121
Figure 6.113 Lost function for number of neurons predictor for layer three 122
Figure 6.114 Evaluation & prediction, prediction: No of neuron predictor L-three 122
Figure 6.115 Neural network for number of neurons predictor for layer four 123
Figure 6.116 Lost function for number of neurons predictor for layer four 123

Figure 6.117 Evaluation & prediction, prediction: No of neuron predictors L-four 123
Figure 6.118 Neural network for number of neurons predictor for layer five 124
Figure 6.119 Lost function for number of neurons predictor for layer five 124
Figure 6.120 Evaluation & prediction, prediction: No of neuron predictors L-five 125
Figure 6.121 Neural network for number of neurons predictor for layer six 125
Figure 6.122 Lost function for the number of neurons predictor for layer six 125
Figure 6.123 Evaluation & prediction, prediction: No of neuron predictors L-six 126
Figure 6.124 Neural network for dropout predictor for layer one 126
Figure 6.125 Lost function for dropout predictor for layer one 127
Figure 6.126 Evaluation & prediction, prediction: No of dropout predictor L-one 127
Figure 6.127 Neural network for dropout predictor for layer two 128
Figure 6.128 Lost function for dropout predictor for layer two 128
Figure 6.129 Evaluation & prediction, prediction: No of dropout predictor L-two 129
Figure 6.130 Neural network for dropout predictor for layer three 129
Figure 6.131 Lost function for dropout predictor for layer three 130
Figure 6.132 Evaluation & prediction, prediction: No of dropout predic. L-three 130
Figure 6.133 Neural network for dropout predictor for layer four 131
Figure 6.134 Lost function for dropout predictor for layer four 131
Figure 6.135 Evaluation & prediction, prediction: No of dropout predictor L-four 132
Figure 6.136 Neural network for dropout predictor for layer five 132
Figure 6.137 Lost function for dropout predictor for layer five 132
Figure 6.138 Evaluation & prediction, prediction: No of dropout predictor L-five 133
Figure 6.139 Neural network for dropout predictor for layer six 133
Figure 6.140 Lost function for dropout predictor for layer six 134
Figure 6.141 Evaluation & prediction, prediction: dropout predictor for layer six 134
Figure 6.142 Neural network for activation function classifier for layer one 135

Figure 6.143 Lost function for activation function classifier for layer one 135
Figure 6.144 Accuracy of activation function classifier for layer one 135
Figure 6.145 Evaluation & prediction: activation function classifier for layer one 136
Figure 6.146 Predicted values of activation function classifier for layer one 136
Figure 6.147 Neural network for activation function classifier for layer two 137
Figure 6.148 Lost function for activation function classifier for layer two 137
Figure 6.149 Accuracy of activation function classifier for layer two 138
Figure 6.150 Evaluation & prediction of activation function classifier L-two 138
Figure 6.151 Predicted values of activation function classifier for layer two 138
Figure 6.152 Neural network for activation function classifier for layer three 139
Figure 6.153 Lost function for activation function classifier for layer three 139
Figure 6.154 Accuracy of activation function classifier for layer three 140
Figure 6.155 Evaluation & prediction of activation function classifier L-three 140
Figure 6.156 Predicted values of activation function classifier for layer three 140
Figure 6.157 Neural network for activation function classifier for layer four 141
Figure 6.158 Lost function for activation function classifier for layer four 141
Figure 6.159 Accuracy of activation function classifier for layer four 141
Figure 6.160 Evaluation & prediction of activation function classifier L-four 142
Figure 6.161 Predicted values of activation function classifier for layer four 142
Figure 6.162 Neural network for activation function classifier for layer five 142
Figure 6.163 Lost function for activation function classifier for layer five 143
Figure 6.164 Accuracy of activation function classifier for layer five 143
Figure 6.165 Evaluation & prediction of activation function classifier L-five 143
Figure 6.166 Predicted values of activation function classifier for layer five 144
Figure 6.167 Neural network for activation function classifier for layer six 144
Figure 6.168 Lost function for activation function classifier for layer six 144

Figure 6.169 Accuracy of activation function classifier for layer six 145
Figure 6.170 Evaluation & prediction of activation function classifier L-six 145
Figure 6.171 Predicted values of activation function classifier for layer six 146
Figure 6.172 Neural network for input dimension of the embedding layer 146
Figure 6.173 Lost function for input dimension of the embedding layer 146
Figure 6.174 Evaluation & prediction of the input dimension of the embed. layer 147
Figure 6.175 Neural network for output dimension of the embedding layer 147
Figure 6.176 Lost function for output dimension of the embedding layer 148
Figure 6.177 Evaluation & prediction, prediction val. of output dim. of embed. 148
Figure 6.178 The hyperparameter prediction class 149
Figure 6.179 Create General Model 149
Figure 6.180 RNN part of the create general model 150
Figure 6.181 Starting part of the ANN layer predictor 150
Figure 6.182 Code for loading diff. predi. when anomalies happen 151
Figure 6.183 Starting code of the RNN layer predictor 152
Figure 6.184 RNN loads other predi. when anomalies in the before-fix value 153
Figure 6.185 ANN model restrictor, layer restrictor 154
Figure 6.186 RNN model restrictor, layer restrictor 155
Figure 6.187 ANN layer creator method 156
Figure 6.188 RNN layer creator method 156
Figure 6.189 Starting of configuration loader method 157
Figure 6.190 ANN code part of the configuration loader method 158
Figure 6.191 RNN code part of the configuration loader 159
Figure 6.192 RNN code part of the config. Loader including diff. input shapes 160
Figure 6.193 ANN part of the model runner 161
Figure 6.194 Print is used to give the code output for the user in the ANN part 161

Figure 6.195 Similar RNN restrictors usage in the RNN part of the model loader 162
Figure 6.196 Simler data conversion and print usage in the RNN: model runner 163
Figure 6.197 Implementation of evaluator and predictor methods 163
Figure 6.198 Generate general frames and apply encoding by the user 164
Figure 6.199 Creating the general model: run the hyperparameter prediction 164
Figure 6.200 Generated ANN general frame output 165
Figure 6.201Generated RNN general frame output 165
Figure 6.202 Configuration loader usage by the user for a regression problem 166
Figure 6.203 Configuration loader usage for text classification 166
Figure 6.204 Model runner method usage by the user for a regression problem 166
Figure 6.205 Model runner method usage for a text classification problem 167
Figure 6.206 Usage of evaluate and predict method by the user 167
Figure 7.1 Evaluating a regression problem for the HPPGeneral Model 171
Figure 7.2 Result of the HPPGeneral model for a regression problem 171
Figure 7.3 Evaluating the AutoML for a regression problem in deep learning 171
Figure 7.4 Result of the AutoML for a regression problem 172
Figure 7.5 Evaluating HPPGegeral model for a classification problem 172
Figure 7.6 Result of the HPPGeneral model for a classification problem 172
Figure 7.7 Evaluating AutoML for a classification problem in deep learning 173
Figure 7.8 Result of AutoML for the classification problem in deep learning 173
Figure 7.9 Comparion the HPPGeneral model with AutoML 175

List of tables

Table 2.1 Table of analysis of the literature 26

Table 3.1 Datasets considered for adding features to the general frame 52

Table 5.1 General Frame Structure 61

Table 7.1 Problem table for testing the solution 169

Table 7.3 Comparison between Expert, AutoML and HPPGeneral Model 174

List of abbreviations

AI artificial intelligence

ANN artificial neural network

CNN Convolutional neural network

DAN Deep averaging network

DBN Deep belief networks

GNN Graph neural networks

HR High resolution

HTM Hierarchical temporal memory, xliii; Hierarchical Temporal Memory

LLMs Large Language Models

LSTM Long Short-Term Memory

LSTM long short-term memory

MLP Multiplayer perceptron

NeSyL Neuro-symbolic learning

NLP Natural language processing

RAAM Recursive auto-associative memory

RAN Residual attention network

RBM Restricted Boltzmann machine

RNN Recurrent neural network

RvNN Recursive neural networks

SOM Self-organizing map

CHAPTER 1

INTRODUCTION

This chapter gives an overview of the entire thesis, especially the aims and the objectives, a brief explanation of the problem approach and the solution, and an evaluation and conclusion.

1.1 Prolegomena

When considering data-related jobs such as Data Analyst, Data Engineer, Database Administrator, Machine Learning Engineer, Data Scientist, Data Architect, Statistician, Business Analyst, and Data and Analytics Manager, the majority of the jobs require knowledge of deep learning because deep learning is the expert of dealing with big data that is unstructured too. This matter is also emphasized by the Data Science Skills Survey Report 2024 [1]. It stated that 76.3% of the recruiters say it is hard to find qualified candidates with the right skills. Automating data science tasks will cut off the human resource requirement, which is one of the possible solutions. Further, the survey identified that AutoML platforms are the 3rd from Future trend that will impact the data science skillset.

Moreover, it identified Deep Learning and Large Language Models (LLMs) as one of the 4 Tools and Technologies to Prepare for the Future. This fact confirms that ChatGPT [2], Like software, will perform deep learning tasks by taking the data with automation in the future, making time and money used for upskilling in deep learning a waste. So this enables ChatGPT, like AGI systems, to perform deep learning-related analyses independently. So, professionals who don’t have deep learning knowledge do not need to upskill their neural network and deep learning knowledge, which will also save them the cost of organizing training programs.

The inspiration for Artificial Neural Networks [3] comes from the Neurons found in the animal's brain. So, the Artificial neuron has almost the same features as the natural neuron. Humans create artificial neurons through a programming language followed by an algorithm [3]. Various types of artificial neurons are designed in a layered architecture with human intervention.

There are many types of neurons, such as CNN [4], GNN [5], and RNN [6], which is based on ANN. The general problem is predicting different types of neural network models and hyperparameters according to the feature set. The model and the hyperparameters between ANN and RNN must be predicted as the specific problem of the research. So, the solution has created a general model by creating a predictive class by employing 34 classifiers and predictors, including the ANN-RNN model selector according to the selected feature set. Moreover, because it can't measure the performance of all the problems, it restricts the issues to the hotel domain (hotel classification and review rating prediction). The evaluation of the solution has been tested with more than five problems with datasets. Finally, it can be concluded that hyperparameter prediction through feature selection is another good approach to hyperparameter prediction and model selection that can be used for the same dataset with different problems. Further, this only provides a solution to some extent because of the low amount of training data available. The rest of this chapter will elaborate on the aims and objectives of this research, background, and motivation followed by the problem definition and the solution extends to input, output, process, and then user features and resource requirements ending with the structure of the thesis.

1.2 Aim and Objectives

This research project aims to develop a hyperparameter prediction and model selection system between ANN and RNN through feature selection for the hotel domain as a research challenge.

Objective 01: A critical review of existing hyperparameter prediction and optimization approaches extending to hotel classification & review rating classification.

Objective 02: In-depth study of the latest research on customized neural networks and ANN, RNN-based Technologies

Objective 03: Design and develop a neural network-based system that automatically selects neural network types for ANN and RNN with hyperparameter prediction for the hotel domain.

Objective 04: Evaluate the Solution to Hotel domain problems with datasets.

Objective 05: Preparation of final documentation

1.3 Background and Motivation

When learning the technologies of ANN, it is a fact to know that ANN is inadequate for sequential problem input and problem-solving. So, there are RNNs to handle that kind of problem. However, in these cases, the user must explicitly choose whether to apply ANN or RNN when considering whether there is a possibility of using the relevant neural network type with optimal hyperparameters automatically,

The hyperparameter prediction on the neural network starts initially with one group of researchers successfully attempting it through the creation of an auto data scientist (ADS) [7]. In this research, the researchers have used a binary classification data set repository consisting of meta-features and optimal hyper-parameter by applying a Bayesian optimizer to generate hyper-parameter prediction models [7]. Furthermore, they increase the number of models with versatility by applying forward model selection ensembles [7]. Their primary concern is hyper-parameter optimization rather than prediction [7]. Both hyper-parameter prediction and optimization happen with the above procedure [7]. The preprocessing part used heuristic rules, including data splitting [7]. Later, yet another research was similarly done in hyperparameter tuning to achieve automated machine learning extending the hyperparameter tuning binary classification problems [8]. In this research, as per the previous one, they generated forward model ensembles with basic optimization, then per standard 124 datasets, they identified the differentiation of their meta-features, ranked the input dataset with a distance value for meta-features, and selected the best-performed 25 models directly for creating a model ensemble [8].

ANN and RNN both fall under the oldest types of neural networks. Artificial neural networks enable computation, generalization ability, learning ability, low Energy Consumption, distributed representation, inherent contextual information processing, massive parallelism, fault tolerance, and adaptability [9]. ANN is essentially divided into two types, namely feed-forwarded networks and feedback networks [9]. Moreover, the above two types of neural networks have many architectures. Linear, multilayer perceptron, RBF networks, CMAC classification only, and regression only are some variations of feed-forward network architectures. Among them, back

propagation, which comes under multilayer perceptron, is the most famous architecture [10]. Finding and correcting errors start from the output layer, and the same process happens backwards using the resultant error values found in each layer until all the weight values are adjusted in each iteration [10].

On the other hand, Binary association memory, Boltzmann machine, and recurrent time series are some initial architectures of feedback networks [9]. The most popular feedback network is long short-term memory, abbreviated LSTM, which performed far better than other feedback network architectures [11]. There are eight variants of the LSTM. Among them, vanilla LSTM is the most popular variant [11].

Even though there are many applications of neural networks, most are of standard ways rather than customized neural networks. Some approaches in artificial general intelligence (AGI) architectures give excellent insights into customized neural network applications. When it comes to AGI approaches for currently developed AGI systems, there are four major types of approaches: Symbolic AGI Approaches, Emergentist AGI Approaches (sub-symbolic), Hybrid AGI, and The Universalist Approach [12]. The focus here is to get some insights about applications of customized neural networks in AGI architects, which aligns with the attention of the architectures given in Emergentist AGI Approaches and Hybrid AGI.

DeSTIN is a convergent HTM and CogPrime architecture with additional complex learning mechanisms [12]. So, the attention goes to Hierarchical Temporal Memory (HTM). In hierarchical temporal memory, a competitive neural network has integrated into the spatial pooler process and as one of the modules in an HTM node arranged in a Hierarchical Bayesian network. SAL is another AGI architecture that blends the ACT-R architecture with neural network architecture called Leabra [13]. NOMAD is an AGI architecture based on natural selection based on Neural Darwinism, specifically evolving more stimulated neurons that carry out sensorimotor and categorization tasks [12]. In addition to the above architectures, Ben Kuipers and his colleagues have combined qualitative reasoning and reinforcement learning to master how to perceive, model, and act the world for intelligent agents. Tsvi Achler mapped weight values in sub-symbolic emergent architecture and symbolic rule-based architecture by changing weight values according to the gradient descendent

algorithm, and the created rules [14] would lead to a turning point in sub-symbolic architecture in future endeavors.

With the blend of symbolic and sub-symbolic architectures, the hybrid AGI approach was achieved to immerse ourselves in the sub-symbolic parts of hybrid AGI architectures. The hybrid architecture of CLARION has been achieved by blending symbolic components for the reasoning part by "explicit knowledge" and connectionist component for achieving "implicit knowledge" [12]. In the DUAL architecture, the agents have a mixture of symbolic and sub-symbolic mechanisms and computations. At the lowest level of DUAL, the most extensive collection of units is called DUAL agents [15]. Further, agents send messages through a network of weighted links that spread activations [15].

Further, it treats the above characteristics as a computational model [16]. The neuro-symbolic representation of the Psi architecture consists of hierarchical networks of neural elements [16]. The neural network consists of 4 types of neurons. Namely, the sub(has-part), sur(is-part), por(subjunctive), and ret (inverse relation to por) are connected with a central neuron, forming a building block of the neural network [16].

One way of achieving emergentist architecture is through ANN. In ANN, the past knowledge is weight values, which is the replication of the synaptic connections of the biological neural networks [9].

1.4 Problem definition

Automated hyperparameter selection is bound to hyperparameter optimization, too. There are three main approaches to hyperparameter prediction and optimization research. The earliest approach is basian optimization in the study area, which is more confined to meta-features for identifying the unique data set by the selected set of datasets and generating the modules with forward model ensemble with Bayesian optimization. Further, the resulting solution of this approach is that AutoML doesn't work for Windows OS. In addition, it should run up to 25 models due to the use of ensembling, creating a requirement for larger RAM. Considering the following approach, population-based algorithms spend a lot of time running the generated models to calculate the fitness values even though there are deviations, namely genetic algorithms, particle swarm optimization, and deferential evaluation. The last approach

is the neural network approach through large language models. Previous is the most promising approach, but because of its high resource usage, it has been implemented as server-centered architecture. As a result, the solutions from this approach do not work without internet access.

Moreover, much of the research involves the generalized machine learning approach, considering all the machine learning techniques. When there is no internet, the best approach will be basian optimization with meta-features because other BO approaches are even slower. So, the research problem is to address those drawbacks resulting from BO. There is inadequate research on hyperparameter selection in deep neural networks, specifically RNN and ANN configurations. So far, no study has been performed to select the model based on knowledge of feature selection. So, the problem definition is still not automated in the model selection between ANN and RNN and hyperparameter prediction through feature selection.

1.5 General model

As in the above problem, there are two tasks as alternatives to identify the data set by meta-features of the data coming with an alternative approach. It is hypnotized that ANN and RNN model selection can be automated through individual hyperparameter prediction and model selection through feature selection. The next task is to predict the hyperparameters based on the above alternative model selection approach. In this solution, the creation of a general frame using natural language processing has been done to overcome arranging features in a single database related to different problems. Then, the hyperparameter prediction was done by training ANN. There are 13 hyperparameters to be predicted. Namely, they are no_of_Layers, optimizer, lost_fun, metrics, no_of_epochs, batch_size, layer_type, no_of_nurons, dropouts, activation_fun, (default)inpDim, (default)outpDim and model selection between ANN and RNN.

1.5.1 Input

In input, the complete feature set is related to the problem of selecting the model between TensorFlow ANN or LSTM and the general model with 33 hyperparameter predictor models. Then, input the selected feature (Y) and the data(X) into the fit function for each model. Also, we should provide the Y feature before preprocessing.

1.5.2 Output

In the output, the general model will give the configured TensorFlow model according to the given complete feature set of the hotel domain. Then, as usual, the run result of the fitness fit function can be used with the history variable to plot the relevant graphs. And save the model.

1.5.3 Process

The input data will be arranged in a frame in a general frame consisting of an independent variable section, a dependent variable section, and network type ANN/RNN in a supervised dataset. This type of column is not there for the general model because the trained model predicts that. Data should be encoded. The Label and naval NLP encoding created by this research project can feed this data to the classifier. As a result of not many everyday data items being present in each column of the initial 60 columns, NLP encoding was used. Label encoding was used for the other columns. In each step, dependent and independent features are fed into the next hyperparameter predictor with the output of the previous hyperparameter predictor for predicting the next hyperparameter. The hyperparameter predictors such as no_of_Layers, optimizer, lost_fun, metrics, no_of_epochs, and batch_size have individual predictors while layer_type, no_of_nurons, dropouts, activation_fun have layer-wise predictors. According to the features set, TensorFlow models for each ANN or RNN model are generated with additional inputs such as the dependent feature set before preprocessing and text column before encoding for RNN. These are details discussed in the approach and design chapters. The predictors model is used for the training with predictors running hyperparameters such as batch size, number of epochs, optimizer, and matrix. Previous will finally produce the predictor and evaluator methods.

1.5.4 Users

It can be used to solve classification and prediction that supports the underlying TensorFlow way of coding. Use the general frame generator as an NLP classifier. NLP-based encoding (for data preprocessing) can be used with more diverse values in the columns.

1.5.5 Features

Can configure neural networks without worrying about network configuration. Easily change the text comparison method of the general frame generator. The classifier and general frame can handle up to 50 independent and ten dependent variables. Auto model configurations apply only to the above restrictions.

1.6 Resource requirements

The hardware resources CPU with i7 is recommended because it should run the underlying NLP comparisons in the general frame. The RAM should be at least 8GB, which is recommended because of NLP usage when creating the general frame. 6GB RAM is also manageable, but it will take more time. For that hard disk space, 5GB is enough regarding the software requirements. Python libraries, NumPy libraries, sklearn libraries, TensorFlow/Keres libraries, spacy_universal_sentence_encoder library, and Jupiter notebook with Anaconda will be needed.

1.7 Structure of the Thesis

This thesis is structured so that the introduction, as in chapter one, will briefly overview these abstracts, especially the research project's aim and objectives. Then, you will come across the literature review in Chapter 2. It will find the literature that referred to this project mainly related to hyperparameter prediction and tuning customized neural networks in artificial general intelligence and neural science, followed by ANN and RNN technologies. Then, chapter 3 discusses the technologies that contribute to solving the problem, especially the chapter that focuses on the coding resources and underlying mathematics of backpropagation and LSTM algorithms. Then, it will be exposed to the approach in Chapter 4, which discusses how it reaches the solution, strategy, hypothesis, input, and output process. Then, for the design chapter, chapter 5 describes the design of the solution. Then, the implementation discusses how the implementation happens, and the 8th chapter elaborates on how the solution can be tested. The last chapter describes the future works and the conclusion.

1.8 Summary

This chapter discusses an overview of the thesis and its aims and objectives. The next chapter, chapter 2, immerses ourselves deep into the literature.

CHAPTER 2

DEVELOPMENT AND ISSUES

2.1 Introduction

In the previous chapter, an overview of the research. Chapter 2 presents the overall picture of the thesis by defining the research problem and identifying the technology that can solve the problem. The literature review is divided into three sections: Early developments in ANN, RNN, and hyperparameter Research with hotel classification and review prediction. Then, customized neural networks in AGI architectures, hyperparameter prediction, and new trends in hyperparameter prediction. This chapter also summarizes the literature regarding their contribution, limitations, and the technology/methodology used. Finally, define the research problem.

The Artificial Intelligence (AI) field has a history of more than 70 years [1]. John McCarthy was given the name of the field at the Dartmouth conference. In general, the evaluation of the field of artificial intelligence can be divided into three major phases [17]. Narrow artificial intelligence is the first and most developed phase recognized [1]. Express this as the development of an AI system with one specific capability. Expert systems demonstrate how experts solve a problem, and artificial neural networks illustrate the process of information as a network of neurons in the human brain does and enable learning by past data or experiences. Aligning with that, some partial AGI systems like chatGPT will gain the capability of configuring a neural network itself and performing data analysis as a data scientist.

2.2 Early developments in ANN, RNN, and hyperparameter Research with hotel classification and review prediction

In 1943, Warren McCulloch and Walter Pitts introduced the concept of artificial neurons. Natural neurons have parts such as dendrites, which get the neuron signals from previous neurons, and the axon, which sends the collected signals according to the striking capacity [9]. This network can focus on recognizing patterns and designs [9]. The previous is similarly simulated in artificial neural networks (ANN), which gives advantages such as real-time operation, self-organization, adaptive learning, and fault tolerance [9]. ANN directs an alternative path to critical thinking other than

conventional computers [9]. While data is handled correspondingly as the human cerebrum does, it ensures a more generalized approach to creating machine arrangements [9]. ANN is a mathematical model or computational model that is utilized for adaptive control, image analysis, speech recognition…etc [9]. Learning happens in biological neurons through synaptic[9] connections in artificial neurons it happens through weight values. Natural neural networks in the human brain make possible adaptability, generalization ability, learning ability, low energy consumption, computation, distributed representation, inherent contextual information processing, fault tolerance, and massive parallelism [9].

There are many types of ANN-based network architectures based on various theories. The following diagram shows some of them.

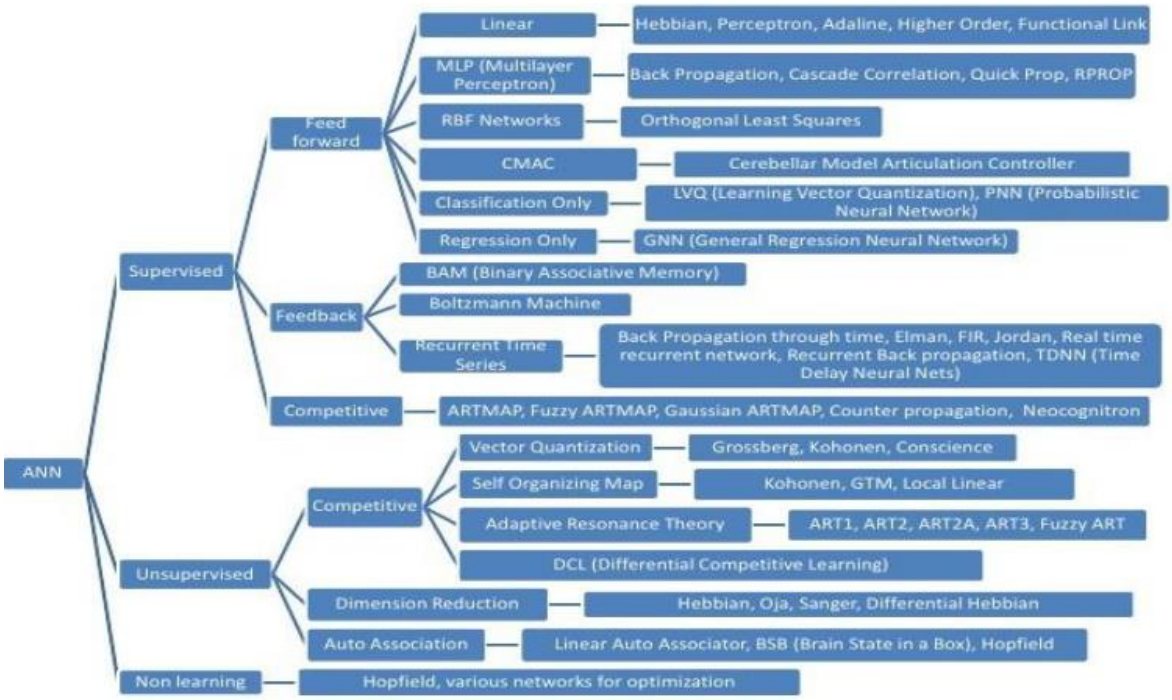


Figure 2.1:ANN network architectures

Source 2.1: Basic Application and Study of Artificial Neural Networks by Md. Tanjil Sarker,Sanjida Noor and prof.(Dr.) Uzzal Kumar Acharjee [9], page 6

ANN division for supervised, unsupervised, and non-learning has evolved in the above diagram 2.1 to supervised, unsupervised, and reinforcement learning with the immersion of the reinforcement learning [3]. Further, more popular ANN types have appeared, and some are extensions of the above types. Multilayer perceptron also appears in the above categorization and becomes popular as deep neural networks that have many hidden layers intermediately. Convolutional neural network (CNN) is the

most advanced type of neural network that is more accurate than deep neural networks and even better than humans. That means CNN has the seeds of superintelligence. A recurrent neural network (RNN) is specially used for natural language processing [3]. As a solution for the vanishing gradient problem long short-term memory (LSTM) has been introduced [3]. LSTM solves the above problem by restricting the changing of the stores' values by preventing activation functions within its recurrent components [3]. The attention-based network focuses on a specific area for example nose on the face eliminating other features of the face [3]. Generative adversarial networks (GAN) are also an advanced type of neural network that worked in the unsupervised learning category [3] with the help of reinforcement learning.

One of the oldest unsupervised ANN learning methods is the self-organizing map (SOM) of Kohonan [18]. Some of the things like the number of inputs for feed-forward multilayer perceptron (MLP) and inputs and weight initialization for SOM are the same. In contrast, SOM usually has two layers output layer is SOM in a neuron lattice of rectangular in shape or hexagonal shape [18]. Moreover, MLP takes the difference between expected output and actual output and SOM takes the difference between the input pattern and the output pattern to update the weight values in neighborhood neurons of the closest neuron in the SOM lattice [18].

In early researches about hyperparameter optimization was done on regularization parameters under the topic of Adaptive regularization in neural network modeling in 1998 [19]. Another of the earliest research papers that could find about hyperparameter optimization is found in the paper titled Gradient-Based Optimization of Hyper Parameters in 1999 [20]. As per the paper, several hyperparameters have been optimized using the computation of the gradient of a model selection [20]. The use of mathematics has derived an equation, especially for quadratic training criterion [20]. Though the model selection criteria selected a model from candidate models that have been created by the variation of given hyperparameters upon the generalization error [20]. Another research uses the majorization-minimization (MM) algorithms to estimate the user parameter in the L2 regularization [21]. In this research paper user constents per one alayer and multiple layers have been estimated [21]. Still a research didn’t conducted to optimize all the hyper paramerters of a neural network or predict all the hyperparameters in a neural network. Some researches has been conducted to

predict the hyperparameters in other machine learning techniques such as the Gaussian process for statistical regression [22].

When it comes to hotel classification, much literature is found in standard classification systems, but computer-operated automated classification systems. Customer needs are different according to the location of the hotel and the hotel types [23]. However, customer expectations for high-star hotels are higher than the low-star hotel [23]. Existing hotel classification systems are more focused on the tangible, quantitative facts such as infrastructure and facilities size of the bed types of the fabric of the bed, and the size of the TV they provided in contrast none of the classification systems considers the most critical intangible qualitative facts such as employ attributes, customer feedback in the rating systems and review systems such as TripAdvisor that any customer looks before booking a hotel [23]. United Nations World Tourism Organization reveals that guests visit 14 different travel-related websites to decide upon hotel bookings [23]. The research article "Hotel Classification Systems: A Case Study" suggested that online rating is one of the major components among others in its suggested model for hotel classification [23].

So, the hotel review ratings and hotel classification have caused an effect relationship that makes one who creates computer-based automated hotel classification can easily integrate the hotel review rating system as well. Customer rating in rating websites such as TripAdvisor is critical for hotels, especially to acquire new customers [24]. The lost trust in hotel classification has been regained due to the trust of the reviews of fellow travelers on rating websites. Moreover, word of mouth in the early era spread positive or negative words of mouth by the customers who got the positive or negative experience. Same happening in the current electronic, internet, and technological era called as eWord of mouth [24]. However, the negative word of mouth through review sites and social media affected so badly on the online image of the hotels so badly even though the hoteliers were not ready to accept it initially [24]. Hotel management should respond to every review and low-rated reviews are handled by the general manager of the hotel himself usually [24]. This implies how crucial are online reviews for a hotel.

Now draw the attention to the customized neural networks that are in AGI architectures wether there are near initiations had done in the direction of hyperparameter prediction or optimization. As per Ben Goertzel, there are various ways can achieve AGI. Those

ways are categorized into 4 as previously explained in the introduction. Namely, they are symbolic, emergentist, hybrid, and universalist. The application of neural networks has happened in the areas of emergentist, and hybrid approaches.

It is believed that no intelligence exists without symbols. The weight values in the neural networks are believed to be indirect representations of symbols. Naming the deep learning approaches as sub-symbolic due to this base. Emerging intelligence from sub-symbolic dynamics which are designed from the neural networks or any other aspect of brain functions, in terms of neurosciences "connectionist architecture" is considered to be in the emergentist AGI approach [12]. Moreover, the emergence of intelligence happening with the interactions of agents together seems to be considered in this category when fully observe the research article of Ben Goertzen's Artificial General Intelligence: Concept, State of the Art, and Future Prospects [12].

2.3 hyperparameter prediction, optimization in neural networks

Hyperparameter optimization become popular due to the upscaling of neural networks for enhanced accuracy [25]. Another objective is to provide a lightweight model with low wait parameters with more accuracy [25]. Hyperparameter prediction and optimization research have been conducted for specific machine learning techniques and by considering overall machine learning techniques. When comes to specific machine learning techniques even in the deep neural network there is researches about optimizing for Convolutional Neural networks using Linearly Decreasing Weight Particle Swarm Optimization [26]. When dealing with hyperparameter optimization metaheuristic algorithms play a major role. Metaheuristic algorithms are consisted with Genetic algorithms, particle swarm optimization, and ant colony optimization [26]. Despite of many algorithms are available for optimization of hyperparameters there are few on hyper parameter prediction due to manual optimization is very time consuming. But when one needs to automate the whole process hyperparameter prediction is also required. Binary classification problem is good point to start such a system because of the complexity of the problem is low. There is a way to predict binary hyperparameters using meta-learning concepts [7]. researchers have used a binary classification data set repository consisting of meta-features and optimal hyper-parameter by applying a Bayesian optimizer for the generation of hyper-parameter

prediction models [7]. Furthermore, they increase the number of models with versatility by applying forward model selection ensembles [7]. Their major concern is hyper-parameter optimization rather than prediction [7]. Both hyper-parameter prediction and optimization happen with the above procedure [7]. The preprocessing part has been done by the heuristic rules including the data splitting [7].

Hyperparameters used for use for design and model training can be considered as two categories of hyperparameters [25]. Hyperparameter optimizers used for training is results in fast learning of the data patterns [25]. Stochastic gradient descendant, Adam, and RMSprop are some of such optimizers [25]. How fast the convergence occur dependent importantly on the learning rate and the batch size too [25]. The remaining category of hyperparameters is used for the structure of neural networks more specifically number of layers(hidden), the number of neurons per layer..etc [25]. When the neural network has more layers it can capture more features while number of neurons per layer decide wether the network may over fitting or under fitting [25]. Regularization also reduces the overfitting of neural networks [25]. Moreover, the activation function makes the neural network capture more features otherwise will be a linear function that can only capture a few [25]. The hyperparameter tuning is a search of the best set of combined hyperparameters that gives maximum accuracy with minimum loss in the search space [25]. There are various search algorithms for searching the optimal values of the hyperparameters. Grid search, random search, Basian optimization, and Tree Parzen Estimators are some of them [25].

Basian optimization (BO) is more important because it finds global optimal in the minimum amount of trials [25]. In BO a probability surrogate model is built and then executed the trials according to the results of the trials probability model is updated and then again executed trials according to the new model [25]. The process continues until the best of the hyperparameters are obtained [25].

To uniquely identify a data set in this (BO)approach researchers use meta-learning [27]. In meta-learning data sets are uniquely identified by approaches such as Statistical and Information-Theoretic Characterization, Model-Based Characterization, and Landmarking [27]. Mapping datasets to predictive models is another aspect of achieving meta-learning [27]. According to that aspect, the hyperparameter prediction also can be taken as an attempt at meta-learning. However,

to achieve hyperparameter prediction and optimization Dataset Characterization has to be used. When hyperparameter prediction and optimization are done with basian optimization forward model ensembling is used to generate a set of candidate models (usually 25) to use together in various deep neural network problems [7], [8]. The ensemble is a collection of models that collaborate with weighted values or voting to give a collaborative output [28]. Moreover, in hyperparameter prediction and optimization research, the ensemble selection method starting from an empty ensemble has been used for the ensembling process instead of weighted values are voting methods [8].

Population-based algorithms are another alternative approach to predicting and optimizing hyperparameters [29]. In this approach, there are several techniques used for the population-based methods such as Genetic Algorithm"(GA), "Differential Evolution"(DE), and "Particle Swarm Optimization"(PSO) [29]. All three techniques have differences in population generation and evaluation [29]. The genetic algorithm is based on the ranking of the solution against the fitness value and the validity of the solution decoding into a possible solution [29]. GA leads to overfitting due to the increase of the population in a non-linear way [29]. When considering the PSO approach it chacks the difference of new swarm particles that are generated by velocity and position update equation [29]. This approach mostly results in premature convergence prominently fitness gaps are large [29]. In DE even it is somewhat similar to PSO better than it due to the best solution is not influenced by the other solutions in the population [29]. However, this approach has a drawback that is it should generate the population of model configuration when the new data set is provided to predict or optimize the hyperparameters that could be computationally costly.

Drawing the attention to ANN and RNN-type neural networks due to the objective is hyperparameter prediction in ANN and RNN networks. There are many characteristics of ANN such as parallel structure, learning, and adaptive capabilities, Very Large Scale Integrated (VLSI) implementation ability, and fault tolerance [10]. Moreover, it stimulates the biological brain's most important unique feature is its ability to "adapt" and "learn" [10]. According to the connection type among neurons, network architectures are divided into two main categories [10]. Namely, they are "feed-forward neural networks" and "recurrent neural networks" [10]. When a neuron of the network doesn't have connections from its outputs towards the inputs then the network

is referred to as a “feed-forward neural network” or otherwise “recurrent neural network” [10].In other words, stated as a feedback network. Single-layer or multilayer feed-forward networks are identified by the number of layers as two categories [10]. The input layer does not count as a layer as it doesn’t involve computation, so the single-layer network considers its only layer as the output layer that computes output signals according to the weights from the input layer [10]. In the multi-layer network, there are one or more layers intervening between the input layer and output layer in some useful manner [10]. The “hidden layer” is the layers in the middle of the input and output layers [10]. When each of the neurons in the current layer is connected to all the neurons in the next layer said to be fully connected otherwise even a single neuron in the current layer not connected with all the neurons in the next layer is partially connected [10]. The backpropagation algorithm is the most famous algorithm that is used for the feedforward neural network for training [10]. The algorithm updates the synaptic weights by back-propagating a gradient vector [10]. A mathematical explanation of training a feedforward neural network through a back-propagation algorithm, in brief, will be stated in the Chapter 3 technology section.

Due to recurrent or very deep neural networks often suffering from the exploding/vanishing gradient problem, training becomes difficult. [11]. To overcome these problems and shortcomings when learning long-term dependencies, the Long Short-Term Memory(LSTM) architecture was introduced [11]. The learning ability of LSTM results in the impact of several fields such as natural language processing (NLP), Google Translate uses it to improve machine translations and speech recognition [11]. Amazon Alexa has energized it by improving its’ functionalities with 4 billion LSTM-based translations on Facebook per day as of 2017 [11]. Moreover, AlphaStar which was created to play the Starcraft II game created by Google Deep Mind climbed up the global rankings by mastering the game, which was unseen before [11] and has energized with LSTM. Expansion of the research in this LSTM network is not limited to the entire RTS gaming genre but also to robotics for instance building a robot hand called Dactyl with support of the reinforcement learning [11]. There are 8 variants of LSTM networks [11]. None of the eight investigated LSTM variants significantly outperforms vanilla, which is the architecture that performs well on several tasks [11]. Vanilla LSTM has been achieved by integrating the peephole connections and “forget gate” to the original LSTM block [11]. Integrated LSTM

networks provide more advantages due to adding components other than LSTM resulting in hybrid neural networks [11]. Mathematical explanation and LSTM cell are discussed in the technology chapter.

When considering the mathematical model of the neural networks it is important to get to know about the complete mathematical model of the neural network. A logical calculus of the ideas immanent in the nervous activity presented by McCulloch and Pitts' was considered the first mathematical model of the neural network in 1943 [30]. Besides the backpropagation algorithm for learning neural networks, there are other algorithms such as Hebb, Abbott, sequential synaptic coefficients, Gardner, Diederich-Opper, and Krauth which call learning rules [30].

Dynamic system theory can be successfully used to express the learning process of the networks that contain nonlinear neurons, linear and weakly nonlinear [31]. Due to the arising problem of the learning process of neural networks mathematical models are formed [31]. For the model ANN architecture graphs and matrices have been used [31]. Even the ANN mathematical model can further extend to model RNN too [31]. The model and its equations are detailly stated in the technology chapter.

RNN is the type of neural network that comes to mind when sequential information or datasets to apply to deep learning methods [6]. Inputs and outputs are independent of each other in the traditional neural network contrast in with RNN output being dependent on previous computations [6]. Nevertheless, In cases of non-sequential input information, RNN has also shown great results such as in image captioning [6]. This is possible when sequentially ordered input and outputs are fixed vectors to process [6].

Classification is a major activity and application of ANN[32]. Statistical classification is a traditional classification procedure that has drawbacks as such it works well only when the underlying assumptions are satisfied [32]. The advantages of the ANN classifications over conventional classification methods are self-adaptive and data-driven according to the data [32]. Further, they are universal functional approximators capable of arbitrary accurate approximation of any function and neural networks are flexible in modeling real-world complex relationships by nonlinear models [32].

Finally, posterior probability estimation is possible which gives the basis for performing statistical analysis and establishing classification rules by them [32].

When come to the classification analysis dependent variable is influenced by both ratio scale variables and qualitative (nominal scale) variables [33]. Numerical inputs being the only type accepted by machine learning algorithms make a requirement to convert the nominal to numerical values through an encoding system [33]. There are 4 main categories of data variables [33]. A nominal scale that doesn't have numerical values such as married/unmarried or male, female [33]. They are further considered as categories specific according to the above marital status (separated, divorced, unmarried, married), and gender (female, male) [33]. Ordinal scale refers to the order in measurement but the distances between the categories cannot be quantified for instance Slower/Faster or High/Medium/Low [33]. Interval scales provide order but equal intervals such as Fahrenheit or Celsius when considering temperature [33]. Nominal, ordinal, and interval scales can extend to ratio scales by introducing something like absolute zero [33]. Whenever the above variable types are immersed for categorical data to be used, the data needs to be encoded into numeric values for Machine Learning purposes [33]. There are seven categorical variable encoding techniques [33]. Namely, they are Polynomial coding, ordinal encoding, Helmert coding, sum coding, binary coding, one-hot coding, and backward difference coding [33]. Nevertheless, any of the above coding techniques gives a solution when data is low and nominal and so diverse.

There are various types of neural networks among them ANN and RNN that can be considered as the oldest. In the future various kinds of neural networks with various capabilities will be discovered by the researchers. However, after 1990 various such types have been discovered.

Recursive neural networks (RvNN) can predict in a hierarchical structure and further utilizing compositional vectors can classify the outputs [4]. Primarily it was inspired by recursive auto-associative memory (RAAM) [4]. RvNN architecture has used randomly shaped structures like graphs or trees like processing objects [4]. Moreover, it uses a variable-size recursive-data structure utilized for fixed-width distributed representation [4]. Backpropagation through a structured learning system has been used to train the network [4]. Recurrent neural networks are already discussed where

LSTM comes under the recurrent neural network. Convolutional neural networks are the most famous type of neural network which it's most valuable feature is that it identifies the relevant features without any human supervision automatically [4]. CNN has a structure inspired by human and animal brains specifically, in a cat's brain [4]. It as the architectures such as AlexNet, High-resolution (HR) model, Network-in-network model, ZefNet, GoogLeNet, Highway network, ResNet, DenseNet, ResNext, ResNext, Pyramidal Net, Xception, Residual Attention Network (RAN) which blend attention into the CNN, Convolutional block attention module, CapsuleNet and High-resolution network (HRNet) [4].

Restricted Boltzmann machine (RBM) is yet another neural network type mostly used in deep neural networks due to its historical importance and relative simplicity [34]. Deep belief networks (DBN) are a descendant of RBM networks that serve as a nonlinear model for feature extraction and dimension reduction [34]. DBN is supported by multiple layers of RBMs [34]. Autoencoder (AE) is another type of ANN that is used for the dimensionality reduction of a data set through coding the dataset [34].

Graph neural networks (GNNs) operate on the graph domain by using deep learning methods [5].GNN is the most widely applied approach to the graph analysis method recently [5]. Recent discoveries in deep neural networks, especially convolutional neural networks energized GNN [5]. For non-Euclidean domains deep neural models introduced an emerging research area by the name of geometric deep learning [5]. GNNs are classified into four categories according to the blended neural network type [5]. Namely, they are graph autoencoders, spatial-temporal graph neural networks, convolutional graph neural networks, and recurrent graph neural networks [5].

There are attempts to overcome the challenges in neural networks by representing and embedding the domain knowledge in the flavor of symbolic representation [35]. To fulfill this need neuro-symbolic learning (NeSyL) notion emerged [35]. It integrates the aspects of symbolic representation and common sense into neural networks (NeSyL) [35]. NeSyL has shown promising outcomes in the domains of video and image captioning, question-answering and reasoning, health informatics, and genomics where interpretability, reasoning, and explainability are crucial [35].

Draw attention to customized neural networks in AGI architectures to see whether any potential designs result in hyperparameter prediction. DeSTIN is an emergent architecture created by integrating hierarchical temporal pattern recognition architecture with CogPrime architecture. Moreover, Hierarchical Temporal Memory (HTM) is used mainly in vision processing while present in both AI and AGI approaches with model cortex [12]. It can be considered as a special kind of hierarchical Bayesian model [36]. Further, to encapsulate and learn the structure and invariance of problem space it also uses spatial-temporal theory where both the hierarchical organization and spatial-temporal coding have well-documented principles of information processing in neural systems [36]. Nods of the Bayesian network are arranged in a tree-shaped hierarchy and all nodes share the same computing algorithm with the Bayesian belief propagation mechanism [36]. Inputs are fed into the bottom layer after performing the preprocessors [36] as happens in the human neocortex. To allow proper distribution of information throughout the network there are several feedback and feed-forward channels [36]. So those two types of networks need a set of training data to be put into the bottom layer of nodes multiple times indeed. CogPrime express in the hybrid AGI section.

Another fabulous AGI emergentist approach is neutrally organized mobile adaptive device automata and their successors [12]. It is based on Edelman's "Neural Darwinism" model of the brain. It features simulated neurons in large numbers through natural selection to achieve configurations to carry out sensorimotor and categorization tasks [12]. In the theory of neural Darwinism its status as the current structure of the brain is evolved by the natural selection of the neurons [37]. During the evolution, the thalamocortical networks were selected to provide humans with the ability to adapt in a superior way to their environment extending to make higher-order discriminations [37]. Value systems of the brain are a result of a strong influence on survival through natural selection [37].

Tsvi Achler had shown weights can be adopted in such as way that it follows the rules in updating weights while following the gradient descendent algorithm [14]. Moreover, this is an overcome of poorly symbolic representations through the difficulty of weights favorable for recognition [14]. To overcome the problem of recognition in their it uses feedforward-feedback configurated supervised recognition algorithm is implemented during testing where the dynamic phase is based on gradient

descent is a key finding [14]. This is a newly emerging field called neuro-symbolic learning.

The cognitive Computation Project is yet another large-scale brain simulator project that aimed reverse engineering of behavior, function, structure, and dynamics of the human brain [12]. They simulated the neocortex of a cat that has 109 neurons and 1013 synapses [12].

Neural subnetworks perform cognitive tasks by Simulation of the dynamic assemblage involved in the large-scale brain modeling especially related to the audition and language capabilities [12]. Brain imaging techniques data such as fMRI, PET, and MEG helped to guide the simulation [12].

Simulation on a scale like a full brain has been achieved in the Large Scale Model of Thalamocortical Systems [12]. It generates special brain features such as brain wave propagation and initial state sensitivity through plasticity and spiking features of the neural cortex [12]. Multiple compartments that have composed of millions of spiking neurons can simulate the model residing in the system joined by a half billion synapses [12].

CLARION is such an architecture that handles reasoning on “explicit knowledge” by the symbolic component and “implicit knowledge” by the connectionist component [12]. Implicit knowledge learning is done with reinforcement learning, neural net, or any other methods [12].

DUAL is yet another hybrid AGI architecture based on Mavin Minsky’s Society of Mind [12]. It provides the basis for high-level context-sensitive cognitive processes [15]. Its most significant feature is a mixture of symbolic and neural network mechanisms and computations [15]. It has a large collection of units called DUAL agents at its lowest level which communicate with each other by message passing as usual agents and spreading activation via weighted links [15]. In the “society of mind,” each mind consists of small processors [38]. If you grab the handle of a door to open it there is communication among small processors like seeing the door, your intention to open it, muscles and joints of your hand move to achieve the objective while communicating with each other [38]. Some of the processors happen sequentially and some hare happen in parallel such as talking with someone while doing it [38]. These

small processors are acting like agents and are also in the society of mind even though the concept appeared in the 1990's it already in the book Society of Mind in 1987 [38].

Dietrich Dorner's Psi model of intelligence, motivation, and emotion powered an AGI hybrid architecture called MicroPsi [12]. The psi model explains the interchange of mental representation, action, emotion, and perception [16]. It doesn't consider motivation, behavior regulation, perception, memory, or emotion in isolation, but blends them all into a common framework [16]. When looking at the neural representation of the architecture which is the emergentist aspect of the MicroPsi it consisted of a special neural network that has central neurons and 4 surrounding neurons. They are sub(has-part), sur(is-part), por(subjunctive), and ret (inverse relation to por) are connected with a central neuron forming a building block of the neural network as in Figure 2.2 [16].

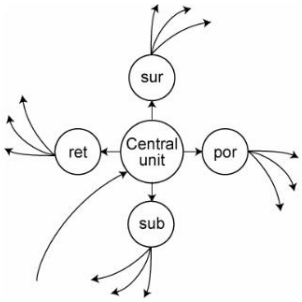


Figure 2.2: Quad—arrangement of nodes as a basic representational unit

Source 2.2: Principles of synthetic intelligence by Joscha Bach, Universität Osnabrück

Other than this type(quad) of neurons there are neural network types such as sensor neurons and motor neurons [16].

Another groundbreaking AGI model is Shruti inspired by biology to model human reflexive inference [12]. The model is based on connectionist architecture and uses focal clusters to represent types, causal rules, relations, and entities [12]. This model demonstrates how a connectionist network can perform class-level inference in a very short time such as a few hundred milliseconds through encoding that involves n-array predicates and variables of millions of facts and rules [39]. The model approach is by the appropriate nodes representing dynamic bindings execute synchronous firing, temporal pattern-matching sub-networks by long-term facts, and propagation of rhythmic activity results due to the direction of the rules as interconnection patterns

[39]. There are many customized neural networks in various types of AGI architectures but none of them still enjoy the benefits of hyperparameter prediction or hyperparameter optimization.

2.4 New Trends in hyperparameter prediction and optimization

There are different techniques of hyperparameter optimization like the Bayesian approach with model ensembles and population algorithm-based technique. All of these techniques had advantages and drawbacks. Technologies like large language models (LLM) have developed that can overcome the drawbacks of the above technologies such as setup complexity, trial efficiency, and interoperability[40]. An LLM-based naval approach named AgentHP is an agent-based, LLM-based solution that the user can interact with in a natural language providing details such as Optimization goals, data set characteristics, and model structure it will give an optimized model [40]. In AgentHP there are two types of agents the creator agent is responsible for providing the hyperparameters [40]. The executor agent is training model recording of experimental data while conducting outcome analysis [40]. So this is the first time one has used LLM for hyperparameter prediction and optimization [40].

As mentioned at the start of this thesis ChatGPT approach for data science by deep neural network is one future target for this research. GPT architecture also has been used with LLM and with auto ML under the name of AutoML-GPT [41]. In there user will give the data card and model card with additional parameters or restrictions through a paragraph to give a solution as an output [41]. The data card has data like data name, data type, and label space like meta features [41]. The model card is the structure of the model with the model name, model description, architecture hyperparameter..etc [41]. To predict hyperparameters as in the AutoML it finds the correlation between existing data sets with meta features [41]. Then, the hyperparameters with the most correlated dataset [41].

MLCopilot is an ML framework that can accomplish hyperparameter prediction and optimization with model selection according to the ML task described by the user through LLM [42]. Unlike previous approaches other than creating a mapping from

the metadata to the model it creates the mapping between the problem fed by the user and the model through text embedding using LLM [42]. So this avoids the requirement of metadata [42]. When the user inputs a problem description, it loads the previous experiences which are previous demonstrations with the same context even with their accuracies in other words the problem space [42]. The problem space consisted of a dedicated experience pool and the knowledge pool [42]. The knowledge pool has a higher level of knowledge acquired by the experience pool by the process called knowledge elicitation [42]. According to the experience and knowledge pool LLM parameters and settings are changed [42]. Using the above two pools LLM can produce a prompt fine-tuned model structure as a solution that can used by the user [42].

Moreover, the use of hyperparameter tools is increasingly popular too. Also, some of them are hyperparameter-optimizing frameworks[43]. Optiuna is one of them but it has some drawbacks like provide the hyperparameter space for each hyperparameter [43] and it tends to assign the same hyperparameter for all layers as more specifically activation function. Besides Optiuna there are other hyperparameters optimizing tools such as Google Vizier, Scikit-Optimize, Hyperopt, SigOpt, and Talos.

Google Vizier is a black box optimizer [44]. A complex optimizer system that is easy to use without understanding the system can be considered black box optimizer[44]. Google Vizier works as a Google internal service for tuning ML applications of Google. Also responsible for core capabilities in Cloud Machine Learning HyperTune subsystem [44]. Google Vizier gives Ease of use, Minimal user configuration and setup, Hosts state-of-the-art black-box optimization algorithms, High availability, Scalable: millions of trials per study, thousands of parallel trial evaluations per study, and billions of studies, Easy to experiment with new algorithms, Easy to change out algorithms deployed in production [44].

Scikit-Optimize is a hyperparameter library residing in the Python sklearn library [45]. There are 3 approaches in there to achieving optimization [45]. Namely, gp_minimize, forest_minimize, gbrt_minimize [45]. In gp_minimize use the Gaussian process with ba bayesian optimization [45]. For forest_minimize uses a Decision tree-based regression model [45]. gbrt_minimize uses gradient boosted tree to determine hyperparameters[45].

Hyperopt is also a Python-based library that focuses on hyperparameter selection and optimization [46]. Hyperot usually uses SMBO (sequence model-based optimization) can be interchanged with other optimization algorithms such as random search, Tree-of-Parzen-Estimators (TPE), and simulated annealing [46]. It also allows to state of the objective function that has to be minimized as a simple Python function [46]. Also, it allows to defining of a configuration space with a distribution [46].

SigOpt is not restricted to hyperparameter optimization[47]. It can also be used in the optimization of the hardware system environment variables, design parameters of the simulated materials, and resource allocation of a cluster [47]. SigOpt supports constraints and multiple objectives [47]. SigOpt also supports Complex search space with integer, discrete, categorical, continuous, variables, conditional dependencies, and parameter constraints [47]. It uses a web dashboard to show the overview of experiments with exploration [47].

Talos is similar to Optiuna, user has to create the configuration structure and place the placeholder without the search space such as [“relu”,”Sigmoid”,”softmax”] in the Optiuna [48]. For example, the usual tensorflow layer is as follows.

modeldropPrdfL2.add(Dense(9, activation=params[‘activation’]))

So it works with any keras (TensorFlow) or pytorch model and is easy to implement [48]. No overhead to the workflow and bugs [48].

2.5 Analysis of Literature

This section provides the analysis of the literature on the aspects of technology used, advantages, and limitations as in Table 2.1. The first citation is from a survey so that citation didn't include in the table.

Table 2.1: Table of analysis of the literature

Research (1st author & citation)	Technology used	Advantages	Limitations
P. P. Ray [2]	LLM,RL, Transformers	The context knowledge acquired by the training of LLMs like GPT results in to generate responses, completing tasks, and even writing coherent pieces of text	their sheer size and complexity can make LLMs like GPT difficult to interpret and analyze. Biaseness due to training data
E. Nisioti [7]	Bayers optimization, Meta features, forward model ensembling	Predicting hyperparameters and optimization by using meta features through basian optimization to results model ensemble	Research is confined to hyperparameter prediction and optimization in binary classification problem

P. Shankar [3]	ANN, CNN, RNN, LSTM, GAN,	Problems the not exsist or are expensive by algorithmic approach can be solved easily by ANN	Inadequate review done on AIN and GAN
L. Alzubaidi [4]	CNN, GAN, LSTM	Using small-size filters in CNN computational complication reduced	Requirement of the large training set. To obtain good performance, CNN models require a sizable volume of data
J. Zhou [5]	GNN	Use of CNN operators to generalize CNN from another domain to the graph domain	Less robust venerability to adversarial attacks, Difficulties in handling very complex graphs such as in social network domains
C. Chopra [6]	RNN, support vector machine (SVM)	Application of RNN on non-sequential data.	Recall, and precision, parameters are not considered.

M. Feurer [8]	Bayers optimization, Meta features, forward model ensembling	Reduce the search space for Bayesian optimization by selecting the nearest k samples by using the meta-features.	This approach is best only for low-dimensional problems. Basian optimization is slow for large spaces of hyperparameters
M. Sarker [9]	ANN	Adaptive learning, Self-Organization	ANN is not a daily life general-purpose problem solver.
M. Sazli [10]	ANN,	brief mathematical proof of the backpropagation algorithm	Only restrict to mathematical proof backpropagation algorithm
G. Van Houdt [11]	Vanilla LSTM	vanilla LSTM forward pass and backward pass, solution for vanishing and exploding gradients	In the use of LSTM with CNN to detect conversation anomalies: initial and stimulating emotions cannot be set at any time during the conversation
B. Goertzel [12]	AGI,	Their ability to accomplish a	Human level general

		variety of goals by performing a variety of tasks in a variety of contexts and environments	intrlligance still not achived
D. Jilk [13]	ANN	Convergent activation dynamics and bidirectional learning with top-down biasing and bottom-up statistical learning of information processing functions	It is still not developed to simulate meta-cognition
T. Achler [14]	ANN, Neuro symbolic learning	Gradient descendant follows symbolic rules as well.	In some cases, due to data is not separable learning may not converge in scenarios.
A. Nestor [15]	DUAL-Architecture ANN, Multi-agent system (MAS)	The duality between MAS and ANN blends symbolic and sub-symbolic together	Making activation spread in the system has to be a very efficient and reliable mechanism.
J. Bach [16]	ANN	autonomous learning,	PSI Agents lack self-reflection,

		emotional modulation, display numerous problem-solving abilities, and hierarchical perceptual processing	meta-management abilities, flexible planning
M. Haenlein [17]	ANN, DL	Four season of AI	Try to limit the development of AI due to profit gain by automation spending to train employees
A. M. Kalteh [18]	ANN	Used for analysis, prediction, and estimation of various hydrological processes	Guesswork and/or trial-and-error approaches are characterized as generally dependent on ad-hoc approaches in SOM applications
J. Larsen [19]	ANN, DNN	Optimization of the regularization parameters	Can't use adaptive regularization for a large set of regularized parameters due to the tendency of overfitting

Y. Bengio [20]	ANN…etc	Use of gradient of a model selection criterion to optimize several hyperparameters simultaneously	Overfitting occurs when too many hyperparameters are predicted.
C.-S. Foo [21]	ANN	High efficiency for large datasets relative to the grid search and gradient-based algorithms	majorization-minimization algorithm applies only for L2 regularization with supervised learning
C. Rasmussen [22]	Gaussian process	Can be conveniently used in flexible non-linear regression	Non-gaussian likelihood training is complicated.
T. Sufi [23]		Hotels are ranked according to their services and facilities allowing customers to choose the best hotels	Due to corruption hotel classifications not realistic
R. Chowdhury [24]		Due to true reviews of the review sites booking reviews	The research was conducted only in two cities hence the behavior of

		strongly impact booking	other towns may different
T. Yu [25]	ANN	Basian optimization is the most promising algorithm over the Grid Search, Random search, Multi-bandit methods, and PBT methods	Grid and the random search are the worst performing and only applicable to a few hyperparameters and only worth applying at the early stages
T. SERIZAWA [26]	CNN, DNN, RNN	linearly decreasing weight particle swarm optimization works best for CNN	Have to test on other datasets other than benchmark image datasets
R. Vilalta [27]	Metalerning	Metalerning can be used to identify data sets through it' characteristics	need for alternative -more detailed- descriptors of the example distribution
R. Caruana [28]	Model ensemble	Encembeling can achieve high-quality models that can outperform any easily computed	Encembeling need a collection of trained models leading to

		performance metric	computationally costly
B. Panda [29]	ANN	Population algorithm-based hyperparameter tuning can achieve more accuracy than the classical hyperparameter tuning methods such as Manual, Grid, and Random Search	Inadequate research done to identify behaviors against basian optimization
R. S. Segall [30]	ANN	different learning rules, Boolean learning functions with backpropagation work well in convergent and non-convergent neural networks.	Inadequate research in to Applications of fuzzy logic with neural networks
A. Bielecki [31]	ANN	introducing bijection mapping β	Research is Restricted to scaler products and real activation functions, Innaduquate research into RNN

P. Zhang [32]	ANN	Estimating posterior probabilities over the conventional classifiers, Mathematical demonstration of classification	researsh not extends for model designing
K. Potdar [33]	Encoding of ANN/RNN	Sum Coding and Backward Difference Coding techniques are the best from other techniques such as Helmert Coding, Polynomial Coding, and Binary Coding	Inadequate research done into nominal values
Manikandan. B [34]	ANN, RBM, DBN, RNN, DNN, AE, CNN	CNN requires minimal preprocessing.	In this research mathematical approach didn't considered
M. Hassan [35]	ANN-Symbolic	feed the advantages of symbolic learning into neural networks, enhancing interpretability and explaining the	In knowledge graphs, the approach may result in the loss of relevant features to be encoded as one heat code.

		ability of ANN results, solution for adversarial attacks	
X. Chen [36]	HTM	The structure of the HTM architecture and its different versions can perform tasks such as recognizing objects, making predictions, and discovering patterns in complex data	The time factor is not considered in the message passing in the HTM architecture
G. Edelman [37]	ANN,	Approaches to consciousness by recognizing patterns, ability to self-organize, learn, and evolve on its own	Systems develop though the neural darwinism has inbult biasness
M. Minsky [38]	Multiagent approach (concept)	Explain the mind as a set of small processors working collaborated, part to-whole thinking enables easy	Not readily demonstratable but strong as a concept.

		understanding of the mind	
L. Shastri [39]	ANN, Shruti architecture	simply detecting coincidence (or the lack of it) among their inputs instead of synchronous activity is 'read' by various long-term structures in the system	The model further should be developed for reflexive reasoning.
S. Liu [40]	LLM,GPT,Multiagent	Accuracy against some benchmark datasets is supper seeded the expert accuracy by AgentHPO	Have to provide the model details also by the user. Has to check the behavior for other benchmark datasets.
S. Zhang [41]	LLM,GPT	GPT has used to take the user input with data card and model card to create and then traing the model to optimal hyperparamters using biliancs of parameters in LLM for learn the	Due to use of model cardstill user required to give the model details

		parameters optimality.	
L. Zhang [42]	LLM, GPT	Due to the problem being mapped into the model user does not need to give model details, due to experience pooling, and the knowledge pooling knowledge is acquired by the user input and the change of the prompt as experience too.	The potential of benchmark datasets already used for training due to the usage of the large corpus of data from the Internet. Besides, this will not work without the internet
T. Akiba [43]	Optimization frameworks,toolkits for hyper parameter optimization,Optuna	The scalable and versatile design of the framwork make it userble for wide variety of works	Define by run make use should enter the search space every time. For example, for activation functions the user should give a list of activation functions to apply
D. Golovin [44]	toolkits for hyperparameter	Scalable to millions of trials per study,	Due to the implementation as a Google service

	optimization, google Vizier	thousands of parallel trial evaluations per study, and billions of studies	to work on this internet is mandatory
H. Shaziya [45]	toolkits for hyperparameter optimization, Skopt	Out of gp_minimize, forest_minimize, and gbrt_minimize the first one is the best which is based on Bayesian optimization	Research was conducted on the optimization approaches in Skopt, not with other approaches such as Optuna, google vizier
J. Bergstra [46]	toolkits for hyperparameter optimization, hyperopt		The user should give the configuration details to the optimizer

Source 2.3: All the papers read to acquire knowledge

2.6 Problem definition

There are several approaches for hyperparameter prediction and optimization. Each of these approaches has its limitations and advantages. One approach is through meta-features with Bayesian optimization and the creation of model ensembling. As explained in the above literature it maps the model with meta-features of the new dataset with existing model ensemble meta-features datasets and tries to generate a model with optimum hyperparameters. The same data set can be used for different types of problems. When identify the model through metafeature silimarities there is a probability of generating same models for diffent problems that can use to solve by the same data set. There is inadequate research on how this approach behaves when come to this situation. Moreover, the provided solution by this approach does not work

for Windows OS wich the solution is named as AutoML. In addition to that due to the use of model ensembling, it should run up to 25 models according to the availability of the RAM. For full model run for a given problem can be considered as 25 models require larger RAM than most usual machines don't have. In addition, it has a black box nature due to less representation of training parameters predicted even though it gives the models generated for ensembling. There are other approaches like the use of population-based algorithms such as Genetic Algorithm"(GA), "Differential Evolution"(DE), and "Particle Swarm Optimization"(PSO) for generating model configurations. However, this approach needs to generate a population of configurations each time is their limitation. The most latest trend in hyperparameter prediction and optimization is though LLMs.most of these approchers again needs to to enter the model details too. This approach due to the server-centred architecture results in high resource usage due to billions of parameters without having internet can't access to LLMs-based solutions. On the other hand, an expert can come up with a configuration that is more suitable than the approaches in above. Alternative to the above three approaches no research has attempted before to predict hyperparameters through feature selection. When there is no internet, the fastest approach would be the stated basian optimization. So, the focus is given to the backs of the stated basian optimization as the defined problem.

So this defines the problem as there is still no automated model selection between ANN and RNN and hyperparameter prediction through feature selection.

2.7 Summary

In this chapter, draws the attention to the brief history of artificial intelligence with existing hyperparameter prediction and optimization methods and customized neural networks existing with hyperparameter prediction tools. Moreover, it discussed technology, advantages, and limitations in the analysis of the literature section. In the next chapter, that will give the attention to the technologies for achieving hyperparameter prediction and model selection.

CHAPTER 3

TECHNOLOGIES ADOPTED

3.1 Introduction

In the previous chapter, immersed deeply in literature. Now in this chapter, that will go through the technologies that help to accomplish the aim and objective of the research. There are various types of neural networks such as feed-forward neural networks with backpropagation, Recurrent neural networks, recursive neural networks, convolutional neural networks, graph neural networks, and many more even underlying existing AGI architectures. ANN and RNN had chosen because they are the oldest types of neural networks hence so many applications of them are available to even publicly. Moreover, in the feedforward networks, the popular algorithm is the backpropagation algorithm [10]. On the other hand, LSTM is the most popular in the RNN[11]. In the LSTM the most popular variant is the vanilla LSTM [11]. The technologies arranged in this chapter are in the order of the first use of them in the development process of the general model the model auto configure generator. The references stated in this chapter are not only from the research papers but also from the coding sites that anyone can run and see the results. So, after the reference 35 are mostly from the above sites.

3.2 Panda data frame

The Panda data frame is a Python library that was created to analyze, clean, explore, and manipulate data [49]. The name pandas implies that both "Panel Data", and "Python Data Analysis" were created by Wes McKinney in 2008 [49]. Panda makes it easy to analyze big data and make conclusions based on statistical theories [49]. Moreover, pandas can use for cleaning messy data sets, and make them readable and relevant [49].

In this research project panda frames have been used for all data manipulations not only in the created component such as general data frames but also in almost all the examples. Pandas are a mandatory library to install if anyone gets benefited from the output of this research. 40 examples were considered to create the dataset to train the ANN-RNN classifier are the examples mostly underplayed by the panda frame.

To upload the data to Panda Frame by following the command

newDataframe=pd.read_csv('GeneralFrameWithIndVars.csv',index_col=0)

and check the data by the bellow command.

newDataframe.head()

Finding a column related to a specific value that contains in the dependent list.

column=originalDf.columns[originalDf.isin([self.dependentList[dependentIndex]]).any()][0]

And replace that value in the above-found column.

originalDf[column].loc[dependentIndex]=originalDf[column].replace([self.dependentList[dependentIndex]],[np.NaN])

Merge the cell to a row fulfilling the final row length.

self.newDataframe.loc[index]=[file]+sq.seequanceRow(feacherList)

Besides all the codes that created a panda frame according to the structure of the general data frame as in the following codes.

dataframeHeadings=['file','indV01','indV02','indV03','indV04','indV05','indV06','indV07','indV08','indV09','indV10','indV11','indV12','indV13','indV14','indV15','indV16','indV17','indV18','indV19','indV20','indV21','indV22','indV23','indV24','indV25','indV26','indV27','indV28','indV29','indV30','indV31','indV32','indV33','indV34','indV35','indV36','indV37','indV38','indV39','indV40','indV41','indV42','indV43','indV44','indV45','indV46','indV47','indV48','indV49','indV50','depV01','depV02','depV03','depV04','depV05','depV06','depV07','depV08','depV09','depV10']

self.newDataframe=pd.DataFrame(columns=dataframeHeadings)

3.3 TensorFlow

TensorFlow is a Python-based open-source library[50] that enables machine learning applications to be far more accurate while being easy to implement. Thanks to machine learning frameworks such as Google's TensorFlow application of machine learning and neural network development become less daunting through the process of acquiring data, training models, serving predictions and refining future results [50].

This fabulous open-source library was created by the Google Brain team and initially released to the public in 2015 [50].

In this research project, the tensor flow has been used to create the model for the following binary classifier model creation

```
inputs = tf. Keras.Input(shape=(60,))

x = tf.keras.layers.Dense(35, activation='relu')(inputs)

x = tf.keras.layers.Dense(18, activation='relu')(x)

x = tf.keras.layers.Dense(8, activation='relu')(x)

x = tf.keras.layers.Dense(3, activation='relu')(x)

outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf. Keras.Model(inputs, outputs)

model.compile( optimizer='Adam', loss='binary_crossentropy',

metrics=[ 'accuracy',  tf.keras.metrics.AUC(name='auc')     ])
```

It has the input layer which takes the inputs from the NumPy array and panda's frame

Then 5 dense layers represent the feedforward network with the backpropagation algorithm for the learning system. Further, the two-problem selected to integrate also created their models with TensorFlow. The following model is created for the classification of the hotels using ANN.

```
inputs=tf.keras.Input(shape=(246,))

X=tf.keras.layers.Dense(64,activation="relu")(inputs)

X=tf.keras.layers.Dense(64,activation='relu')(X)

outputs=tf.keras.layers.Dense(1,activation='sigmoid')(X)

model=tf.keras.Model(inputs,outputs)
```

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy',tf.keras.metrics.AUC(name='auc')])

It also has an input layer for taking the input from the matrix and has 3 dens layers with optimizers and lost function. The hotel review rating prediction model is an RNN model which has the following model configuration.

model = tf.keras.Sequential([#Start the sequential model, doing one layer after another in a sequence

Layer.Embedding(size, outputDimensions, input_length = texts.shape[1]), #Embed the model with the number of words and size

Layer.Bidirectional(Layer.LSTM(units, return_sequences = True)), #Make it so the model looks both forward and backward at the data

Layer.GlobalMaxPool1D(), #Take the max values over time

Layer.Dropout(0.3), #Make the dropout 0.3, making about a third 0 to prevent overfitting.

Layer.Dense(64, activation="relu"), #Create a large dense layer

Layer.Dropout(0.3), #Make the dropout 0.3, making about a third 0 to prevent overfitting.

Layer.Dense(3) #Create a small dense layer.

The embedding layer receives the word embeddings according to the text. Shape where the review text holding by text array. Then the data flow through the LSTM layer bidirectionally enables weight configuration more accurately by taking the max values over time and feeding them into the larger dense layer and smaller dense layer while restricting overfitting by introducing dropouts.

3.4 Mathematical Model of ANN

This is the extension of the mathematical model discussed in Chapter 2 literature search [31]. Due to the model used for the project having only one output neuron the model explanation has been reduced to a linear model with a single output neuron. The

neuron is a function of two vector variables. K Inputs transfer through a set of $X \subset IR^k$ is create a function F [31].

$$F : \mathbb{R}^k \times X \ni (\vec{w}, \vec{x}) \longmapsto F(\vec{w}, \vec{x}) = f(\langle \vec{w}, \vec{x} \rangle) \in \mathbb{R},$$

Where, $\vec{\omega}$ is the weight vector. $\langle \cdot, \cdot \rangle$ is the scaler product. f: IR → IR is the activation function.

When the neuron is trained, the above function can be denoted as below [31].

$$F^* := F(\vec{w}, \cdot) : X \ni \vec{x} \longmapsto F^*(\vec{x}) \in \mathbb{R},$$

The trained neuron's weight vector is a fit. There a trained neuron is a mapping of only one vector variable [31]. The scalar product can be denoted further as follows [31].

$$\langle \vec{w}, \vec{x} \rangle = \sum_{k'=1}^{k} w_{k'} \cdot x_{k'} := \vec{w} \circ \vec{x}.$$

The identity function can use in a linear neural network as the activation function [31]. Here it restricted the consideration to the composition of a scalar product and a real function as an activation function of a neuron [31].

3.4.1 Mathematical model of multilayer ANN

Multilayer architecture is based on graph theory [31]. (A, $\mathscr{E}$) are order pairs where A is a finite set of nodes and if $\mathscr{E} \subset A \times A$(fully connected) is a set of oriented edges can identify as an oriented graph or orgraph [31]. The edge is directed from the node a_i to the node a_j in such a way that $(a_i, a_j) \in \mathscr{E}$ [31]. The degree of graph G can express as the node set the power of graph G where denoted by δ_G [31]. For instant, if take the graph G= (A, $\mathscr{E}$), the power of the set $\{aj : (a_j, a_i) \in \mathscr{E}\}$ said to be input semi-degree of the node a_i and denoted by $\delta_{a_i}^+$ and on the other way for the set $\{a_j : (a_i, a_j) \in \mathscr{E}\}$ is denoted as $\delta_{a_i}^-$ for the node a_i [31]. Orgraphs are commonly used in the definition of ANN where graph nodes are the neurons, and the directed edges are inputs that sent by the other neurons (previous neurons).

G:= (A, $\mathscr{E}$) an orgraph of a degree δ_G such that $\{a \in A : \delta_{a_i}^+ = 0\} \neq \emptyset$;

γ: A ∋ a → γ(a) ∈ {1,...,δG} – a bijection mapping;

$\mathscr{J}$– the set of all neurons.

α: A ∋ a → α(a) ∈ F; if $\delta_{a_i}^+ = 0$ then α(a) is a k-neuron, otherwise α(a) is a $\delta_{a_i}^+$ neuron;

W –set indexing all inputs of neurons in the ANN;

β : $\mathscr{E}$ → W – a bijection mapping.

A_k:= (G,γ,α, W,β) is called an ANN Architecture.

If $\mathscr{J}$ is the set of all trained neurons then the above 5 tuples are said to be trained k-ANN architecture [31]. As per $\{a \in A : \delta_{a_i}^+ = 0\} \neq \emptyset$; their exist input neurons in the networks [31]. But the definition doesn't worry about output neurons due to recurrent networks are considered in which such neurons do not exist [31]. On that occasion, dynamical excitations of neurons in general and asymptotic equilibrium are the response to the input signal [31]. Natural numbers to graph nodes assign by mapping γ. Moreover, the mapping α assigns the neuron to the graph in the way that a k-neuron (if a network is a k-ANN) is assigned to a node in which the input semi-degree is equal to zero, whereas a $\delta_{a_i}^+$ neuron is assigned to a node with an input semi degree $\delta_{a_i}^+$ [31] . When denoting the weight values in such a way that first is neuron number, and the second is the number of the neuron input [31]. For multilayer it will be three tuples, first for layer number, second is for number of the neuron and the last is for input number of the neuron [31]. The mapping of the β is important especially in AGI because it determines to which input of a neuron the signal from another neuron or the ANN input signal is given. Because every input of a neuron is weighted and β is a bijection, every weight in noninput neurons of the ANN can be identified with an edge of the graph describing the ANN architecture [31]. In ANN all the layers can have the same activation function or different activation functions for a different layer [31]. Fully connected or complete ANN is if the following,

A_k := (G = (A,E),γ,α,W,β)

architecture exists in such a way that if set A of nodes of the corresponding orgraph G can be decomposed into a finite family of disjoint sets A_1,..., A_R while enabling each graph edge (ai, aj) to satisfy the condition ai ∈ Ar, and aj ∈ Ar+1, where r ∈ {1,..., R − 1} if it is an R-layered ANN then every pair a_i, a_j of nodes such that ai ∈ A_r and a_j ∈ Ar+1 the ordered pair (a_i, a_j) is an edge of the graph [31].

While ANN is in the above architecture there are two sets X ⊂ IR_k and Y ⊂ IR_m, where m is the number of the output layer neurons.

$\mathscr{L}_k$(X, Y):= (X, A_k, Y) said to be R-Layer K-ANN on X. where X and Y are sets of input and output signals [31]. For RNN it is a set of dynamic excitations of neurons or equilibrium states attained in response to input signals will be for the Y [31].

$$\mathscr{S} := \{(\vec{x}, \vec{y}(\vec{x})) : \vec{x} \in X, \vec{y}(\vec{x}) \in Y\},$$

Where, $\vec{y}(\overrightarrow{x})$ is the output signal of the network $\mathscr{L}_k$(X, Y) if $\vec{x}$ given as the input to build the following function

$$\mathscr{S} : X \ni \vec{x} \longmapsto \mathscr{S}(\vec{x}) = \vec{y}(\vec{x}) \in Y,$$

$\mathscr{L}_k$(X, Y) in the above single linear M -neuron on X ⊂ IR_M generates a linear function

$\mathscr{L}$: X → IR

Let two trained R-layer k-ANNs be given: (X, A_k, Y) and (X, A_k^*, Y), and let only the second one be complete. The architecture of the first one is.

A_k := (G = (A,$\mathscr{E}$),γ,α,W,β)

Architecture for the second one is.

A_k^*:= (G* = (A*, $\mathcal{E}^*$), γ*, α*, W*, β*)

If A_k^*= $A_{k,}$ $\mathcal{E}$ ⊂ $\mathcal{E}^*$ If weights of the network (X,A_k^*, Y), corresponding to edges that do not exist in (X, A_k, Y) are all equal to zero[31]. Then the remaining weights are of (X,A_k^*,Y) are equal to the corresponding weights of (X,A_k,Y), then the networks are equivalent. Every incomplete or partially connected ANN is equivalent to a complete ANN, it is sufficient, without losing the generality, to consider only complete ANNs

[31]. Every linear multilayer M -ANN is equivalent to a one-layer M -ANN. A multilayer ANN is a superposition of one-layer networks of which it consists [31]. Let a multilayer ANN (X, $\widetilde{A_k}$,Z) with the corresponding function $\mathscr{L}$: X → Z be a superposition of (X,A_k,Y) and (Y,A_m^*,Z), with corresponding functions $\mathscr{L}$: X → Y and $\mathscr{L}^*$: Y → Z, respectively [31]. Then it represented as,

$$\tilde{\mathscr{L}} = \mathscr{L}^* \circ \mathscr{L}.$$

In a multilayer ANN act in such a way that response to a given input signal the output signal should be equal to the desired value.

$$\left((\vec{x}^{(1)},\vec{z}^{(1)}),\ldots,(\vec{x}^{(N)},\vec{z}^{(N)})\right),$$

When, $\vec{x}^{(i)}$are the input signals k-ANN on X and desired output signal is $\vec{z}^{(i)}$.it called a training sequence of the multilayer k-ANN on X.

$$E:\mathbb{R}^{s_1} \ni \mathbf{w} \longmapsto E(\mathbf{w}) \in [0,\infty)$$

The above mapping is said to be the error function of E(w) in the form of

$$U\left(\vec{y}^{(1)}(\mathbf{w}),\ldots,\vec{y}^{(N)}(\mathbf{w})\right),$$

Where,

$$U:\left(\mathbb{R}^{M_R}\right)^N \longrightarrow [0,\infty) \text{ and } \vec{y}^{(n)}:\mathbb{R}^{s_1} \longrightarrow \mathbb{R}^{M_R}$$

The number of inputs in the output layer is s_1 and the number of neurons in the output layer is M_R while it should satisfy the following equalities.

$$\vec{y}^{(1)}(\mathbf{w}) = \vec{z}^{(1)},\ldots,\vec{y}^{(N)}(\mathbf{w}) = \vec{z}^{(N)}$$

3.4.1.1 Backpropagation algorithm

$e_j(n) = d_j – y_j(n)$ (1)

The desired output of neuron j is denoted by DJ and y_j is the actual output that is calculated by using the current weights of the network related to the iteration n for the neuron j [10]. Error energy for the neuron j is given by,

$$\varepsilon_j(n) = \frac{1}{2} e_j^2(n) \quad (2)$$

For all neurons in the output layer, it is,

$$\varepsilon(n) = \frac{1}{2} \sum_{j \in Q} e_j^2(n) \quad (3)$$

For N examples the average would be

$$\varepsilon_{av} = \frac{1}{N} \sum_{n=1}^{N} \varepsilon(n) \quad (4)$$

For the output expression for y, it can write it as

$$y_j(n) = f\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n) \right) \quad (5)$$

Where f is the activation function while w_{j0} equals the bias j_b applied to the neuron j[10]. And it corresponds to a fixed input y_0=+1. Weight updates applied to neuron is proportional to the partial derivatives of the instantaneous error energy Ɛ(n) by corresponding to the weight. $\partial \varepsilon(n) / \partial w_{ij}(n)$

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial w_{ji}(n)} \quad (6)$$

$$\frac{\partial \varepsilon(n)}{\partial e_j(n)} = e_j(n) \quad (7) \text{ obtained by (2), (1), (5)}$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \quad (8)$$

$$\frac{\partial y_j(n)}{\partial w_{ji}(n)} = f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n) \right) \frac{\partial \left(\sum_{i=0}^{m} w_{ji}(n) y_i(n) \right)}{\partial w_{ji}(n)}$$

$$= f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) y_i(n) \quad (9)$$

Where,

$$f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) = \frac{\partial f\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right)}{\partial\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right)} \quad (10)$$

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = -e_j(n) f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) y_j(n) \quad (11)$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} \quad (12);\ \eta \text{ is the learning rate.}$$

3.4.1.2 Other learning rules

There are other learning rules such as Hebb, Diederich-Opper, Gardner, Abbott, Krauth, and sequential synaptic coefficients [30]. Here it is only state that Hebb's is the oldest of them only.

The synaptic strength or coupling between the neuron i and j given by,

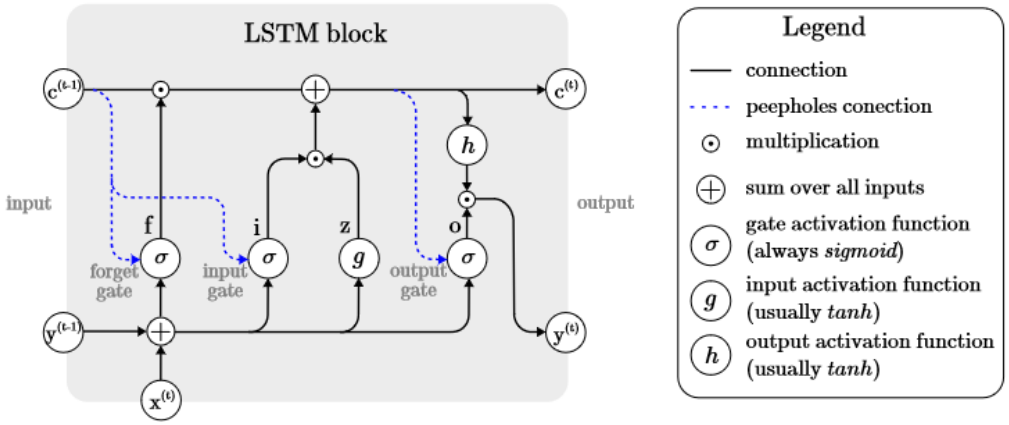
$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{P} \sigma_i^{\mu} \sigma_j^{\mu}$$

Where, σ_i^{μ}- stored pattern μ in memory by neuron i

W_{ij}- Synaptic strength between neuron i and j and p- number of patterns and N-number of neurons [30]. If $\sigma_i^{\mu} = -\sigma_j^{\mu}$ then, w_{ji} <0 results in inhibitory and $\sigma_i^{\mu} = \sigma_j^{\mu}$ due to both are active or dormant for the given time, so , w_{ji} >0 results in synaptic excitatory [30]. Furthermore, if w_{ji} = 0 the two neurons are not connected [30].

3.5 Long Short-Term Memory

Chapter 2 discusses the usage of LSTM. Now it will discuss how the LSTM works. Also, as already mentioned, it focuses on the most popular variant of LSTM, Vanilla LSTM. The vanilla LSTM compost with 3 gates namely an input gate, an output gate, and a forget gate [11].



Source 3.1:A Review on the Long Short-Term Memory Model by Greg Van Houdt · Carlos Mosquera · Gonzalo Nápoles

Figure 3.1:Architecture of a typical vanilla LSTM block

Assuming the LSTM has an N processing block with M inputs the function of the LSTM has described as follows.

Z is the input of the neuron as the usual input of a neuron but as per the structure of the block, the additional components has added as the following equation.

$$z^{(t)} = g(W_z x^{(t)} + R_z y^{(t-1)} + b_z) \quad (1)$$

where W_z and R_z are the weights associated with $x^{(t)}$ and $y^{(t-1)}$, respectively, while b_z stands for the bias weight vector. $y^{(t-1)}$ is the previous time input which is zero at the initial state. $x^{(t)}$ is the input of the current time. W_z $x^{(t)}$ and b_z are usual components of a neural network except for the time concern. G is the activation function usually tanh [11].

The gate controls the above input is the input gate according to the current input $x^{(t)}$, the last output of the LSTM cell, $y^{(t-1),}$ and the cell value $c^{(t-1)}$ at the last time/iteration as bellow equation [11].

$$i^{(t)} = \sigma(W_i x^{(t)} + R_i y^{(t-1)} + p_i \odot c^{(t-1)} + b_i) \quad (2)$$

$\odot$ is the pointwise multiplication of the two vectors. W_i, R_i, p_i, b_i are the associated weights and the bias term where σ is the activation function usually sigmoid. According to the status of the network it will decide with candidate values including the $Z^{(t)}$ keep in the cell status $c^{(t)}$ [11].

By the forget gate determine which values in the cell status $c^{(t)}$ will remove or not according to the status of the network as following equation [11].

$$f^{(t)} = \sigma(W_f x^{(t)} + R_f y^{(t-1)} + p_f \odot c^{(t-1)} + b_f) \quad (3)$$

W_f, R_f, and P_f are weight values relevant to $x^{(t)}$, $y^{(t-1)}$,$c^{(t-1)}$, and b_f is the bias. σ is the activation function most of the time it is sigmoid [11].

The C cell determines what is the current cell value according to the input block $z^{(t)}$ and input gate value and previous cell value and forget gate value according to the following equation [11].

$$c^{(t)} = z^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)} \quad (4)$$

The output gate will control the output of the whole LSTM cell will determine by the input and the previous output of the LSTM cell and the current value of the cell status as following formula [11].

$$o^{(t)} = \sigma(W_o x^{(t)} + R_o y^{(t-1)} + p_o \odot c^{(t)} + b_o) \quad (5)$$

W_o, R_o, and P_o are the associated weight values of $x^{(t)}$, $y^{(t-1)}$,$c^{(t)}$ and b_o is the bias value and as previously σ is sigmoid as other gates [11].

Finally, the following equation gives the block output. The block output also uses the tanh as the block input as an activation function [11].

$$y^{(t)} = g(c^{(t)}) \odot o^{(t)}. \quad (6)$$

3.6 Dataset building with existing problems.

Each dataset has been studied before considering putting them into the general frame. When the dependent variables are set with merging columns they are not considered to add. Also, when the user has used a customized neural network other than the standard TensorFlow that is also ignored due to reduce the time of problem understanding. Also whether there is the dataset along with the notebook. The considered problems as in the following table.

Table 3.1: Datasets considered for adding features to the general frame

No	The problem	ANN	RNN
1	Rain Prediction: ANN [51]	Yes	No
2	Keras with breast cancer data[ANN] [52] …….. Making Model for Binary Classification	Yes	No
3	Single RNN with 4 folds (CLR) [53]	No	Yes
4	Google stock price prediction – RNN [54]	No	Yes
5	Heart Failure Prediction: ANN [55]	Yes	No
6	Mushroom Classification with ANN [56]	Yes	No
7	Predicting BTC Price Using RNN [57]	No	Yes
8	Associated Model RNN + Ridge [58]	No	Yes
9	Customer Churn Prediction Using ANN [59]	Yes	No
10	Predicting House Prices (Keras - ANN) [60]	Yes	No
11	Fake News Detection Using RNN [61]	No	Yes
12	Lyrics Generator: RNN [62]	No	Yes
13	Credit Card Fraud Detection ANNs vs XGBoost [63]	Yes	No
14	ANN starter [64]	Yes	No
15	Deep Learning For NLP: Zero To Transformers & BERT [65]	No	Yes
16	RNN_Detailed Explanation_[0.2246] [66]	No	Yes
17	Who said this line[EDA/Classification/Keras/ANN] [67]	Yes	No
18	[ANN/SLP] Making Model for Multi-Classification [68]	Yes	No

19	PLAsTiCC RNN [69]	No	Yes
20	Deep Learning Multivariate RNN / LSTM Network [70]	No	Yes
21	ANN on Electricity Consumption [71]	Yes	No
22	Minimizing Risks for Loan Investments Keras – ANN [72]	Yes	No
23	Hourly energy consumption time series RNN, LSTM [73]	No	Yes
24	Sarcasm Detection: RNN-LSTM Python · News Headlines [74]	No	Yes
25	Titanic Prediction – ANN [75]	Yes	No
26	Healthcare-dataset-stroke-data [76]	Yes	No
27	Comparison of ML models with RNN [77]	No	Yes
28	Botnet Host Prediction using RNN [78]	No	Yes
29	Autistic Patients EDA + Classification Using ANN [79]	Yes	No
30	Heart Attack Prediction | EDA | ANN [80]	Yes	No
31	Sentiment Analysis with RNNs - Disaster Tweets [81]	No	Yes
32	Transformer-RNN-Tool-Box-for-Text-Classification [82]	No	Yes
33	hotel-reservations [83]	Yes	No
34	ml_ann_ch(fuel) [84]	Yes	No
35	Sentiment Analysis(ML & RNN) [85]	No	Yes
36	Simple LSTM for text classification [86]	No	Yes
37	Travel Customer Churn Prediction using simple ANN [87]	Yes	No
38	Credit analysis with KNN/DTree/RF/Bagging/ANN [88] Page:3	Yes	No
39	Sentiment Analysis on Amazon Product (RNN-97% Acc) [85]	No	Yes
40	Hate speech detection: RNN [89] Page:5	No	Yes

3.7 Application programming interface

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® Distribution that allows you to launch applications and manage conda packages, environments, and channels without using command line interface (CLI) commands. Anaconda Navigator has been used as the application programming environment. It facilitates easy library installation with existing libraries in it.

JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality. In this research project, all the implementation has been done in Jupiter notebook. It also has some drawbacks such as when a program is running for a long time the program will automatically get paused due to the computer goes ideal mode. However, it is overcome by playing a video in the background.

3.8 Summary

In this chapter, discussed the various technologies that used to accomplish the research projects and underlying mathematical models for ANN and RNN. In the next chapter, will draw attention to the approach to accomplish the objectives of the project.

CHAPTER 4

APPROACH

4.1 Introduction

In the previous chapter, had to explain the usage of the technologies and underlay mathematical models. This chapter explains how to approach a solution for achieving an automated generation of neural network configuration. Different experts configure neural networks in different ways. So achieving a task in an automated way is a harder challenge due to these different possibilities. As mentioned in the introduction chapter there are no databases of dependent and independent features stored so in the initial phase of the research have to create such a database using the support of NLP distance parameters.

The sample selection for selecting the problems for creating the above database was systematically done by finding the best website for learning problems through a Google search and finding that the number one site is “Kaggle”. The problem search was done in Kaggle using the keywords “ANN” and “RNN” and the first acceptable 20 problems were selected for supervised learning. In the above the acceptable problems imply the removal of problems with multiple dependent variables and customized neural networks that result in merge layers. Moreover, the three testing data sets in the hotel domain were used in 7 different ways for testing that is not in training samples.

To store the features in the independent and dependent variable database in such a way that same or closer distance variables are arranged in the same column while independent and dependent variables are placed in different groups of columns. Because these features are different from each other even in the same column it can’t use ordinary encoding techniques even though there are many encoding techniques [33].

The attempt to tackle the problem can result in achieving the naval encoding technique it can be named NLP encoding. NLP encoding is achieved by measuring the distance between the independent and dependent variable, for encoding independent variables and the distance between the empty variable and dependent variable to achieve encoding for dependent variables.

After progressing with the solution of the above way that could predict the model of selection between ANN and RNN. Then to create a database of hyperparameters also will be a challenge because different people use different way of configurations. To proceed to the next step is to select two templates of ANN and RNN to proceed to the next level. If there are different configurations, then must make them in a similar configuration to put them in a less complex database.

4.2 Hypothesis

Hypnotized automation in the model selection between ANN and RNN through hyperparameter prediction by feature selection in the hotel domain. This will result in solving problems in the hotel domain with the support of deep neural networks though hyperparameter prediction will be far easier for the user due to the user not needing to worry about the configuration and optimality of hyperparameters.

4.3 Input

Give the location of the CSV files that contain the data sets and the location for saving the base files such as generated data frames as input with the dependent variable that has been put into a list. If there are multiple files, then the user must give the dependent variable in the above-mentioned list to the corresponding dataset. In addition, encoded data frames with dataset-independent and dependent variables related to the current problem and unencoded dependent variable columns to determine whether the problem is regression or classification, and unencoded text columns to determine embedding hyperparameters must be given as inputs.

4.4 Output

After generating hyperparameters, the neural network's structure can be obtained as an output. In addition, the evaluator and the predictor for the generated neural network training for predicting values according to the trained model can be obtained as outputs.

4.5 Process

Using the dataset location and the dependent variable list general frame that had already been discussed is created with the help of a preprocessing unit for preprocess feature headings. Then using the ANN-RNN classifier it will determine whether the inserted dataset problems are ANN or RNN. Then if a specific problem is an ANN

problem, then those problems feature values including the nature of the neural network save to ANN's general frame. Next by using the existing columns of the ANN general frame it predicts the next hyperparameter as the number of layers by using the hyperparameter predictor. The return prediction of many layers related to each ANN problem is then evaluated for restrictions. If there are any anomalies they are fixed and saved to ANN general frame in a column called number of layers. Then by using the updated ANN general frame, it will predict the next hyperparameter by using the hyperparameter. Then the result will be checked by the restrictor and if there are any anomalies they will fix and save them to the ANN general frame as a new column. This process will happen for all 34 hyperparameters for ANN. For the RNN problems, the same process happens by using the RNN general frame, Hyperparameter predictor, and the RNN model restrictor. After predicting all the hyperparameters by using the configuration loader that will take each dataset, an unencoded dependent variable column for each problem, and unencoded text columns, if there are any text classification or predictions for loading the configuration using the ANN and RNN data frames and pre-created ANN and RNN layers, are blended to generate neural network structures for each of the input problems. After this, each generated configuration will be trained using each dataset and training by already predicted training hyperparameters using the model runner. After using the model runner for each problem it can use the evaluation and predictor method for each problem for evaluate and predict the values related to each problem.

4.6 Users

This general model can be used for classification and prediction in the hotel domain. The general model is not confined to any domain, so it gives good results even if you use it in any other ANN application with RNN application together. But the training set only has 40 problems. Sometimes there can be problems due to no available past data to properly predict the network type and other hyperparameters. Furthermore, other intermediate outputs can be used such as the general frame can be used to fully automate hyperparameters. And the possibility of the usage of the general frame as an NLP classifier. The general frame may be used with SPSS as a supporting tool. NLP-based encoding for very diverse data columns as an alternative to mean encoding or label encoding.

4.7 Features

The general model can be used to configure the ANN and RNN neural networks without worrying about the network type. You can easily update the classifier and use more advanced classifiers such as nested classifiers. The classifier and the general frame can handle up to 50 independent and 10 dependent variables.

4.8 Summary

While going through this chapter has saw how you can start to develop ANN and RNN auto-model configurations. Finally, the users of the general model created by this research project and its features. In the next chapter, will navigates through the design of the general model.

CHAPTER 5

DESIGN

5.1 Introduction

In the previous chapter, it came across how it can have an approach for developing a general model that can predict hyperparameters for both ANN & RNN together enabling automated model selection. In this chapter, that will look further into the design of the program. An overview of the design of the general model is shown in the following diagram.

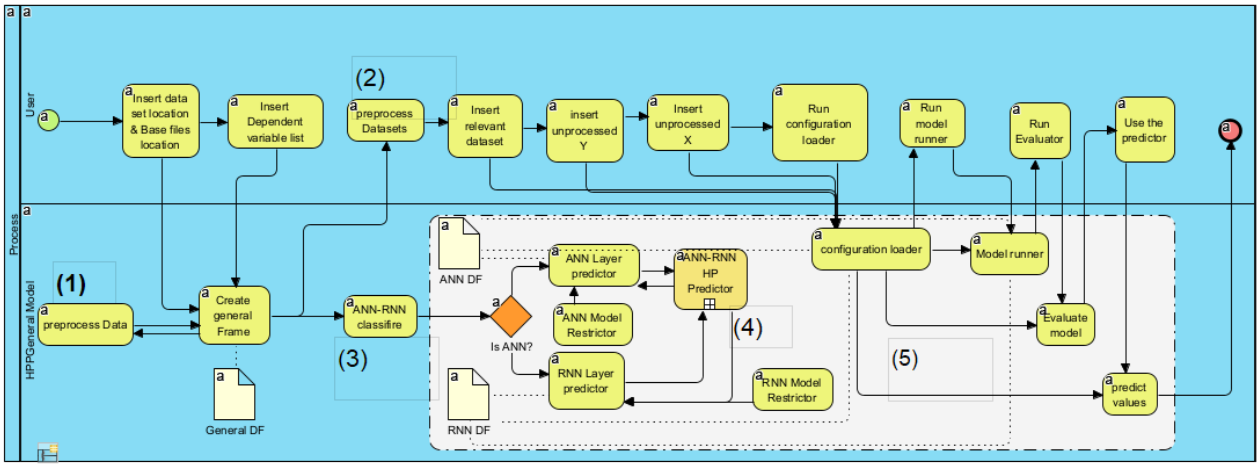


Figure 5.1:Overview of the Design diagram of the general model

The module (1) in Figure 5.1 is the preprocessing module for the feature set that will be sent to the classifier. Preprocessing such as removing special characters has to be done by it. The data set will have to be preprocessed as usually trained a neural network to fulfill a certain objective. This is accomplished by module (2) preprocessing. module (3) is the classifier module. This module selects which of the RNN or ANN networks will be selected according to the nature of the feature set. The Module (4) is the module that predict the hyper parameters according to the feature set that will send to the main module by prediction of 12 basic hyperparameters extended to 32 models with layer wise predictions. The module (5) is main and final module that implements the ANN or RNN module that will generate the required layers and compile the model in to the modle variable. Moreover, the general model provides the model variable that can be used for the predictions.

As already mention in the chapter four, the charateristics of the dataset in the approach chapter features of selected 20 ANN and 20 RNN had used as a supervised dataset to train 34 hyperparameters. Moreover, for the testing separate 3 datasets had used with 6 problems. Because of already discussed about the data set in the approach chapter it is not vise to explain everything again in here hence more of a summary has stated in here.

5.2 Module 01 & Module 02: Preprocessing

This preprocessing module01 is used to remove the special characters from the feature names before comparing them by NLP. Further, the same module 01 introduce spaces when camelCase feature names have detected consistently formatting all feature names.

Module 02 performs the usual preprocessing done to the data set such as removing data columns if there are many null values. Search for outliers and remove them. Check for the correlation and remove other correlated columns by keeping one correlated column in the specific area in the heatmap.

5.3 Module 03: ANN-RNN Classifier

Usually, useing a classifier when having a lot of data. For the feature set data are available but only as headings of data sets. There wasn't a way to acquire the feature sets of various problems in one frame or data table. So, to fulfill that requirement a general frame is created.

5.3.1 General frame

The general frame consists of 50 columns to store the independent variables and 10 columns to store the dependent variables. The feature values should be stored in a way such that similar feature names are in the same column. Usually, a column of a data set has some kind of relationship in domain or semantics with its column headings. But in the general frame, there isn't such a relationship. So, the approach was when getting a new row to the general data frame it will store an item comparing the columns' previous values and locate and store at the most appropriate column. So, this arrangement can be another secret of neural networks classifying well. By using the NLP library, it easily measured the similarity of two phrases, usually, that is two

feature headings at a time. The general frame is arranged in the following structure for training as in table 5.1.

Table 5.1:General frame Structure

indV01	indV02	indV03	indV04	depV01	depV02	Network type

When adding a new row, it happened as described in the following flow chart Figure 5.2.

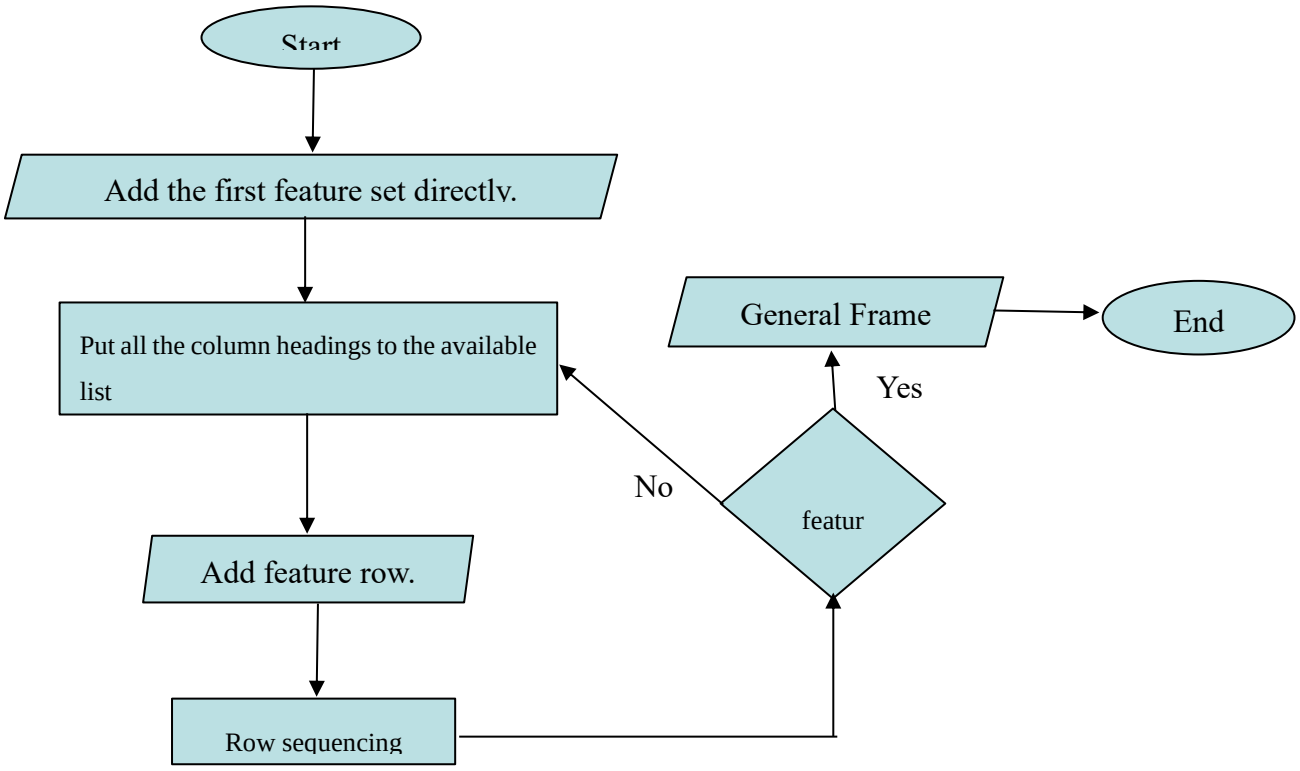


Figure 5.2:Row adding process to the general frame.

Since there are no existing rows at the beginning the first row can be added directly. Then after the second row, it added all column headings to a list named available columns list then the row sequencing process happens. At the row sequencing feature

values are re-arranging their best position according to the best match column selected by text comparison. Then it will be added while the added column will be removed from the available column list. So, the available column list makes sure that no two values are added to the same column in the same row. After adding a row, it looks at whether there is another row to add if so, again make all columns available, and the process of row sequencing start. Otherwise, it returns the general frame with the newly added row. The detailed process of row sequencing is illustrated in the following flow chart.

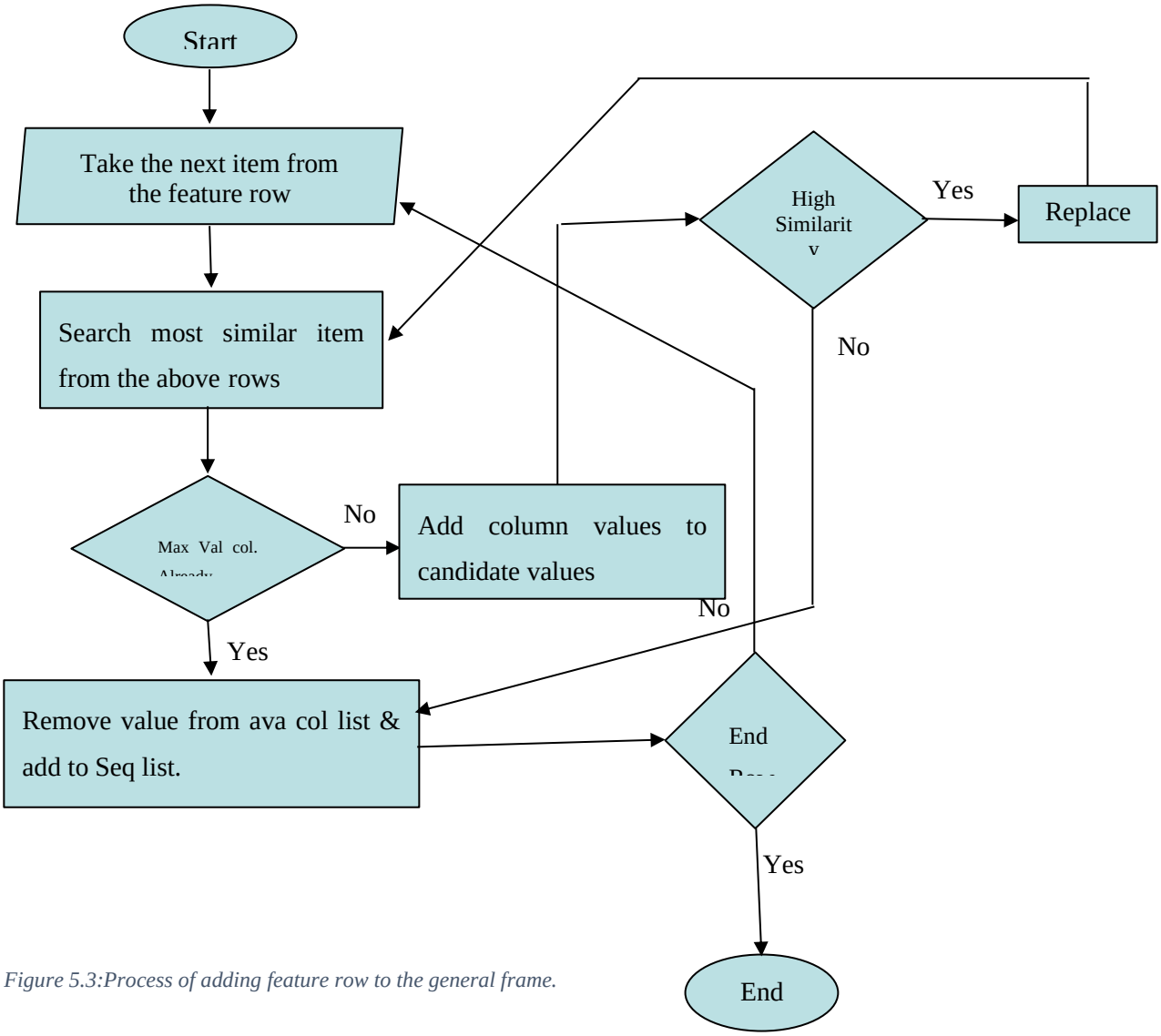


Figure 5.3:Process of adding feature row to the general frame.

The first item in the row that is going to add to the general frame gets from the list and then compares its similarity with values in all the above rows as in Figure 5.3. The value returns as 1 if to feature heading is 100 similar. Hence find the column that most

similar value to the current time. If this is the first item, it is unlike to have an already assigned value. However, when one by one item adds to the columns there is an increased probability of existing already placed items. When such a situation occurs instead of replacing the item it is sent to a candidate list. And continues the check for the remaining columns. Then at the end, it may have a column already placed. However, it compares the similarity between all candidates' columns. And if there is a column currently place the item and see whether the current place is the most similar.

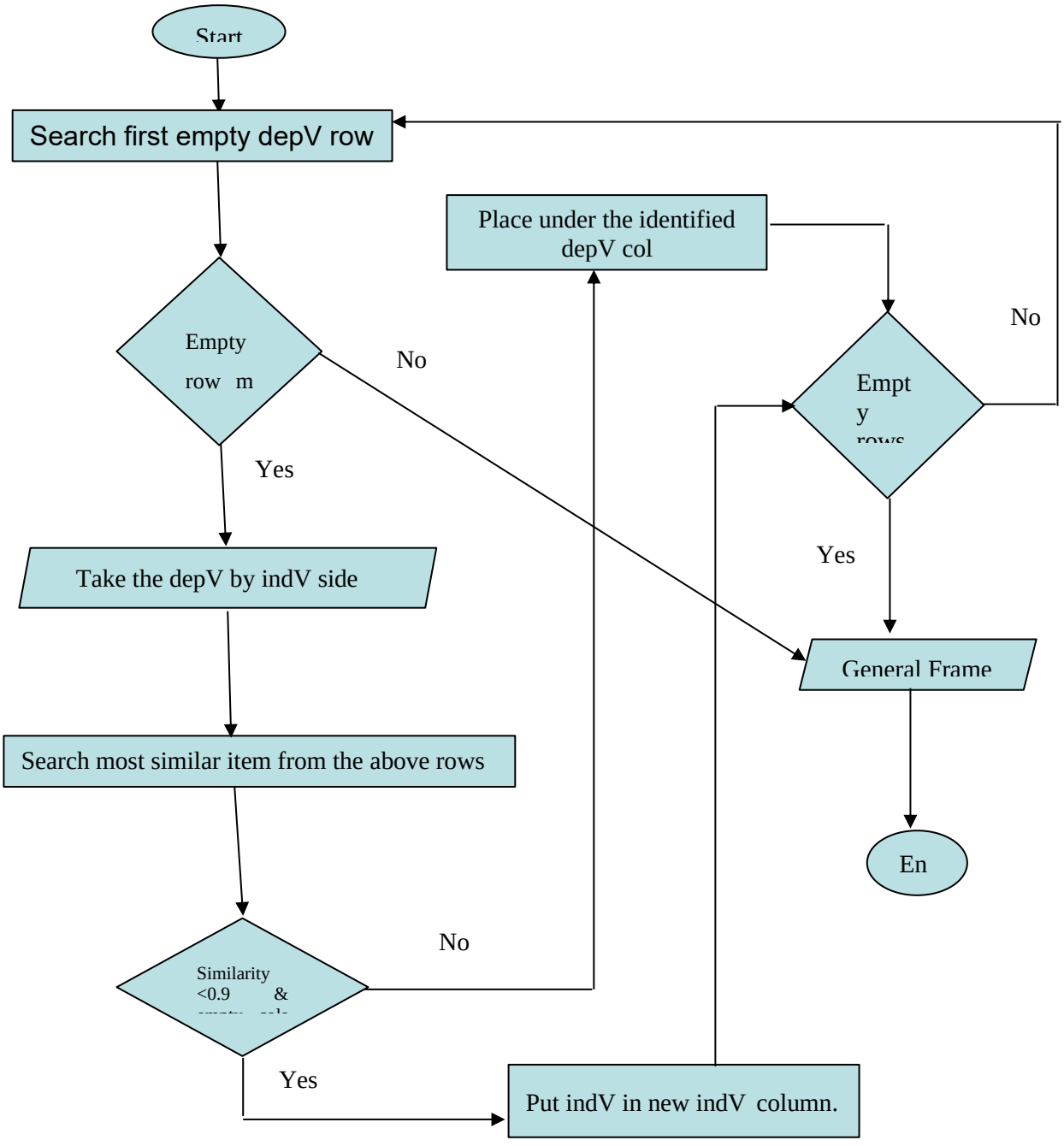


Figure 5.4:process of transfer dependent variable from independent side to dependent side

If so the value will keep in that column otherwise if the candidate column or one of the candidate columns has the most similar place the current item is placed in that candidate column taking the existing item back and researching the best similar place for that item. Then after all the items are arranged the row will be added. As explained previously, this arrangement happens in the independent variable section. As treat all the feature values as independent variables initially. The next step is taking the dependent variables that are in the independent section to the dependent section. It is performed as the above flow chart Figure 5.4.

First, it will search the first empty cells of a row on the dependent variable side then search the given dependent variable on the independent variable side then take the variable and check the similarity. If the similarity score is less than 0.9 and there are empty columns available, then the variable is put into the empty column that doesn't have previous dependent variable values. Otherwise put to the existing variable column that it most similar previous dependent feature value found. If there are no empty cell rows in the dependent section of the general frame, then it returns the general frame with arranged cell values at the dependent variables side.

5.3.2 Encoding general data frame

After arranging values, the neural network type column has been added to the general frame which will consider the dependent variable of the classifier. The next step is the conversion of the nominal values into integer values by use of encoding techniques. The label encoding by manually fitting is not successful when very much diverse values contain in each column. However, the two encoding methods are possible to use to accomplish it.

The design of the encoder class that is in the classifier is as follows.

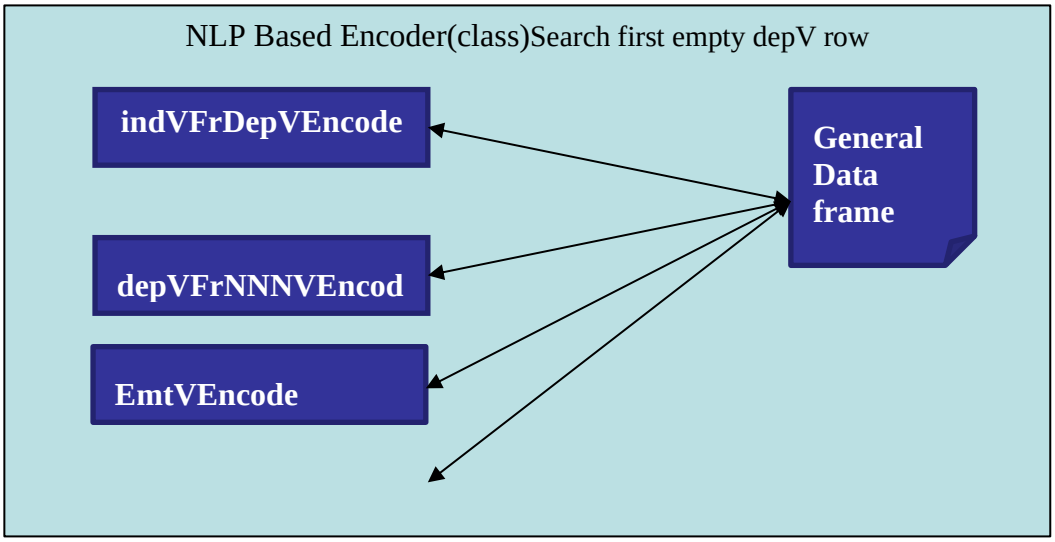


Figure 5.5:Encorder class

fullLableEncodee

The usage of each method is described in each section below.

5.3.3 NLP Encoding

The initial problem was that there was no proper encoding method to encode the values while preserving the semantics of the nominal features. Even using the mean encoding is not successful due to the diverseness being so much high and the occurrence of each value being so much low.

5.3.3.1 NLP encoding method 01

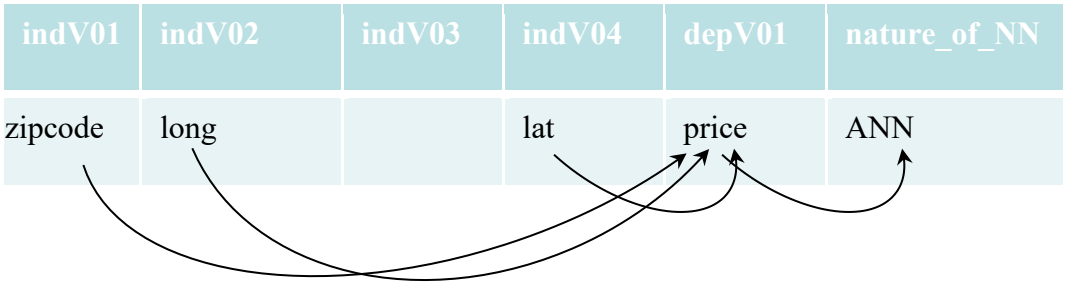


Figure 5.6:General frame

The similarity score is taken concerning the dependent variable for each row for each independent item. For the dependent variable, the similarity score between the dependent variable and the Nature of the neural network variable for each row has been taken. In the above figure 5.6 design indVFrDepVEncode and depVFrNNNVEncod should be used to achieve the encoding described above.

5.3.3.2 NLP encoding method 02

Similarity scores of all the variables concerning to single empty string have been taken. So, this approach is something like ratio encoding. Concerning NLP similarity scoring. In the encoding class EmtVEncode, the method can achieve this encoding method

5.3.4 Label encoding with random fit

The net possible standard encoding method is label encoding with random fitting values which will be selected the fitting values randomly for transformation. This

random fitting normalizes the effect of labeling a large number of diverse values. Because for all the columns around 5-6 same value occurrences are there. In the above design “fullLableEncodee” can achieve this method of label encoding.

5.3.5 The neural network for ANN-RNN classifier

The neural network should classify whether the feature set input to it is relevant for the ANN network or RNN network as in the diagram below.

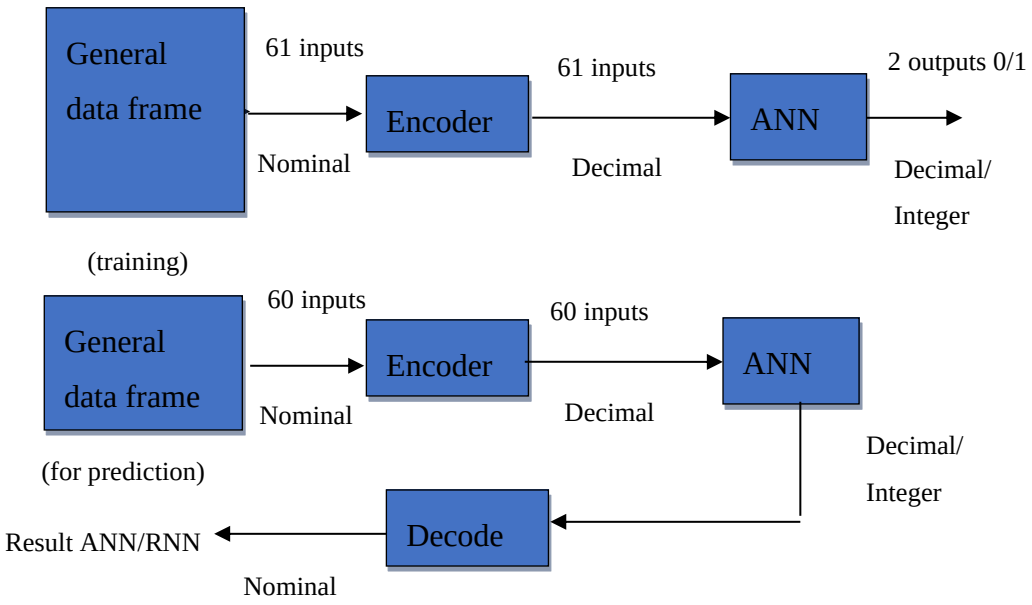


Figure 5.7Training and prediction process of the classifier

As in the above Figure, the 5.7 neural networks have trained with 61 inputs then the prediction has taken as in its lower process. So the neural network has 61 inputs with two outputs making it a binary classifier.

5.4 Module 04: Hyperparameter predictor module

This module receive the output from the ANN-RNN classifier and the predicting of hyperparamerts start with ANN-RNN classifier input and the selected feature input. The structure of the hyperparameter predictors appear as following diagram.

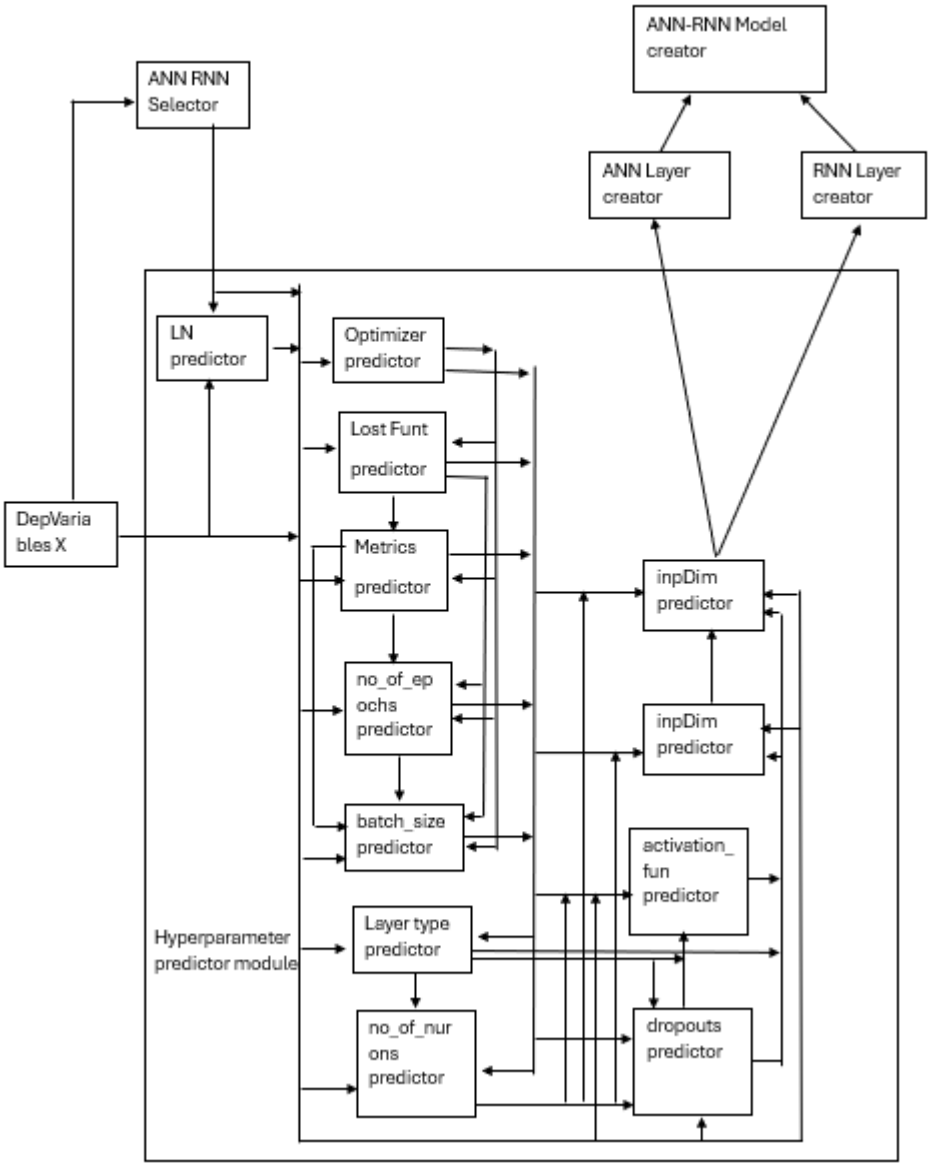


Figure 5.8 Structure of the hyperparameter predictor

The predictors are arranged from number of layer predictor. For each of the predictor uses all the outputs of all the previous predictors with the indepented feature set to predict the current hyperparameter.

5.4.1 Layer Number Predictor

The layer number predictor is the first predictor of the series of hyperparameters predictors. This is receiving the inputs from the independent lables and the ANN-RNN classifier only. Because that has to predict the layer number that can use regration for this predictor. It can use the mean squared error as the lost function and Adam as the optimizer.

5.4.2 Optimizer classifier

As the second hyperparameter predictor, this has the inputs specifically from the ANN-RNN classifier, layer number predictor and independent variables. Due to the optimizers are nominal they have to use the multi-class classifier. So in the output layer, that use the softmax function with categorial cross entropy as the lost function with adams optimizer and the accuracy metrix.

5.4.3 Lost function classifier

The lost functions are also nominal type values so have to use the multiclass classification as previously occurred. For the lost function, classifier receiving the outputs from the ANN-RNN classifier, layer number predictor, optimizer predictor and independent features. For this classifier it will have 7-8 classes like in the optimizer classifier too. So the predicted values will be so sensitive and can change in each training session due to low data.

5.4.4 Lost metrix classifier

Lust metrix predictor is yet another multiclass classifier that receive the outputs from the ANN-RNN calssifre, layer number predictor, optimizer predictor, lostfunction classifier. As other all classifier or predictors this classifier also recvies the independent feature set too.

5.4.5 Number of epochs predictor

Another important hyperparameter predictor is number of epochs. This predictor also receives it’s inputs from the outputs of ANN-RNN classifier, number of layers predictor, optimizer classifier,lost function classifier,lost metrix classifre. As the previous classifier or predictors this also redive the set of independent variables. Due to eporch is a numerical value that use regression to predict the eporches with the mean squared error lost function with Adams optimizer.

5.4.6 Batch size predictor

Batch size predictor is the last predictor that will generate a single value per each received data set. On the other words for each problem. From the next classifier the calssifire have to develop layer wise. So as before for the batch zise predictor also receives the inputs from ANN-RNN classifier, number of layers predictor, optimizer classifier, lost function classifier, lost metrix classifier, number of eporchers predictor.

As the previous predictor this is also a regression problem so that use the regression with mean squared error with the Adams optimizer.

5.4.7 Layer type classifier

Dispite previous classifiers or predictors layer type classifier is not a single classifier. It is a group of classifiers that develop for each layer up to 6 layers due to the selected two templates of ANN and RNN has nearly 6 layers. In this 6 classifiers are same in the structure but tained by layers wise data separately. Due to the less number of data that have to use the techniques such as data augmentation or dulplication. Out puts of the previous predictors with Independent feature set before this classifier development share all the layers wise layer type classifiers. The out puts of the 6 classifiers are merge together and provide as a single output to the upcoming classifiers and predictors that are develop after this layer type classifier. These six calssifires are multi calss calssifires that use categorical cross entropy as lost function with adams optimizer.

5.4.8 Number of neurons predictor

As per previous Layer type classifier the number of neurons predictor is a group of predictors that has develop to predict a separate model for each layer up to 6 layers. In the similar way to the previous classifier this predictor shair same structure of the nural network configuration with layer wise training and testing data sets. The merged output is given as the output of the total number of neurons predictor for the upcoming predictors or the calssifires. The next hyperparamerter for predicting is the input shape. The input shape is the number of features in the independent feature list so threre is no requirement to predict it so given the focus on the dropout predictor.

5.4.9 Dropout predictor

Dropout predictor predictor is yet another set of predictors as neurons predictor which has 6 pedictive models that has trained layer wise. Like the previous predictors this also give the merged out put of the 6 predictors to the next predictors and classifiers. These set of preditors also shairs the same neural network configuration with the mean squred error lost function and the adams optimizer.

5.4.10 Activation function classifier

This will be the last set of layer-wise predictors that classify which activation function is needed to apply to which layer. The classifier uses the outputs of the layer number predictor to the dropout predictor with the independent variables as it's inputs. Those inputs are the structure or the configuration has shaired between all the 6 layer-wise activation functions classifiers. Since this set of classifiers are nominal values rather than numerical values that have to develop them as multiclass classifiers.

5.4.11 Embeding Layer-Input dimension predictor

Not as previous predictors this predictor only work when there is a embedding layer is available. On the other hand the requirement of this predictor will be depend wether the user take the inputs from data preprocessor. As previous predictoers and classifiers this predictor also takes outputs from all the previous classifiers and the predictors mentioned in above with the independent variables.

5.4.12 Embeding Layer -Output dimention predictor

The last hyperparameter that going to predict for this research is the output dimention of the embedding layer. As per the input dimension predictor this predictor also work with the embedding layer. In the same way as input dimension predictor this predictor also takes the outputs of all above mention predicors and classifiers as inputs with the independent variables. However after this predictor accordin to the received ANN-RNN out put the hyper parameter set will be receive to the relevant layer creation menthod in the general module.

5.5 Module 05: General module

The general model is the main module that integrates everything to provide the final output which is model implementation according to the creating according to the the generated hyperparameters by the hyperparameters predictor mudule and create the model layers according to the relevant later creator between ANN and RNN layer creator with safety resrictions applied by the layer crator it self. Using the layers created by the ANN or RNN layer creator the modle creator will be create the model configuration give the output that can be use as model for the fit function. Henace can use for the relevant classification or the prediction.

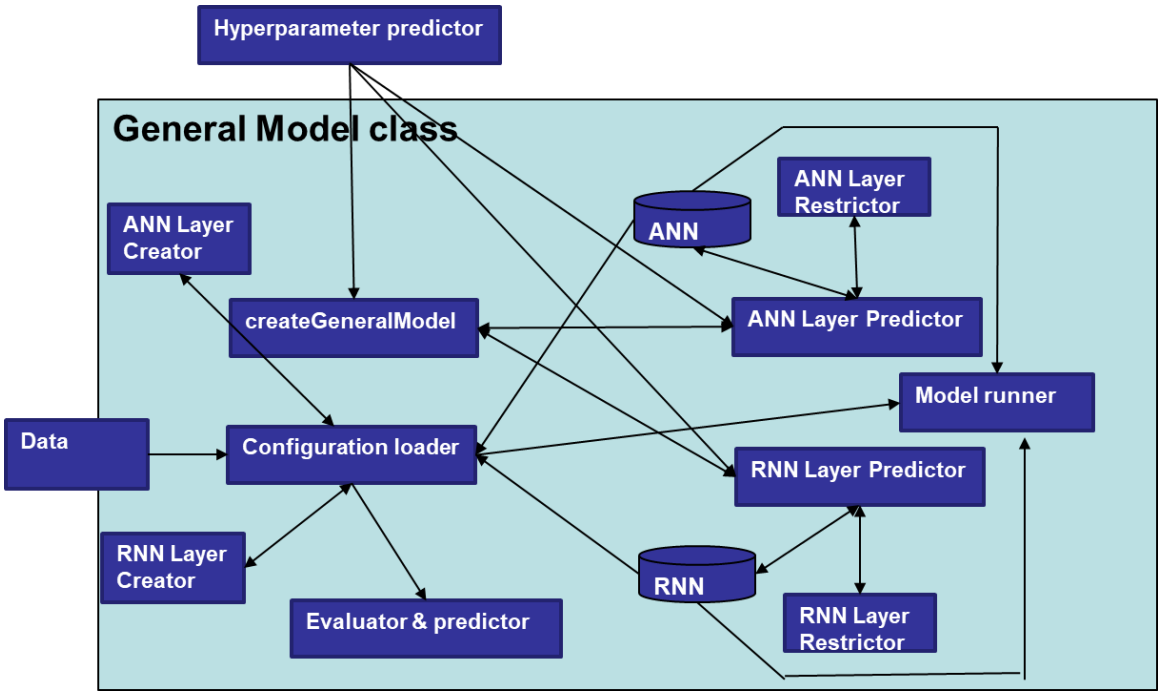


Figure 5.9:General module class

Each of the methods in the design is explained in the following topics.

5.5.1 Create a general model

In this model process the ANN and RNN layer predictions according to the available problem types. If there are no RNN problem this module let not RNN processors to turn on due to no data that is same for the ANN layers. Moreover, developers also can explicitly control the run of ANN and RNN processors for debugging purposes.

5.5.2 ANN & RNN Layer Predictors

ANN and RNN layer predictor modules are the ones responsible for saving the predicted values by hyperparameter predictor by observing the anomalies according to the ANN and RNN layer restrictors into the ANN and RNN general frames. Moreover, if there are anomalies in the current predictor and change the predictor for other those are already trained to predict the same hyperparameters before applying the final fixed value due to all the available predictors for the hyperparameter return anomalies is the task of these modules.

5.5.3 ANN & RNN layer restrictors

The predictions given by the neural network predictors are not 100% accurate sometimes due to the low number of training data the accuracy can be down to 40% for some or high-layer predictors such as layer 5 neural network type predictor. On that occasion, the remaining accuracy will be covered by a rule base preventing to generation of unrealistic structures and running hyperparameters.

5.5.4 ANN & RNN layer creators

The ANN & RNN layer creators contained the predefined TensorFlow layers with the variables for the hyperparameters such as activation function, and number of neurons per each layer denoted by the variables. The required layers for usual layer patterns have been covered except for merging layers not in the selected scope.

5.5.5 Configuration loader

Generating the layer structure by using the already predicted hyperparameters in the ANN general frame and RNN general frame using ANN and RNN layer creator while blending the parameters with the current problem-focused setting is the task of this module. Changes according to the nature of the problem such as the binary classifier or multiclass classifier or the regression problem are determined by this module. So, we can consider these as runtime configuration restrictions.

5.5.6 Model runner

The model runner will add the already predicted training parameters such as the number of epochs, batch size, optimizer, and performance matrix. Also, if there any anomalies they are handled by the model runner.

5.5.7 Evaluator and Predictor

After running the configuration and the model runner for a specific problem evaluator and predictor automatically change according to that problem. So evaluate the training while the predictor predicts values related to each problem that configuration and model runner runes.

5.5.8 User process for HPPGeneral model

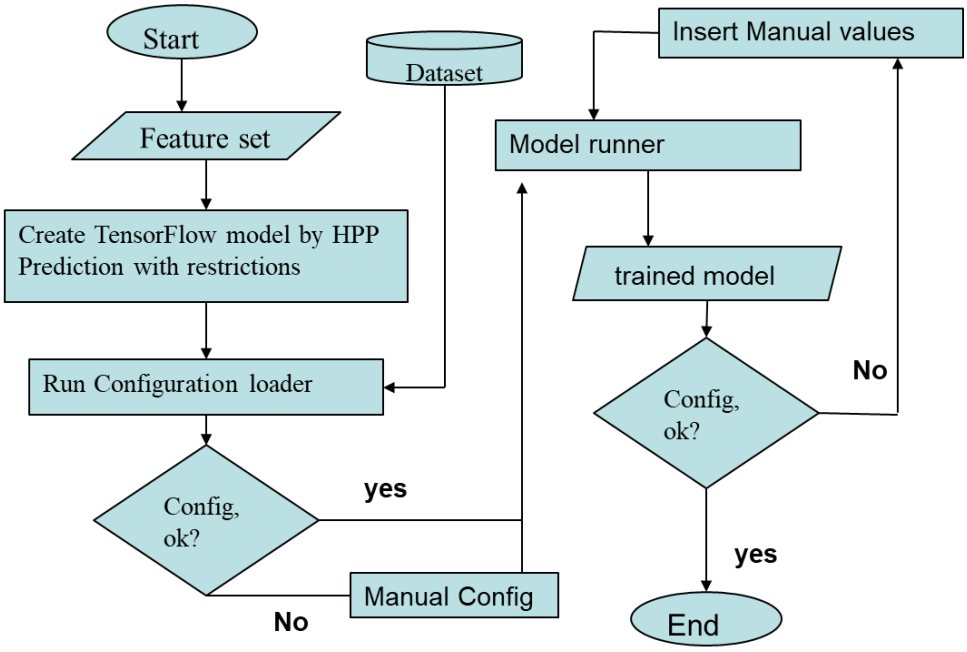


Figure 5.10 General model Creation

After the user enter the location of the feature set including the dependent variable list, unencoded Y and text column if any, HPP general model will be predict the hyperparameters related to all the problems. By using the configuration Loder user can load the neural network structure related to each problem. If the user happy with the existing configuration user can use it, otherwise user can copy the output given by the configuration loader then change it as desired and add to the model. Then the user can run the model runner to train the model with predicted training hyperparameters. If the user is not happy with predicted hyperparameters then they can manually enter the hyperparameters such as batch size and number of epochs.

5.6 Summary

In the design chapter by getting a good understanding of the function of each module in design and how they will create the final general model that will be able to solve ANN and RNN problems together. Now focus on the next chapter to see how these models have been implemented by using the relevant libraries and the tools that discussed in the technology chapter.

CHAPTER 6

IMPLEMENTATION

6.1 Introduction

The design discussed in the previous chapter has been implemented using Python Notebook and Anaconda Navigator as the API. This chapter will explain each of the modules and their relevant coding. Also, the screenshots of the output are included in the relevant locations. Moreover, from time to time relevant libraries have been installed by using an anaconda environment and using pip.

6.2 Module 01 & Module 02: Preprocessing

The following method has been used to preprocess the features that construct the general frame. It has been implemented as a method. The main method is cellValueFormat and it can make all text into lowercase and remove special characters from the feature names that will make the similarity score more realistic score values. This method uses another method namely camel_case_to_phrase which will convert the camelCase feature name without spaces to with spaces.

```
def camel_case_to_phrase(self,s):
    prev = None
    t = []
    n = len(s)
    i = 0

    while i < n:
        next_char = s[i+1] if i < n -1 else ''
        c = s[i]
        if prev is None:
            t.append(c)
        elif c.isupper() and prev.isupper():
            if next_char.islower():
                t.append(' ')
                t.append(c)
            else:
                t.append(c)
        elif c.isupper() and not prev.isupper():
            t.append(' ')
            t.append(c)
        else:
            t.append(c)
        prev = c
        i = i +1

    return "".join(t)
```

Figure 6.1:code for converting camelCase feature without spaces to spaces with lowercase

```
def cellValueFormat(self,item,spaceChar):
    item=str(item)

    if '_' in item:
        itemf=item.replace("_",spaceChar).lower()
    elif '-' in item:
        itemf=item.replace("-",spaceChar).lower()
    else:
        itemf=self.camel_case_to_phrase(item).lower()
        itemf=itemf.replace(" ",spaceChar)
    itemf=re.sub('\W+',spaceChar,itemf)
    return itemf
```

Figure 6.2:code to introduce lowercase with spaces

6.3 Module 03: ANN-RNN Classifier

The ANN-RNN is the key module in the general model because that is the model identifies which type of model from the ANN and RNN should be selected to be used according to the feature set. As mentioned before that created the dataset using the general frame to train the neural network.

6.3.1 General frame

As in the design the general frame is created as following coding in Figure 6.3 without the data if the CSV file for the data frame is not available in the target location.

```
try:

    filelist=os.listdir(dataDirectory)
    filelist=natsorted(filelist)
    self.newDataframe=pd.read_csv(csvFile,index_col=0)
except:
    print("Nofile or data directory exist. file will automatically created.Data dictinary default 'data/'")
    createf=True
if createf:
    dataframeHeadings=['file','indV01','indV02','indV03','indV04','indV05','indV06','indV07','indV08','indV09',
                       'indV10','indV11','indV12','indV13','indV14','indV15','indV16','indV17','indV18',
                       'indV19','indV20','indV21','indV22','indV23','indV24','indV25','indV26','indV27',
                       'indV28','indV29','indV30','indV31','indV32','indV33','indV34','indV35','indV36',
                       'indV37','indV38','indV39','indV40','indV41','indV42','indV43','indV44','indV45',
                       'indV46','indV47','indV48','indV49','indV50','depV01','depV02','depV03','depV04',
                       'depV05','depV06','depV07','depV08','depV09','depV10']
    self.newDataframe=pd.DataFrame(columns=dataframeHeadings)
else:
    print(csvFile,' reading...')
    self.newDataframe=pd.read_csv(csvFile,index_col=0)
```

Figure 6.3:code for creating general frame structure.

self.newDataframe=pd.DataFrame(columns=dataframeHeadings) is the code for creating the general data frame

add the first row directly and if this is not the first row the column sequence will sequence the items according to best-suited location.

```
if self.newDataframe.empty:
    cs=ColumnSeequancer()
    feacherIndex=0
    for feacher in feacherList:
        feacher=cs.cellValueFormat(feacher,"_")
        feacherList[feacherIndex]=feacher
        #print('feacher:',feacher)
        feacherIndex+=1
    self.newDataframe.loc[index]=[file]+feacherList
else:
    Seq=[]
    print('Items sequencing...')
    sq=ColumnSeequancer(Seq,self.newDataframe)
    self.newDataframe.loc[index]=[file]+sq.seequanceRow(feacherList)
```

Figure 6.4:Code for adding the first row directly otherwise sequence the items.

The file name column will be added separately at last.

6.3.1.1 Sequence Row

The sequence row is the main method of the ColumnSeequancer class that will run the sequence. It loads the values to the rowItems list of the class then takes one by one item from the list to compare them with the existing values as below.

```
def seequanceRow(self,rowItems=[]):
    self.rowItems=rowItems
    #self.df.columns
    self.maxCmpValues=[]
    self.remItems=[]
    i=0
    self.avaCols=self.df.columns.to_list()
    print("rowItems:",rowItems)
    for rowItem in rowItems:
        maxCmpVal=0
        maxCmpCol=""
        self.brk=0
        self.ava=0
        selOtherCol=False
        print("Item:",rowItem)
        #print("rowItems:",rowItems)
        print("toSeq01:",self.toSeq)
        self.maxCmpCols=[]
        self.compVals=[]
        self.compCols={}
        self.candidateCols={}
        self.shiftValues=[]
        columns=self.df.columns.to_list()
        columnIndex=0
        print(len(columns))
```

Figure 6.5: code to take each item from the item list that waiting to add to the general frame.

Check each item with rows as Figure 6.6.

```
while columnIndex<len(columns):
    if columns[columnIndex]=="file" or columns[columnIndex]=="Unnamed: 0":
        columnIndex+=1
        continue
    print("column:",columns[columnIndex])
    for colItem in self.df[columns[columnIndex]]:
        #print("colItem:",colItem)
        compaireVal=self.compaireContents(colItem,rowItem)
        self.compVals.append(compaireVal)
        self.compCols[compaireVal]=columns[columnIndex]
        print('columnIndex:',columnIndex,'compaireVal:',compaireVal,' maxCmpVal:
        if compaireVal>maxCmpVal:
            if columns[columnIndex] in self.avaCols:
                maxCmpVal=compaireVal
                maxCmpCol=columns[columnIndex]
                #print("maxCmpVal assined")
                selOtherCol=False
```

Figure 6.6: Code to compare the maximum comparison value and update it if the current value is high

Output created by the algorithm in Figure 6.7:

```
column: indV01
targetDItem: date sourceDItem: comment_text
columnIndex: 1 compaireVal: 0.4227598906826712  maxCmpVal: 0  maxCmpCol:
maxCmpVal: 0.4227598906826712 & maxCmpItem for  comment_text
rowItems index: 0
avaCols index: 1
toSeq index error
targetDItem: symmetry_worst sourceDItem: comment_text
columnIndex: 1 compaireVal: 0.10521451472760458  maxCmpVal: 0.4227598906826712  maxCmpCol: indV01
targetDItem: nan sourceDItem: comment_text
columnIndex: 1 compaireVal: 0.2835896285702727  maxCmpVal: 0.4227598906826712  maxCmpCol: indV01
targetDItem: date sourceDItem: comment_text
columnIndex: 1 compaireVal: 0.4227598906826712  maxCmpVal: 0.4227598906826712  maxCmpCol: indV01
```

Figure 6.7:The output created by the code.

compaireVal=self.compaireContents(colItem,rowItem) is the function that compares and gives the score to the compaireVal function

Then by the if compaireVal>maxCmpVal: it stores the maximum score in the maxCmpVal variable while it searches for a location for each item in the rowItems list.

As explained before the condition of columns[columnIndex] in self.avails: check whether the location of the maximal already has a value or not. If it is available only it replaces the maxCmpVal.

The following code is related to what happens if there are value exists in the palace.

```
    #print("maxCmpVal assined")
    selOtherCol=False
else:
    preCompVal=self.maxCmpValues[self.toSeq.index(columns[columnIndex])]
    print('old cmp Val:',preCompVal,' New cmp Val:',compaireVal)
    if preCompVal<compaireVal:
        self.candidateCols[compaireVal]=[columns[columnIndex],preCompVal]
        self.shiftValues.append(compaireVal)
        selOtherCol=True
        print('self.shiftValues:',self.shiftValues,'self.candidateCols[compaireVal]:',s

    else:
        if not(selOtherCol) and not(maxCmpCol in self.avaCols) :

            print('compaireVal:',compaireVal,'compVals:',self.compVals)
            for preVal in self.compVals:
                if (compaireVal>preVal) and (self.compCols[preVal] in self.avaCols):
                    maxCmpCol=self.compCols[preVal]
            if maxCmpVal==0 and columnIndex+1==len(columns):
                maxCmpCol=self.avaCols[1]
```

Figure 6.8:Code for the action when value is already existed in the place, shifting values or not

Visual output:

```
columnIndex: 2 compaireVal: 0.25718609291347516  maxCmpVal: 0.43613671203521365  maxCmpCol: indV01
targetDItem: zipcode sourceDItem: comment_text
columnIndex: 2 compaireVal: 0.33219223978837  maxCmpVal: 0.43613671203521365  maxCmpCol: indV01
targetDItem: title sourceDItem: comment_text
columnIndex: 2 compaireVal: 0.43646750424582836  maxCmpVal: 0.43613671203521365  maxCmpCol: indV01
maxCmpVal: 0.43646750424582836 & maxCmpItem for  comment_text
rowItems index: 0
avaCols index: 2
toSeq index error
targetDItem: title sourceDItem: comment_text
columnIndex: 2 compaireVal: 0.43646750424582836  maxCmpVal: 0.43646750424582836  maxCmpCol: indV02
```

Figure 6.9:Output when a higher comparison value found

If preCompVal<compaireVal: in the above code in figure 6.9 determined whether the score of the currently existing value is higher or lower than the attempted value. If the item currently in the location has a lower score, then that cloud be replaced after checking other cells so the value is put into the candidate list. There are some occasions the highest value of the attempting value is lower than the already existing value, but it is the highest similarity value after checking with all previous values in all columns so in that case the height attempting value makes lower than the next lower value which is available in the available columns list. Inside of if not(selOtherCol) and not(maxCmpCol in self.avaCols): condition and its contents are for that.

After finishing comparing all the previous values, that have to check the candidates' locations (columns) that have previously been saved as mentioned before. The following code in Figure 6.10 accomplishes it.

```
if(columnIndex+1==len(columns)):
    print('self.shiftValues:',self.shiftValues)
    for shiftValue in self.shiftValues:
        if maxCmpVal<shiftValue:
            maxCmpVal=shiftValue
    if maxCmpVal in self.shiftValues:
        print('rowItems Before shifted:',rowItems,' compaireVal:',maxCmp
        currItemIndex=self.rowItems.index(rowItem)
        oldItemIndex=self.toSeq.index(self.candidateCols[maxCmpVal][0])
        tempItem=rowItems[oldItemIndex]
        #self.avaCols.append(column)
        #print('tempItem:',tempItem,' rowItem:',rowItem)
        rowItems[oldItemIndex]=rowItem
        #print('current Item index:',currItemIndex,'rowItem-current Item
        rowItem=rowItems[currItemIndex]=tempItem
        self.maxCmpCols.reverse
        tempMaxCmpVal= self.candidateCols[maxCmpVal][1]
        tempCol=self.candidateCols[maxCmpVal][0]
        print('selected Column:',tempCol)
        del self.candidateCols[maxCmpVal]
        self.candidateCols[tempMaxCmpVal]=[tempCol,maxCmpVal]
        self.maxCmpValues[oldItemIndex]=maxCmpVal
        maxCmpVal=tempMaxCmpVal
```

Figure 6.10: Code for checking candidate values to consider for a shift

Visual output:

```
columnIndex: 60 compaireVal: 0.3503796151186468  maxCmpVal: 0.497444831124821  maxCmpCol: indV05
self.shiftValues: [0.5755558654297235, 0.5259693602364891]
rowItems Before shifted: ['id', 'name', 'asins', 'brand', 'categories', 'keys', 'manufacturer', 'reviews.date', 'reviews.date
Added', 'reviews.dateSeen', 'reviews.didPurchase', 'reviews.doRecommend', 'reviews.id', 'reviews.numHelpful', 'reviews.ratin
g', 'reviews.sourceURLs', 'reviews.text', 'reviews.title', 'reviews.userCity', 'reviews.userProvince', 'reviews.username', na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]  compaireVal: 0.5755558654297235  preCompVal: 0.338210855
36089773
selected Column: indV11
Values shifted
rowItems After shifted: ['id', 'name', 'asins', 'brand', 'categories', 'keys', 'manufacturer', 'reviews.date', 'reviews.dateA
dded', 'reviews.dateSeen', 'reviews.didPurchase', 'reviews.doRecommend', 'reviews.id', 'reviews.rating', 'reviews.numHelpfu
l', 'reviews.sourceURLs', 'reviews.text', 'reviews.title', 'reviews.userCity', 'reviews.userProvince', 'reviews.username', na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]  compaireVal: 0.33821085536089773  preCompVal: 0.57555586
54297235
restarted...
column: indV02
```

Figure 6.11:Visual output when shifting occurs.

By the “if(columnIndex+1==len(columns)):” last iteration is detected then by the loop code “for shiftValue in self.shiftValues:” I will look at whether there are candidate locations that give a higher maximum value than the current maximum value if there are then the values are interchanged then research from the beginning for the changed value. The interchange happens in the row item list which holds the items to be added. The column is assigned to the item by removing it from the available list and placing it in the seq list. It is shown by the following code.

```
if self.ava==1 and len(self.avaCols)>1 and self.avaCols[1]!="file":
    print("Direct append & Direct remove:",self.avaCols[1])
    self.toSeq.append(self.avaCols[1])
    self.avaCols.pop(1)
    self.maxCmpValues.append(maxCmpVal)

else:
    self.toSeq.append(maxCmpCol)
    self.maxCmpValues.append(maxCmpVal)
    #print('toSeq:',self.toSeq)
    #print('avaCols:',self.avaCols,"maxCmpCol:",maxCmpCol)
    self.avaCols.remove(maxCmpCol)
print("Avilable columns:",self.avaCols)
#rowItems.remove(maxCmpItem)
```

Figure 6.12: Code for adding the item to the sequence list by removing it from the available list.

Specifically, it happens by following codes.

self.toSeq.append(maxCmpCol)

self.maxCmpValues.append(maxCmpVal)

self.avaCols.remove(maxCmpCol)

the visual output of removal and addition

```
self.shiftValues: []
Avilable columns: ['file', 'indV03', 'indV04', 'indV05', 'indV06', 'indV07', 'indV08', 'indV09', 'indV10', 'indV11', 'indV1
2', 'indV13', 'indV14', 'indV15', 'indV17', 'indV18', 'indV19', 'indV20', 'indV21', 'indV22', 'indV23', 'indV25', 'indV26',
'indV27', 'indV28', 'indV29', 'indV30', 'indV31', 'indV32', 'indV33', 'indV34', 'indV35', 'indV36', 'indV37', 'indV38', 'indV
39', 'indV40', 'indV41', 'indV42', 'indV43', 'indV44', 'indV45', 'indV46', 'indV47', 'indV48', 'indV49', 'indV50', 'depV01',
'depV02', 'depV03', 'depV04', 'depV05', 'depV06', 'depV07', 'depV08', 'depV09', 'depV10']
Item: categories
toSeq01: ['indV24', 'indV01', 'indV16', 'indV02']
61
```

Figure 6.13:Visual output from removing an item a add the item to the sequence list.

After sequencing happens, that have the column order according to the item order of the adding row. Now it must arrange the columns in the order of the headings of the general frame. Because of know what the relevant value of each column in the sequence order is. It is performed by the following codes in Figure 6.14.

```
self.finalseq=[]
for colSeqI in self.df.columns:
    if colSeqI=="file":
        continue
    colIndex=self.toSeq.index(colSeqI)
    self.finalseq.append(self.cellValueFormat(rowItems[colIndex],"_"))
    print('colSeqI:',colSeqI," colIndex:",colIndex,' rowItemsw:',rowItems[colIndex])
```

Figure 6.14: Code for rearranging the feature items in the order of columns in the general frame.

Output for final arrangement:

```
colSeqI: indV07  colIndex: 8  rowItemsw: nan
colSeqI: indV08  colIndex: 9  rowItemsw: nan
colSeqI: indV09  colIndex: 10  rowItemsw: nan
colSeqI: indV10  colIndex: 11  rowItemsw: nan
colSeqI: indV11  colIndex: 12  rowItemsw: nan
colSeqI: indV12  colIndex: 13  rowItemsw: nan
colSeqI: indV13  colIndex: 14  rowItemsw: nan
colSeqI: indV14  colIndex: 15  rowItemsw: nan
colSeqI: indV15  colIndex: 16  rowItemsw: nan
colSeqI: indV16  colIndex: 0  rowItemsw: comment_text
colSeqI: indV17  colIndex: 17  rowItemsw: nan
colSeqI: indV18  colIndex: 1  rowItemsw: toxic
colSeqI: indV19  colIndex: 18  rowItemsw: nan
colSeqI: indV20  colIndex: 19  rowItemsw: nan
colSeqI: indV21  colIndex: 20  rowItemsw: nan
colSeqI: indV22  colIndex: 21  rowItemsw: nan
colSeqI: indV23  colIndex: 22  rowItemsw: nan
colSeqI: indV24  colIndex: 23  rowItemsw: nan
colSeqI: indV25  colIndex: 24  rowItemsw: nan
```

Figure 6.15:Out put of the rearranged list according to the order of the columns in the general frame

After running the values are arranged in the frame as following figure.

file	indV01	indV02	indV03	indV04	indV05	indV06
1.weatherAUS.csv	date	location	min_temp	max_temp	rainfall	evaporation
2.data.csv	symmetry_worst	area_se	compactness_worst	perimeter_worst	perimeter_se	concave_points_se
JN_with 4_folds_train.csv	NaN	NaN	NaN	NaN	NaN	NaN
Google_stock_pricePr.csv	date	NaN	NaN	NaN	NaN	NaN
iical_records_dataset.csv	age	time	NaN	NaN	NaN	serum_sodium
6.mushrooms.csv	NaN	habitat	population	cap_color	gill_spacing	cap_shape
2012-01-01_to_2017-10...	NaN	timestamp	NaN	NaN	weighted_price	NaN
l_Model_RNN_Ridge.csv	name	brand_name	item_description	NaN	price	NaN
9.Churn_Modelling.csv	age	geography	num_of_products	NaN	estimated_salary	credit_score
10.kc_house_data.csv	date	zipcode	long	lat	price	view
etection_Using_RNN.csv	date	title	NaN	NaN	NaN	NaN

Figure 6.16:appearance of the frame after adding 15 rows.

6.3.1.2 Transfer dependent variables into the dependent section(dependent variables)

In this part of the coding, that have to find the first row with 10 empty independent variables cells as below in Figure 6.17.

indV09	...	depV01	depV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	dep
ust_speed	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
ass_mean		NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Figure 6.17:general frame with empty cells in dependent variables side

It is done by the following code.

```
def emptyRowFinder(self,columnList=[],serchDf=pd.DataFrame):

    print("length:",len(serchDf.index))
    depVeriables=serchDf.columns[51:61].to_list()
    serchDf=serchDf[depVeriables]
    idx_first_empty_row=0
    searchStop=False
    #while index<len(serchDf.index):
    #for column in serchDf.columns.to_list():
    index=0
    for item in serchDf[depVeriables[0]]:
        #print("\n item: \n",item,"\n",'index:',index)
        #print('serchDf[',depVeriables[0],'][',index,']:',serchDf[depVeriables[0]][index],'serchDf[depVeriabl
        if str(item)==str(np.NaN):
            if(str(serchDf[depVeriables[0]][index])==str(serchDf[depVeriables[1]][index])==str(serchDf[depVer
                idx_first_empty_row=index
                searchStop=True
                break

            #idx_first_empty_row=index

        index+=1
    #print(len(serchDf.index)-1,":",index)

    if len(serchDf.index)==index:
```

Figure 6.18:code to find the first empty row in the dependent side of the general frame.

It will check when all the cells on the dependent sides are equal to each other. The only situation is when there is no dependent variable selected. After finding the empty 10 cells that have to find the relevant dependent variable on the independent side and put it into the dependent side. Since that have the dependent variable value that should transfer in the different list, find its location in the dependent side and set it to NaN

Then in the first row, it directly put the indV01 location for other rows it checks the similarity scores with the other value existing rows and if the similarity value is less than 0.9 and other columns are empty value will be put to the empty column it first finds. Otherwise, it will place the column where the most similar item is found. The following code performs the abovementioned tasks.

```
def dependentVariables(self,dependentList,originalDf=pd.DataFrame):
    depVeriables=self.df.columns[51:61].to_list()
    print('depVeriables:',depVeriables)
    self.dependentList=dependentList
    ##originalDf=pd.DataFrame(self.df.copy())
    dependentIndex=self.emptyRowFinder(self.dependentList,originalDf)
    #print('dependentIndex:',dependentIndex,'len(self.dependentList):',len(self.dep
    for dependentIndex in range(dependentIndex,len(self.dependentList)):
        column=originalDf.columns[originalDf.isin([self.dependentList[dependentInd
        originalDf[column].loc[dependentIndex]=originalDf[column].replace([self.de
        maxCompVal=0
        maxCompCol=''
        for depVeriable in depVeriables:
            nanColumn=True
            for clItem in originalDf[depVeriable]:
                compareVal=self.compaireContents(clItem,self.dependentList[depender
                if compareVal>maxCompVal:
                    maxCompVal=compareVal
                    maxCompCol=depVeriable
                #print('clItem:',type(str(clItem)),' str(np.nan):',type(str(np.NaN
                print('maxCompVal:',maxCompVal,'maxCompCol:',maxCompCol)
                if str(clItem)!=str(np.NaN) and maxCompVal!=1.0:
                    print('!')
                    nanColumn=False
            #print('nanColumn:',nanColumn)
            if (nanColumn) and (maxCompVal<0.9):
                maxCompCol=depVeriable
                print('maxCompVal(NaN):',maxCompVal,'maxCompCol(NaN):',maxCompCol)
```

Figure 6.19:Set of codes for transferring the dependent variables from the independent side to the dependent side.

The output after the task is done by the above code.

depV01	depV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	depV10
tomorrow	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	diagnosis	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	target	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	open	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	death_event	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	class	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	weighted_price	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	price	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	exited	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	price	NaN	NaN

Figure 6.20:Output after transferring dependent variables to the dependent side.

The result of this has been saved as a CSV if the CSV file existed it will load otherwise new CSV file creates as the following code.

```
def runDepVar(self,dependentList=['rain_tomorrow','diagnosis','target','open'],csvFile='GeneralFrameWithDepVars.csv'):
    createf=False
    try:
        dfDepSh=pd.read_csv(csvFile,index_col=0)
    except:
        print("Nofile or data directory exist. file will automatically created.Data dictinary default 'data/'")
        createf=True
    if createf:
        self.newDataframe.to_csv(csvFile)
        dfDepSh=pd.read_csv(csvFile,index_col=0)
    else:
        dfDepSh=pd.read_csv(csvFile,index_col=0)
    seq=[]
    #dependentList=dependentList[:3]
    sq=ColumnSeequancer(seq,self.newDataframe)
    dfFinal=sq.transerRows(self.newDataframe,dfDepSh)
    self.df2=sq.dependentVariables(dependentList,dfFinal)
    self.df2.to_csv(csvFile)
```

Figure 6.21:Code for saving the output as CSV or loading it if already existed.

Then as the final step neural network type column has been added because this is the training set by the following code.

```
def addComColumn(self,colName="",dataSet=[],genDF=pd.DataFrame):
    genDF[colName]=dataSet
    return genDF
```

Figure 6.22:Add column method to add a column to the general frame.

```
def addComColumns(self,nameList=[],datasetList=[],cSVfile='GeneralFrameWithDepVars.csv',cSVfile2='GeneralFrameWithComCo
    createf=False
    try:
        dfWithDepVars=pd.read_csv(cSVfile,index_col=0)

    except:
        print("No file or data directory exist.")
        createf=True
    seq=[]
    sq=ColumnSeequancer(seq,self.newDataframe)
    if createf:
        #self.df2.to_csv(cSVfile)
        dfWithDepVars=self.df2


    index=0
    print('datasetList[index]:',datasetList[0],'dfWithComCols:',dfWithDepVars.head())
    for name in nameList:
        finalDf=sq.addComColumn(name,datasetList[index],dfWithDepVars)
        finalDf.head()
        index+=1
    dfWithDepVars.to_csv(cSVfile2)
    return finalDf
```

Figure 6.23:this make allow the addition of one or more columns to the general frame at once.

The above code in Figure 6.23 facilitates adding more than one column with its data set at once. After running the above code, the following output is taken.

indV09	...	depV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	depV10	nature_of_NN
ıst_speed	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	ANN
ess_mean	...	diagnosis	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	ANN
NaN	...	NaN	target	NaN	NaN	NaN	NaN	NaN	NaN	NaN	RNN
NaN	...	NaN	NaN	open	NaN	NaN	NaN	NaN	NaN	NaN	RNN
NaN	...	NaN	NaN	NaN	death_event	NaN	NaN	NaN	NaN	NaN	ANN
bove_ring	...	NaN	NaN	NaN	NaN	class	NaN	NaN	NaN	NaN	ANN
NaN	...	NaN	NaN	NaN	NaN	NaN	weighted_price	NaN	NaN	NaN	RNN
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	price	NaN	NaN	RNN
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	exited	NaN	ANN
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	price	NaN	NaN	ANN
NaN	...	NaN	NaN	NaN	NaN	class	NaN	NaN	NaN	NaN	RNN
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	lyrics	RNN
v7	...	NaN	NaN	NaN	NaN	class	NaN	NaN	NaN	NaN	ANN
oil_type27	...	NaN	cover_type	NaN	NaN	NaN	NaN	NaN	NaN	NaN	ANN
NaN	...	NaN	toxic	NaN	NaN	NaN	NaN	NaN	NaN	NaN	RNN
NaN	...	NaN	NaN	NaN	deal_probability	NaN	NaN	NaN	NaN	NaN	RNN

Figure 6.24Final general frame after generating all the columns.

6.3.2 Encoding general data frame

There are several ways of attempts to encode the dataset. Here it will give the focus on all of these attempts and the successful attempts as well.

6.3.2.1 NLP encoding method 01

As already discussed in the designing chapter, it implemented the text similarity comparison between the dependent and each independent variable to convert them to values. But not the same with neural network type values and dependent variables. Instead, it took between the dependent variable and empty variables in the following code.

```
createf=False
try:
    if inFDirectory!='':
        filelist=os.listdir(inFDirectory)
        filelist=natsorted(filelist)
    self.genFra=pd.read_csv(csvFile,index_col=0)
except:
    print("Nofile or data directory exist. file will automatically created.")
    createf=True
if createf:

    row=0
    indVCols=self.genFra.columns[:51].to_list()
    depVCols=self.genFra.columns[51:61].to_list()
```

Figure 6.25:code to check whether there is a frame already saved as a CSV file and if not create it.

As previously it first checks if there is a CSV file and if it will load that file otherwise it runs the encoding algorithm and generates the encoded general frame again.

First, take the dependent and independent column headings into two lists. Then on the dependent side, it takes the value. By the code “for depVCol in depVCols:” After that, it will compare each value in the row by the code “for indVCol in indVCols:” section. When the cell is equal to NaN the value is assigned to 0. It gives the following encoding output in Figure 6.26 for the independent side.

file	indV01	indV02	indV03	indV04	indV05	indV06	indV07	indV08	indV09	...	depV02
1.weatherAUS.csv	0.336735	0.283549	0.209251	0.270096	0.658881	0.401368	0.428180	0.318123	0.331493	...	NaN
2.data.csv	0.238386	0.358801	0.243014	0.162223	0.243634	0.270281	0.201914	0.216305	0.221139	...	diagnosis
with 4_folds_train.csv	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN
gle_stock_pricePr.csv	0.410736	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN
_records_dataset.csv	0.410105	0.335925	0.000000	0.000000	0.000000	0.254232	0.000000	0.000000	0.000000	...	NaN
6.mushrooms.csv	0.000000	0.462786	0.458673	0.346881	0.313001	0.401518	0.389349	0.355011	0.110162	...	NaN
-01-01_to_2017-10...	0.000000	0.294572	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN
odel_RNN_Ridge.csv	0.488698	0.459097	0.358441	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN
Churn_Modelling.csv	0.366818	0.264699	0.190689	0.000000	0.384182	0.280562	0.192468	0.000000	0.000000	...	NaN
0.kc_house_data.csv	0.424266	0.358543	0.413940	0.461378	0.000000	0.443914	0.301461	0.262474	0.000000	...	NaN
tion_Using_RNN.csv	0.479414	0.533667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN
_Generator_RNN.csv	0.000000	0.441644	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN
13.creditcard.csv	0.463596	0.436268	0.473630	0.535071	0.452151	0.495106	0.312725	0.426376	0.426037	...	NaN
rest_Cover_Type.csv	0.374114	0.329097	0.330494	0.318033	0.344691	0.348785	0.280115	0.000000	0.318655	...	NaN
ic-comment-train.csv	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	NaN

Figure 6.26:output after encoding for the independent side by NLP encoding method 01

After that, the dependent side encoding is done as mentioned above like in the following code. As with the previous one, this will also create a CSV file and if that file does not exist the frame is created again.

```
def depVFrEmtVEncode(self,csvFile='GeneralFrameWithdepVEncode.csv',inFDirectory=''):
    createf=False
    try:

        if inFDirectory!='':

            filelist=os.listdir(inFDirectory)
            filelist=natsorted(filelist)
        self.genFra=pd.read_csv(csvFile,index_col=0)
    except:
        print("Nofile or data directory exist. file will automatically created.")
        createf=True
    if createf:
        row=0
        depVCols=self.genFra.columns[51:61].to_list()
        typeValue=''
        while row<len(self.genFra.index):
            #typeValue=typeValues[row]
            for depVCol in depVCols:
```

Figure 6.27:This piece of code sees whether the general frame is already saved as a CSV file if not create the file.

```
 while row<len(self.genFra.index):
     #typeValue=typeValues[row]
     for depVCol in depVCols:
         depVCmpSel=self.genFra[depVCol][row]
         print('depVCmpSel:',depVCmpSel)
         if str(depVCmpSel)!=str(np.nan):
             typeValue=str(np.nan)#'ANN'
             cmpVal=self.compaireContents(typeValue,depVCmpSel)
             self.genFra[depVCol][row]=cmpVal
         else:
             self.genFra[depVCol][row]=0 # insted 0
     row+=1
 #endloop
 self.genFra.to_csv(csvFile)
urn self.genFra
```

Figure 6.28: The code compares the values with an empty string and inserts the value scores

It gives the following output.

depV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	depV10	nature_of_NN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	ANN
0.447067	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	ANN
0.000000	0.391161	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	RNN
0.000000	0.000000	0.376762	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	RNN
0.000000	0.000000	0.000000	0.318676	0.000000	0.000000	0.000000	0.000000	0.000000	ANN
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.000000	ANN
0.000000	0.000000	0.000000	0.000000	0.000000	0.313094	0.000000	0.000000	0.000000	RNN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.529946	0.000000	0.000000	RNN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.267156	0.000000	ANN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.529946	0.000000	0.000000	ANN
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.000000	RNN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.425531	RNN
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.000000	ANN
0.000000	0.250938	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	ANN
0.000000	0.419452	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	RNN
0.000000	0.000000	0.000000	0.189058	0.000000	0.000000	0.000000	0.000000	0.000000	RNN

Figure 6.29: Output after encoding the dependent side.

The manual fitted label encoding was done for the NN_nature column.

depV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	depV10	nature_of_NN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
0.447067	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
0.000000	0.391161	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1
0.000000	0.000000	0.376762	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1
0.000000	0.000000	0.000000	0.318676	0.000000	0.000000	0.000000	0.000000	0.000000	0
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.000000	0
0.000000	0.000000	0.000000	0.000000	0.000000	0.313094	0.000000	0.000000	0.000000	1
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.529946	0.000000	0.000000	1
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.267156	0.000000	0
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.529946	0.000000	0.000000	0
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.000000	1
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.425531	1

Figure 6.30:After encoding the Nature of the neural network column with label encoding

Correlated matrix after encoding.

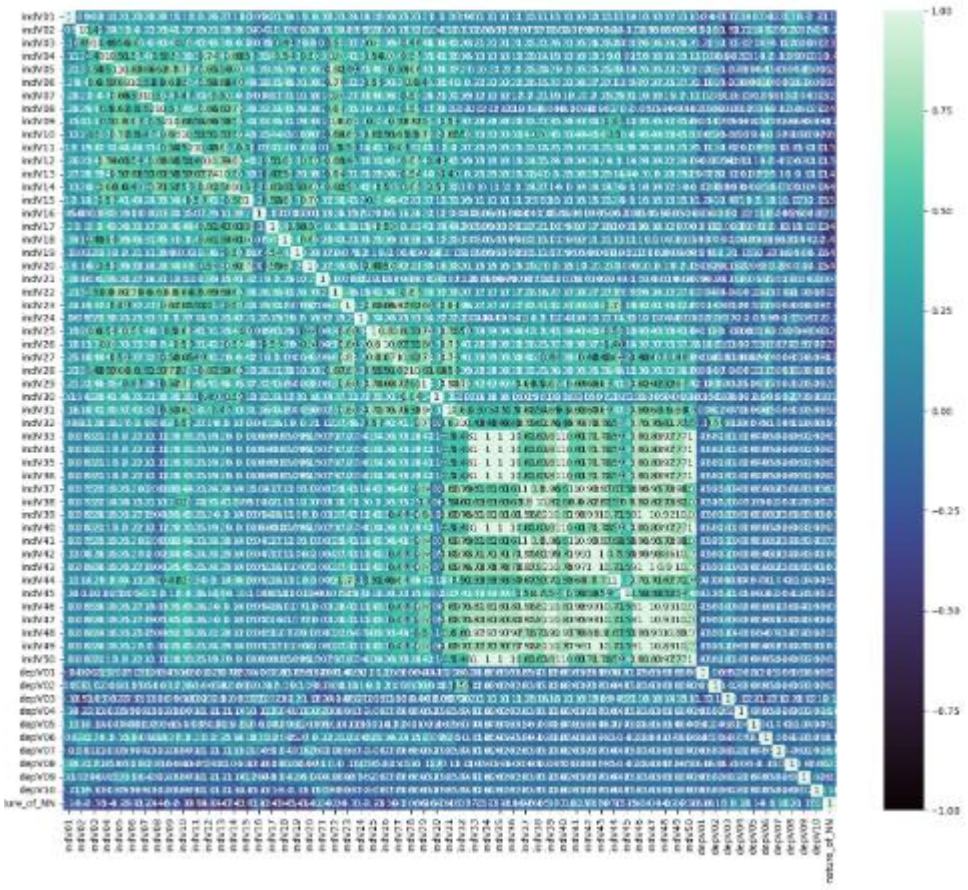


Figure 6.31Correlated matrix for the NLP encoding method 01

Outputs of the neural network

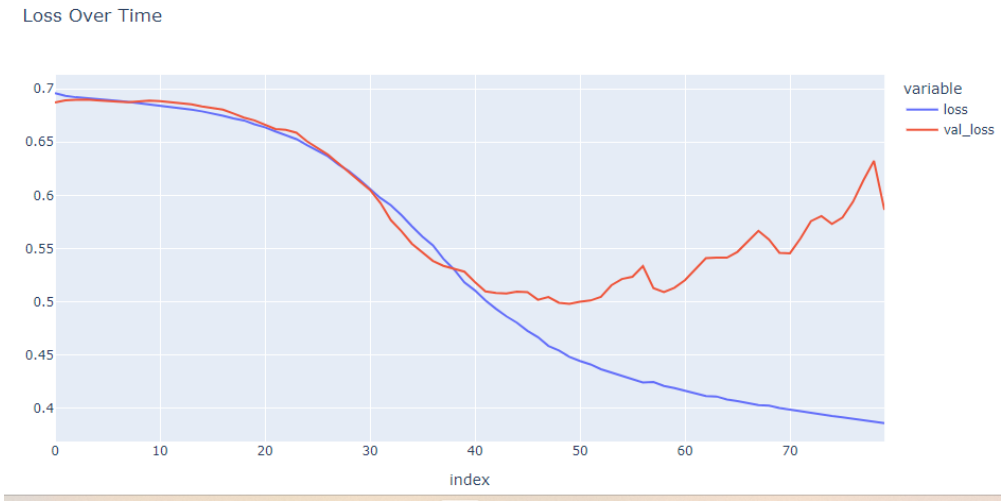


Figure 6.32:Output of the neural network trained by the encoded values(the lost)

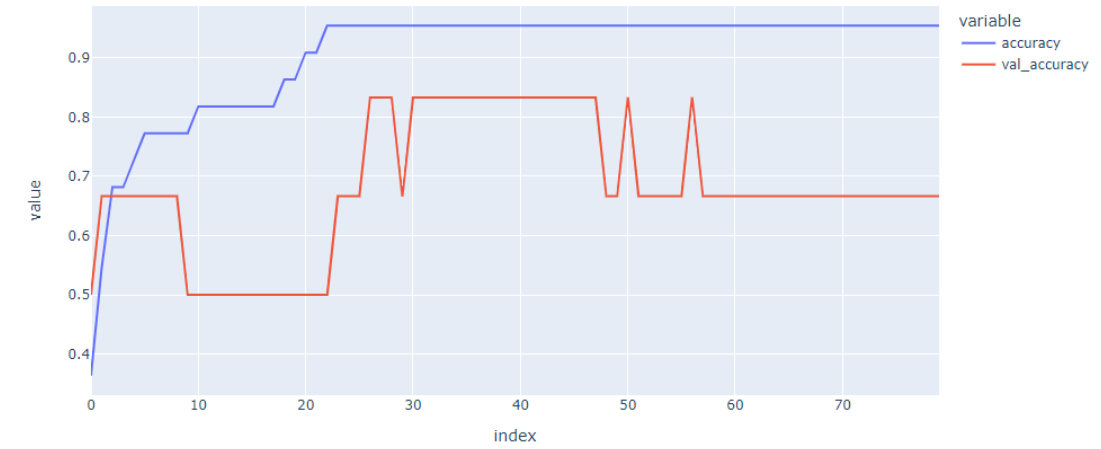


Figure 6.33Output of the neural network trained by the encoded values(the accuracy)

But the model is still not stable because of low data. This is achieved by the first run. But in the second run of the model, it has given low performance as follows.

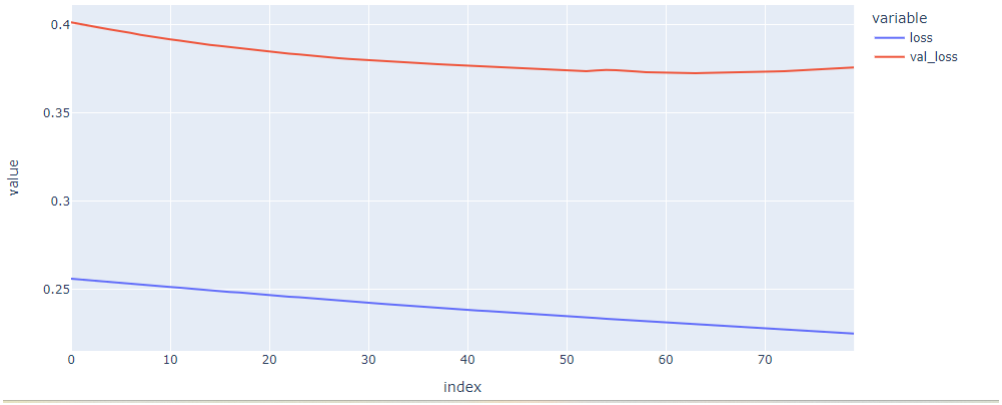


Figure 6.34output for the second time training of the neural network.

6.3.2.2 NLP encoding method 02

In this method, similarities are taken for all the values concerning the empty value string by NLP as per the following code

```
cols=self.genFra.columns[1:61].to_list()
typeValue=''
while row<len(self.genFra.index):
    #typeValue=typeValues[row]
    for col in cols:
        depVCmpSel=self.genFra[col][row]
        print('depVCmpSel:',depVCmpSel)
        if str(depVCmpSel)!=str(np.nan):
            typeValue=str(np.nan)
            cmpVal=self.compaireContents(typeValue,depVCmpSel)
            self.genFra[col][row]=cmpVal
        else:
            self.genFra[col][row]=0 # insted 0
    row+=1
#endloop
self.genFra.to_csv(csvFile)
```

Figure 6.35: This is the code for NLP encoding method 02.

The encoded output is as follows.

depV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	depV10	nature_of_NN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	ANN
0.447067	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	ANN
0.000000	0.391161	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	RNN
0.000000	0.000000	0.376762	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	RNN
0.000000	0.000000	0.000000	0.318676	0.000000	0.000000	0.000000	0.000000	0.0	ANN
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.0	ANN
0.000000	0.000000	0.000000	0.000000	0.000000	0.313094	0.000000	0.000000	0.0	RNN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.529946	0.000000	0.0	RNN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.267156	0.0	ANN
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.529946	0.000000	0.0	ANN
0.000000	0.000000	0.000000	0.000000	0.564076	0.000000	0.000000	0.000000	0.0	RNN

Figure 6.36:Encoding output for NLP method 02.

Correlated matrix

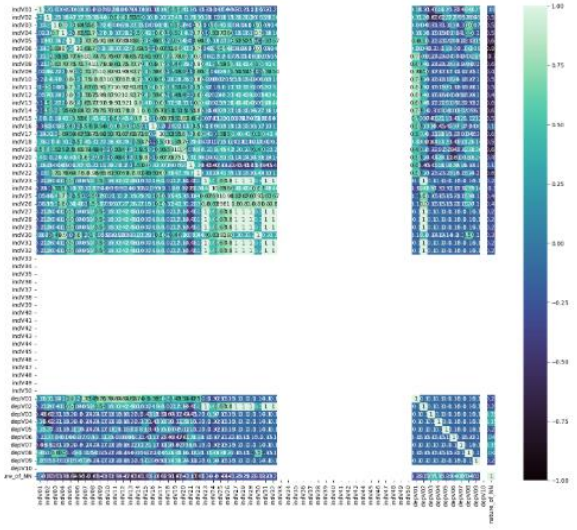


Figure 6.37: This is the correlation matrix for the NLP encoding method 02.

a problem is in the correlated matrix

Outputs of the neural network

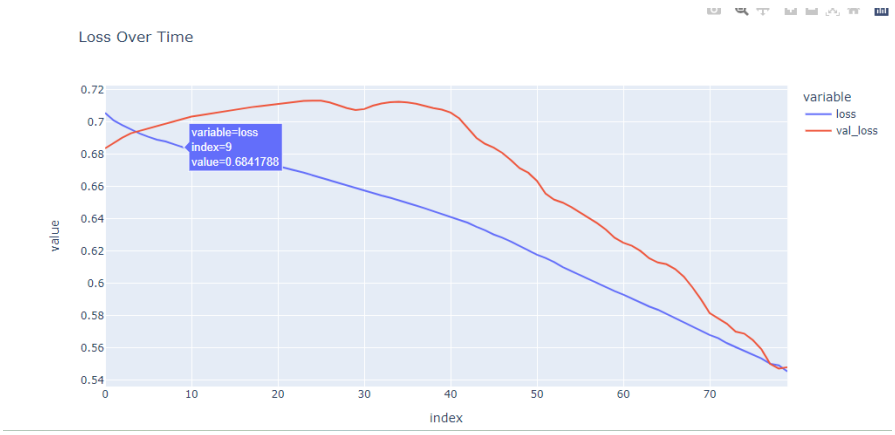


Figure 6.38: The output for the training by NLP method 02.

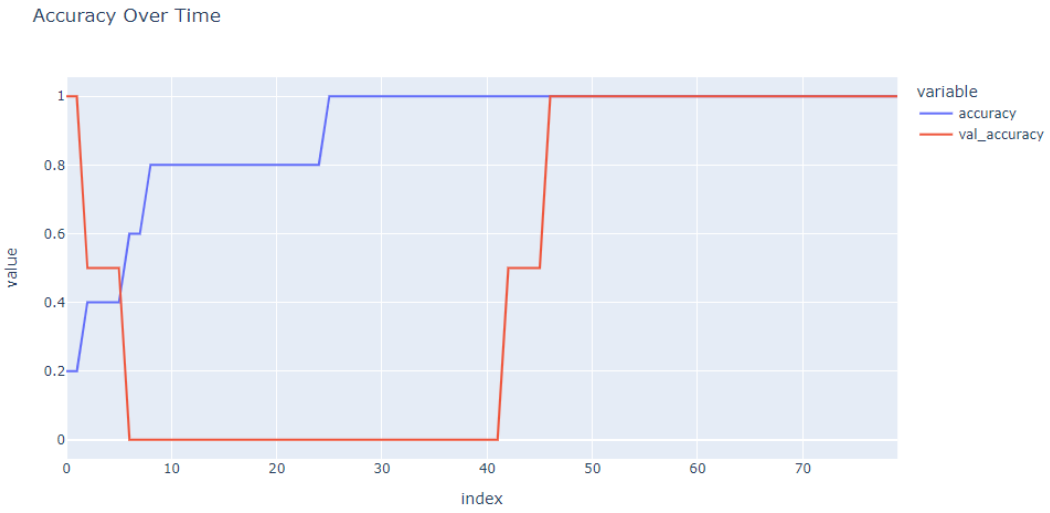


Figure 6.39:Accuracy of the neural network after training by giving the encoded values by NLP encoding method 02.

Here also give the correct answers but the performance values are strange.

6.3.2.3 Label encoding with random fit

The label encoding with random fit values has given stable results from the beginning.

The following coding implements label encoding.

```
def fullLableEncode(self,csvFile='GeneralFrameLBLEncode.csv',inFDirectory=''):
    createf=False
    try:

        if inFDirectory!='':

            filelist=os.listdir(inFDirectory)
            filelist=natsorted(filelist)
        self.genFra=pd.read_csv(csvFile,index_col=0)
    except:
        print("Nofile or data directory exist. file will automatically created.")
        createf=True
    if createf:
        lbl=LabelEncoder()
        cols=self.genFra.columns[1:61].to_list()
        typeValue=''
        for col in cols:
            self.genFra[col]=lbl.fit_transform(self.genFra[col])
        #endLoop
        self.genFra.to_csv(csvFile)
    return self.genFra
```

Figure 6.40: This is the code for the label encoding for the entire table.

More specifically the code: self.genFra[col]=lbl.fit_transform(self.genFra[col])

It gives the output below.

epV02	depV03	depV04	depV05	depV06	depV07	depV08	depV09	depV10	nature_of_NN
1	9	1	3	2	1	3	1	1	ANN
0	9	1	3	2	1	3	1	1	ANN
1	7	1	3	2	1	3	1	1	RNN
1	9	0	3	2	1	3	1	1	RNN
1	9	1	1	2	1	3	1	1	ANN
1	9	1	3	0	1	3	1	1	ANN
1	9	1	3	2	0	3	1	1	RNN
1	9	1	3	2	1	1	1	1	RNN
1	9	1	3	2	1	3	0	1	ANN
1	9	1	3	2	1	1	1	1	ANN
1	9	1	3	0	1	3	1	1	RNN
1	9	1	3	2	1	3	1	0	RNN

Figure 6.41:Output after applying label encoding with the random fit.

Correlated matrix

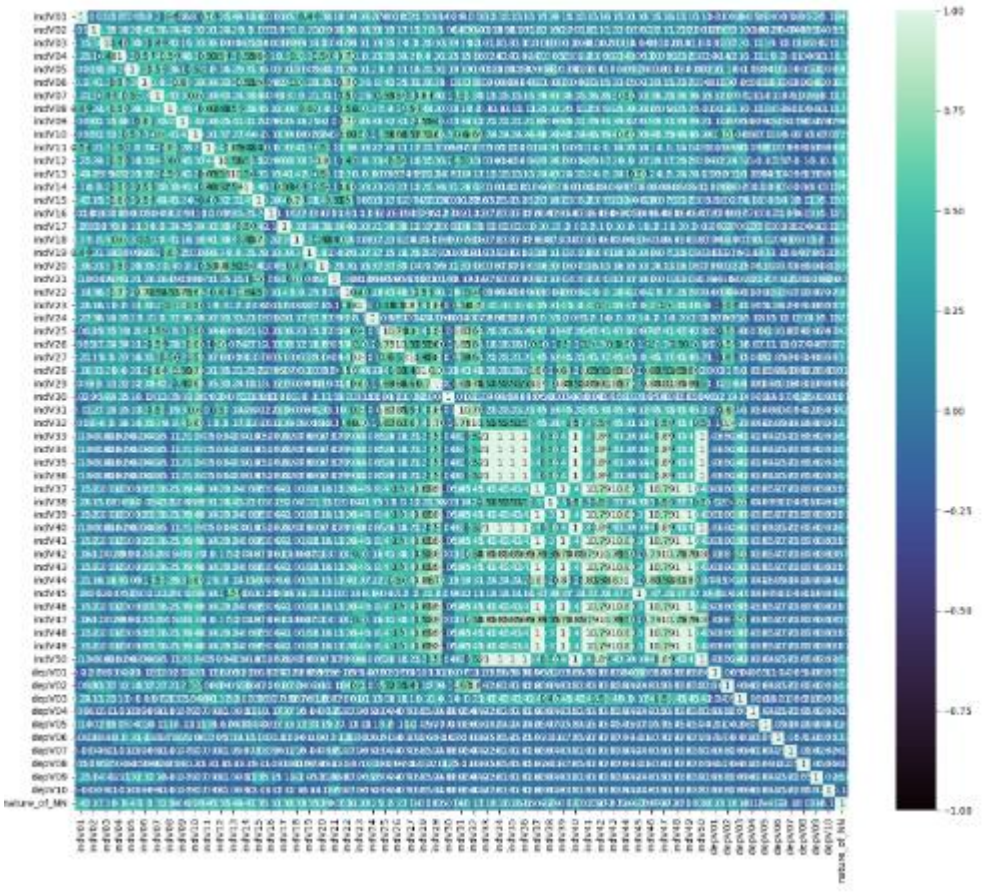


Figure 6.42:Thisis the correlation matrix after label encoding.

Outputs of the neural network are.

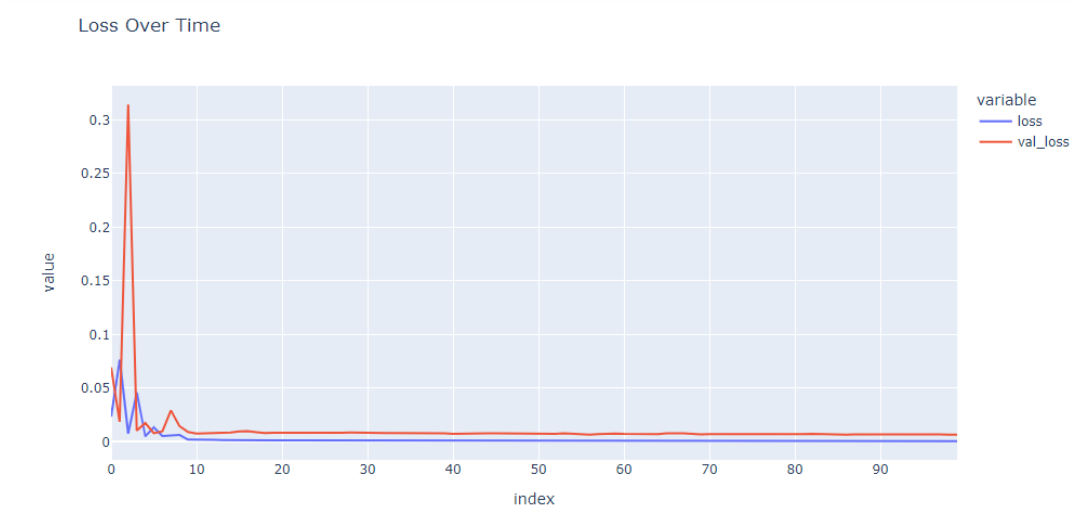


Figure 6.43: this is the output of the neural network after applying encoded values for the training

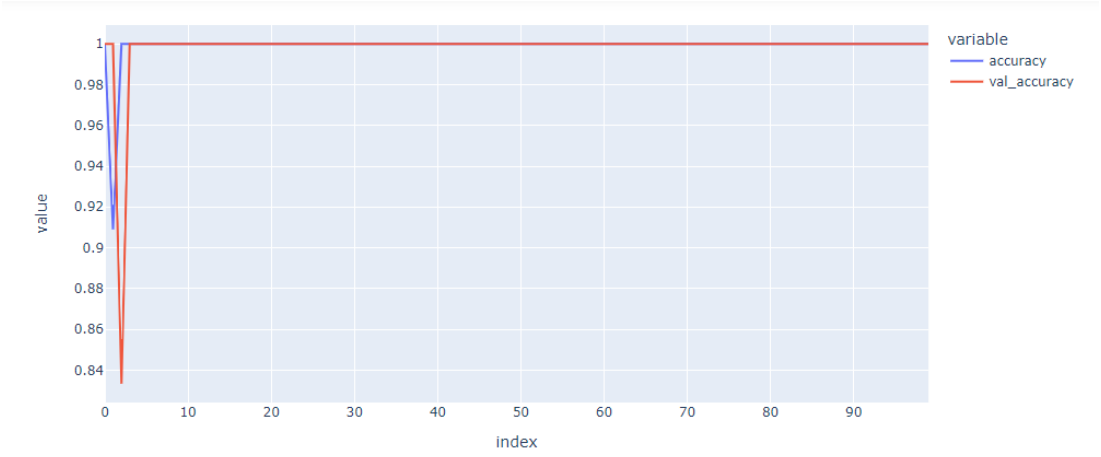


Figure 6.44:This is the accuracy after the training by label encoding.

This gives more stable results for 40 feature sets in the general frame. However, even though the NLP encoding 01 method is unstable it has given a accuracy of 83%

```
In [37]:  1 model.evaluate(X_test, Y_test)
          1/1 [==============================] - 0s 217ms/step - loss: 0.7410 - accuracy: 0.8333 - auc: 0.7031
Out[37]: [0.7410456538200378, 0.8333333134651184, 0.703125]
```

Figure 6.45:Prediction accuracy for the NLP encoding method01.

The label encoder only gave an accuracy of 41%

```
  1 model.evaluate(X_test, Y_test)
1/1 [==============================] - 0s 25ms/step - loss: 3.5680 - accuracy: 0.4167 - auc: 0.4688
[3.567983865737915, 0.4166666567325592, 0.46875]
```

Figure 6.46: Predicting accuracy for the label encoding.

So, NLP Encoding method 01 has been selected to use.

6.3.3 The neural network

Encoded data is fed to the neural network. Through NumPy array has fed to the network. As follows. Further to verify the value distribution between 0 and 1 the min max scaler has been used.

```
scaler = MinMaxScaler()
```

```
Y = EncodedGendfExComcol.loc[:, 'nature_of_NN']
Y=Y.to_numpy()
#Y=np.asarray(Y)
X = EncodedGendfExComcol.drop('nature_of_NN', axis=1)
scaler.fit(X)
X=scaler.transform(X)
#X=X.to_numpy()
#X=np.asarray(X)
X
```

Figure 6.47: value transfer through NumPy and min-max Scaler.

SKlern train_test_split has been used to split the training and testing split as follows.

```
[20]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, random_state=46)
```

Figure 6.48:Traning testing split by sklearn.

The neural network has 61 inputs and 1 output because of the binary classifier. So, the activation function “relu” is used for the layers except for the output layer. For the output layer “sigmoid” has been used the number of neurons taken around half the previous layer. Adams optimizer has been used to optimize the results.

```
inputs = tf.keras.Input(shape=(60,))
x = tf.keras.layers.Dense(35, activation='relu')(inputs)
x = tf.keras.layers.Dense(18, activation='relu')(x)
x = tf.keras.layers.Dense(8, activation='relu')(x)
x = tf.keras.layers.Dense(3, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs, outputs)

model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.AUC(name='auc')
    ]
)
```

Figure 6.49:Neural network configuration for the binary classifier.

After the training as in the figures 6.32,6.33,6.38,39,6.43 and 6.44 taken for different encoding. As the NLP encoding method 01 has been selected here will use that model. It also has given the accuracy 0f 83% as in Figure 6.45.

6.4 Module 05: Hyperparameter prediction module

As explained in the hyperparameter prediction section in the previous chapter of this thesis 12 main hyperparameter predictors have the capability of predicting more than 32 hyperparameters that assume the final generated configuration has around 6 layers. First, that can look into the implementation of single-type predictors and classifiers that will classify or predict single-type predictors then move to the group-type predictor or classifiers that will predict or classify hyperparameters layer-wise.

6.4.1 Layer Number Predictor

The following neural network structure created for the layer number predictor with 4 layers

```
modelRe = Sequential()
modelRe.add(Dense(30, input_shape=(61,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelRe.add(Dense(15, activation='relu'))
modelRe.add(Dense(8, activation='relu'))
modelRe.add(Dense(1))#, activation='linear'
modelRe.compile(loss='mse', optimizer='adam')
```

```
batch_size = 100
epochs = 1300
```

Figure 6.50: neuralnerwork configuration number of layers predictor

The above neural network structure results in the following graph after the training

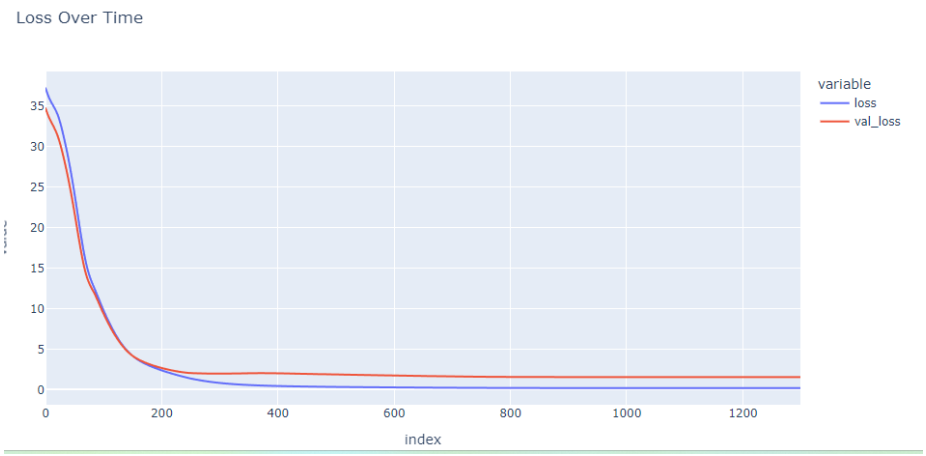


Figure 6.51loss function number of layers predictor

Following ac accuracy is achieved for the layer number. however, it can observer that the actual value and the predicted value are the same.

```
modelRe.evaluate(X_test,Y_test)
```

```
2/2 [==============================] - 0s 5ms/step - loss: 0.3563
0.35634711384773254
```

```
y_result=modelRe.predict(X_test)
```

```
2/2 [==============================] - 0s 2ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_result[i]).astype(int)))
```

```
Actual=10, Predicted=[10]
Actual=4, Predicted=[4]
Actual=5, Predicted=[5]
Actual=5, Predicted=[5]
Actual=12, Predicted=[12]
Actual=3, Predicted=[3]
Actual=3, Predicted=[3]
Actual=5, Predicted=[5]
Actual=3, Predicted=[3]
Actual=8, Predicted=[8]
Actual=6, Predicted=[7]
Actual=5, Predicted=[5]
```

Figure 6.52Actual value and predicted value of number of layers

6.4.2 Optimizer classifier

For optimizer classifier following 3 layer neural network structure has created

```
modelOptmizp = Sequential()
modelOptmizp.add(Dense(31, activation='relu', input_dim=62))
modelOptmizp.add(Dense(16, activation='relu'))
modelOptmizp.add(Dense(7, activation='softmax'))
modelOptmizp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
#outputs.shape
```

```
batch_size = 10
epochs = 80
```

Figure 6.53Neural network for optimizer classifier

The structure results in the following loss and accuracy graphs that are converging

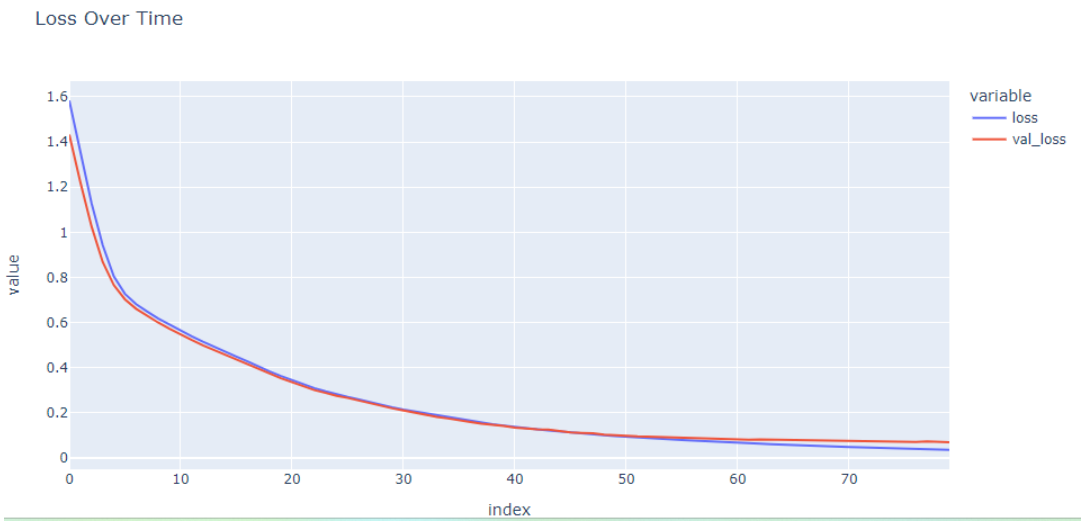


Figure 6.54Lost function for optimizer classifier

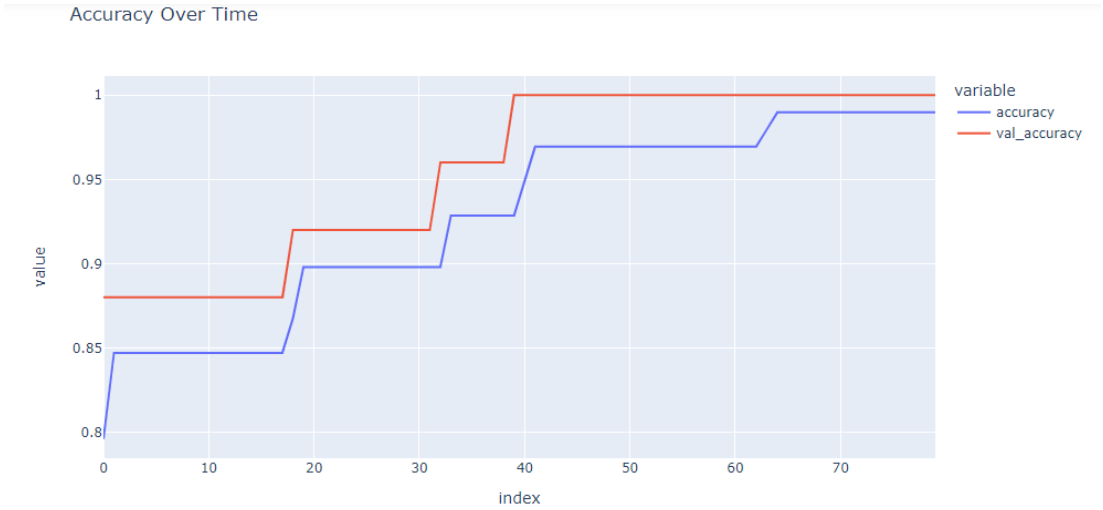


Figure 6.55 accuracy of optimizer classifier

Optimizer accuracy is given in 100% as bellow

```
: 1 modelOptmizp.evaluate(X_test, to_categorical(Y_test,7))

  2/2 [==============================] - 0s 5ms/step - loss: 0.0273 - accuracy: 1.0000

: [0.027255585417151145, 1.0]

: 1 y_resultOptimP=modelOptmizp.predict(X_test)

  2/2 [==============================] - 0s 3ms/step

: 1  OptimPL=[]
   2  leGenrl.fit(unqValues[0])
   3  for i in range(len(X_test)):
   4      Optim=np.argmax(y_resultOptimP[i])
   5      OptimPL.append(Optim)
   6      #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
   7  y_resultOptimP=leGenrl.inverse_transform(OptimPL)
   8  Y_testOptimP=leGenrl.inverse_transform(Y_test)
   9  for i in range(len(X_test)):
  10
  11      print("Actual=%s, Predicted=%s" % (Y_testOptimP[i], y_resultOptimP[i]))
```

Figure 6.56 Evaluation and prediction of optimizer classifier

Observe the actual values and observed values they are alike as follows.

```
 10
 11     print("Actual=%s, Predicted=%s" % (Y_testOptimP[i], y_resultOptimP[i]))

Actual=nadam, Predicted=nadam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=RMSprop, Predicted=RMSprop
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
Actual=adam, Predicted=adam
```

Figure 6.57 Predicted values of optimizer classifier

6.4.3 Lost function classifier

For develop the lost function classifier following neural network structure has been used.

```
]:
1 modelLostFnp = Sequential()
2 modelLostFnp.add(Dense(64, activation='relu', input_dim=63))
3 modelLostFnp.add(Dense(24, activation='relu'))
4 modelLostFnp.add(Dense(18, activation='relu'))
5 modelLostFnp.add(Dense(8, activation='softmax'))
6 modelLostFnp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
]:
1 batch_size = 16
2 epochs = 40
```

Figure 6.58 Neural network for lost function classifier

When training the neural network it results following the loss and the accuracy graphs.

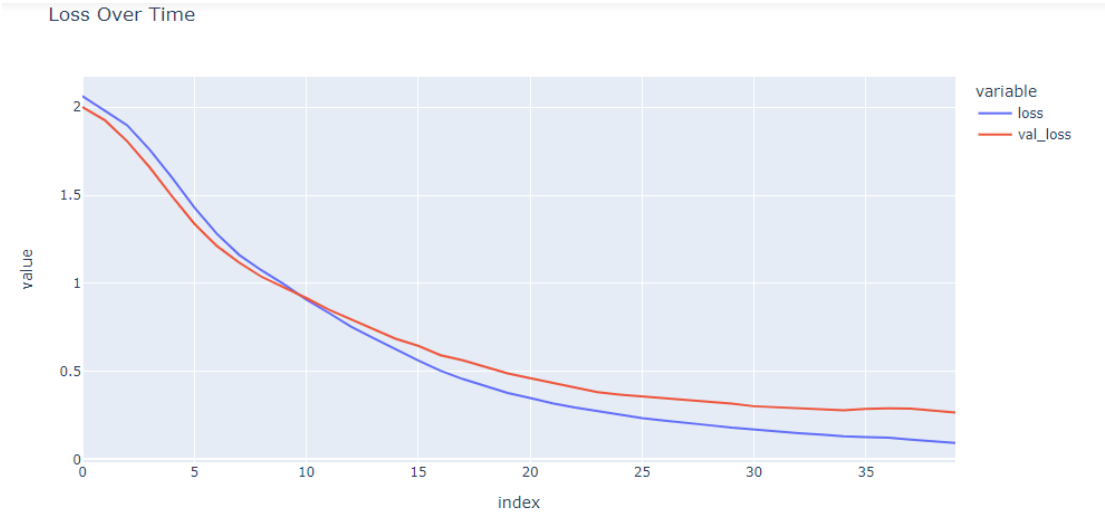


Figure 6.59 Lost function for lost function classifier

Figure 6.60Accuracy of lost function classifier

Accuracy of the classifre is 96% as bellow.

```
modelLostFnp.evaluate(X_test, to_categorical(Y_test,8))
```

```
2/2 [==============================] - 0s 2ms/step - loss: 0.2108 - accuracy: 0.9623
[0.2107648253440857, 0.9622641801834106]
```

```
y_resultLostFnp=modelLostFnp.predict(X_test)
```

```
2/2 [==============================] - 0s 3ms/step
```

Figure 6.61Evaluation and prediction of lostfuntion classifier

When checking values the predicted value and the actual values is almost the same in all the times.

```
LostFnpL=[]
leGenrl.fit(unqValues[1])
for i in range(len(X_test)):
    LostFnp=np.argmax(y_resultLostFnp[i])
    LostFnpL.append(LostFnp)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLostFnp=leGenrl.inverse_transform(LostFnpL)
Y_testLostFnp=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLostFnp[i], y_resultLostFnp[i]))
```

```
Actual=Manual, Predicted=Manual
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=mse, Predicted=mse
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=categorical_crossentropy, Predicted=mean_squared_error
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=binary_crossentropy, Predicted=binary_crossentropy
Actual=mean_squared_error, Predicted=mean_squared_error
Actual=binary_crossentropy, Predicted=binary_crossentropy
```

Figure 6.62Predicted values of lost function classifier

6.4.4 Lost metrix classifier

Fowllwing nural network structure is configured to classify the metrixes upon previous classifiers and predictoers outputs.

```
tf.keras.backend.clear_session()
```

```
modelMetrixp = Sequential()
modelMetrixp.add(Dense(32, activation='relu', input_dim=64))
modelMetrixp.add(Dense(25, activation='relu'))
modelMetrixp.add(Dense(18, activation='elu'))
modelMetrixp.add(Dense(8, activation='softmax'))
modelMetrixp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 10
epochs = 80
```

Figure 6.63Neural network for metrix classifier

Traing of the neural networks has ended with following graphs of loss and the accuracy.

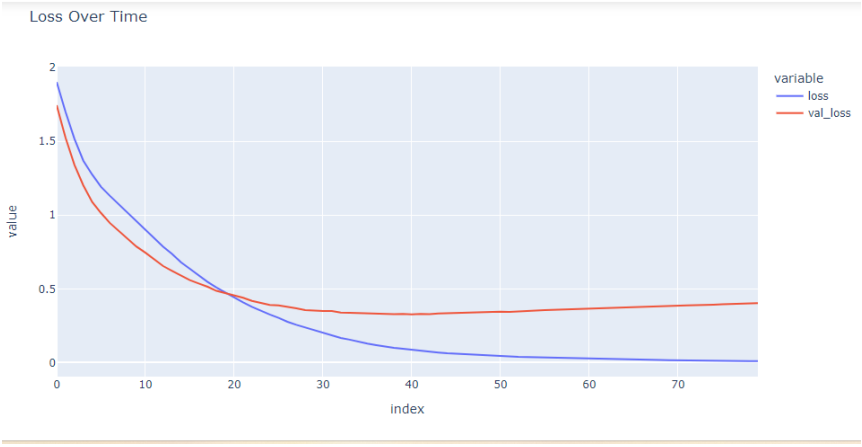


Figure 6.64Lost function for metrix classifier

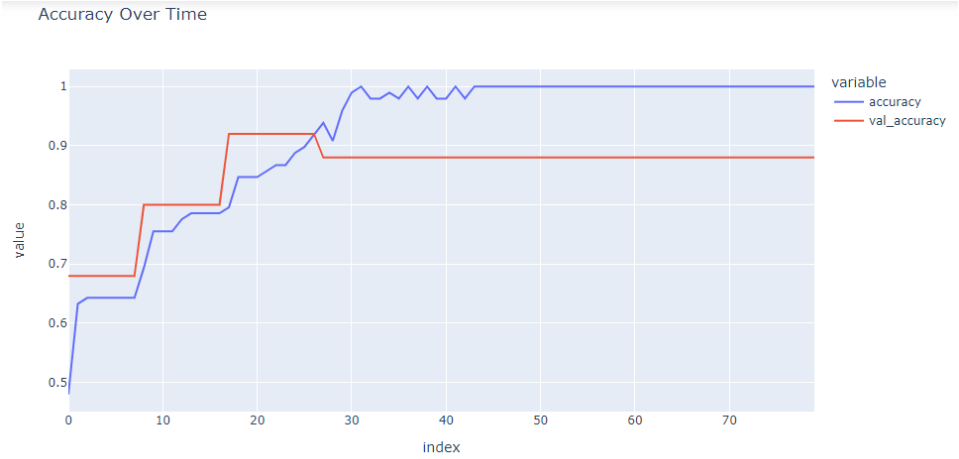


Figure 6.65Accuracy of metrix classifier

Metrix classifier also endup with 96% accuracy.

```
: 1 modelMetrixp.evaluate(X_test, to_categorical(Y_test,8))

  2/2 [==============================] - 0s 3ms/step - loss: 0.1222 - accuracy: 0.9623

: [0.12220624834299088, 0.9622641801834106]

: 1 y_resultMetrixp=modelMetrixp.predict(X_test)

  2/2 [==============================] - 0s 2ms/step
```

Figure 6.66Evaluation and prediction of metrix classifier

When checking the values it give almost the same values as bellow.

```
MetrixpL=[]
leGenrl.fit(unqValues[2])
for i in range(len(X_test)):
    Metrixp=np.argmax(y_resultMetrixp[i])
    MetrixpL.append(Metrixp)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultMetrixp=leGenrl.inverse_transform(MetrixpL)
Y_testMetrixp=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testMetrixp[i], y_resultMetrixp[i]))
```

```
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=accuracy
Actual=mean_squared_error, Predicted=mean_squared_error
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=mae
Actual=accuracy, Predicted=accuracy
Actual=cust, Predicted=cust
Actual=accuracy, Predicted=accuracy
Actual=accuracy, Predicted=accuracy
```

Figure 6.67 Predicted values of metrix classifier

6.4.5 Number of epochs predictor

The next neural network is built to predict the number of epochs.

```
modelNoEpochP = Sequential()
modelNoEpochP.add(Dense(33, input_shape=(65,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelNoEpochP.add(Dense(17, activation='relu'))
modelNoEpochP.add(Dense(9, activation='relu'))
modelNoEpochP.add(Dense(1))#, activation='linear'
modelNoEpochP.compile(loss='mse', optimizer='adam')
```

```
batch_size = 120
epochs = 10000
```

Figure 6.68 Neural network for epochs predictor

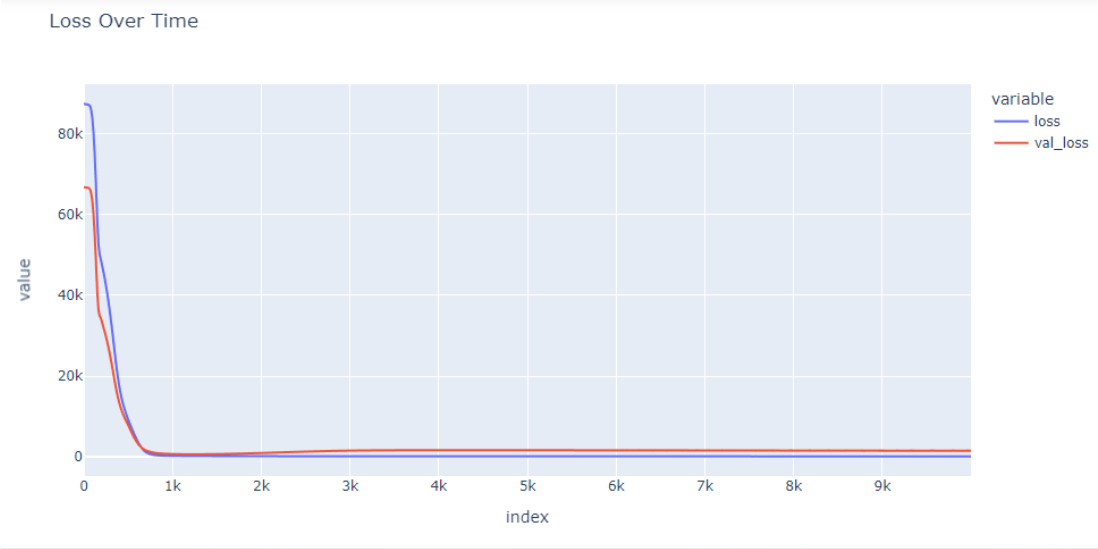


Figure 6.69Lost function for epochs predictor

Even the training has result a 148 loss when checking values they are almost the same.

```
1 modelNoEpochP.evaluate(X_test,Y_test)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 148.0472

148.04721069335938
```

```
1 y_resultNoEpochP=modelNoEpochP.predict(X_test)
```

```
2/2 [==============================] - 0s 3ms/step
```

Figure 6.70Evaluation and prediction of epochs predictor

```
1 for i in range(len(X_test)):
2
3     print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_resultNoEpochP[i]).astype(int)))
```

```
Actual=1000, Predicted=[1000]
Actual=60, Predicted=[60]
Actual=150, Predicted=[150]
Actual=10, Predicted=[6]
Actual=2, Predicted=[2]
Actual=5, Predicted=[5]
Actual=5, Predicted=[5]
Actual=10, Predicted=[10]
Actual=25, Predicted=[74]
Actual=500, Predicted=[500]
Actual=100, Predicted=[100]
Actual=10, Predicted=[6]
Actual=500, Predicted=[500]
Actual=100, Predicted=[100]
Actual=10, Predicted=[6]
Actual=400, Predicted=[400]
Actual=500, Predicted=[500]
Actual=10, Predicted=[6]
```

Figure 6.71Predicted values of epochs predictor

6.4.6 Batch size predictor

The purpose of creating following configuration is to achive batch size predictor.

```
modelBtchSizeP = Sequential()
modelBtchSizeP.add(Dense(33, input_shape=(66,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelBtchSizeP.add(Dense(17, activation='relu'))
modelBtchSizeP.add(Dense(9, activation='elu'))
modelBtchSizeP.add(Dense(1))#, activation='linear'
modelBtchSizeP.compile(loss='mean_squared_logarithmic_error', optimizer='adam',metrics=['mse'])
```

```
batch_size = 200
epochs = 10000
```

Figure 6.72Neural network for batch size predictor

The lost functions converges as follows after training.

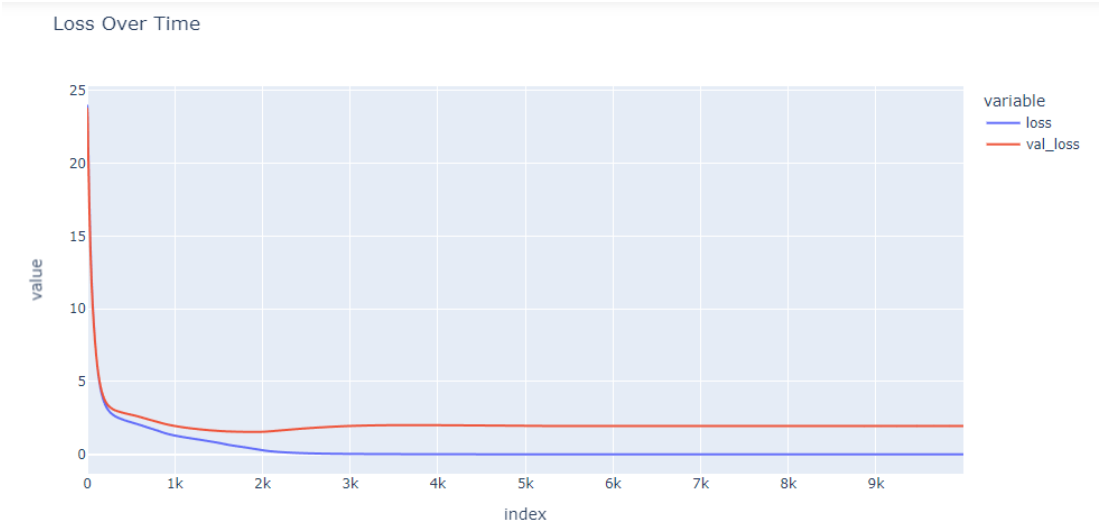


Figure 6.73Lost function for batch size predictor

The training of the neuralnetwork results in the lost of 0.47 as follows.

```
]:  1  modelBtchSizeP.evaluate(X_test,Y_test)

    2/2 [==============================] - 0s 2ms/step - loss: 0.4705 - mse: 51171.5430

]:  [0.47054606676101685, 51171.54296875]

]:  1  y_resultBtchSizeP=modelBtchSizeP.predict(X_test)

    2/2 [==============================] - 0s 2ms/step
```

Figure 6.74Evaluation and prediction of batch size predictor

When checking the values actual value and the predicted values are almost same.

```
1  for i in range(len(X_test)):
2
3      print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_resultBtchSizeP[i]).astype(int)))
```

```
Actual=1000, Predicted=[1000]
Actual=256, Predicted=[256]
Actual=32, Predicted=[32]
Actual=32, Predicted=[32]
Actual=1024, Predicted=[1024]
Actual=64, Predicted=[64]
Actual=1024, Predicted=[1024]
Actual=16, Predicted=[16]
Actual=32, Predicted=[1052]
Actual=32, Predicted=[32]
Actual=32, Predicted=[32]
Actual=30, Predicted=[30]
Actual=32, Predicted=[32]
Actual=10, Predicted=[10]
Actual=128, Predicted=[128]
Actual=128, Predicted=[128]
Actual=32, Predicted=[32]
Actual=1000, Predicted=[585]
```

Figure 6.75Predicted values of batch size predictor

6.4.7 Layer type classifier

As per the design it had bult neuralnetwork for each layer to classifiy layer type by using the manual configured layer type heyperparameters for each layer.

6.4.7.1 Layer type classifier for layer 01

Following neural network make achives the layer type classifier for layer one.

```
modelLType1p = Sequential()
modelLType1p.add(Dense(68, activation='relu', input_dim=68))
modelLType1p.add(Dense(34, activation='relu'))
modelLType1p.add(Dense(17, activation='relu'))
modelLType1p.add(Dense(7, activation='softmax'))
modelLType1p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 80
epochs = 100
```

Figure 6.76Neural network for layer type classifier for layer one

Trining of layer one classifier results in following loss and the accuracy graphs.

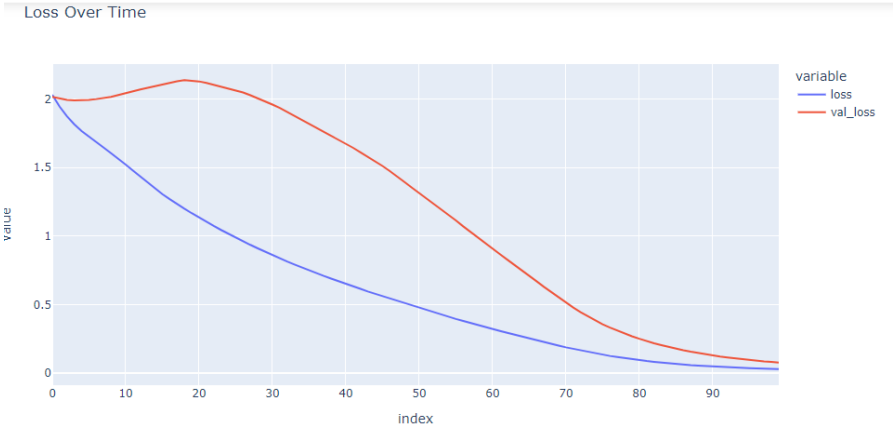


Figure 6.77Lost function for layer type classifier for layer one

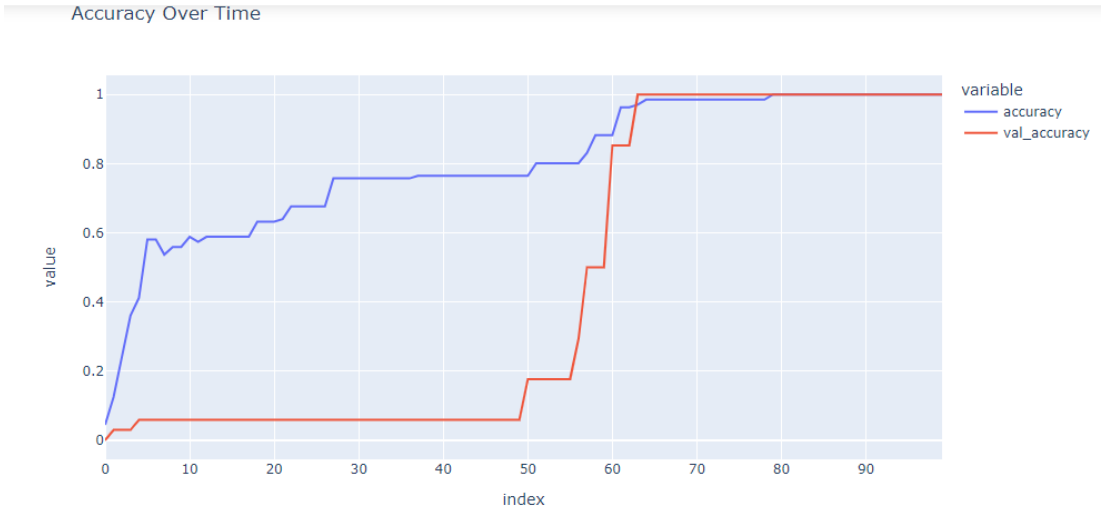


Figure 6.78Accuracy of layer type classifier for layer one

The training results in 76% accuracy for the layer one layer type classifier as follows.

```
In [2226]: modelLType1p.evaluate(X_test, to_categorical(Y_test,7))

           2/2 [==============================] - 0s 3ms/step - loss: 2.6752 - accuracy: 0.7667

Out[2226]: [2.6752233505249023, 0.7666666507720947]
```

Figure 6.79Evaluation of layer type classifier for layer one

When chaking the values it most of the times predict the actual value but sometimes it is not. When the times it predict wrong can be coved by the model restrictions to cover the values.

```
LayTypePL=[]
#le.fit(unqValues[2])
for i in range(len(X_test)):
    LayType=np.argmax(y_resultLType[i])
    LayTypePL.append(LayType)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leLType.inverse_transform(LayTypePL)
Y_testLI=leLType.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=Dense, Predicted=Dense
Actual=Input, Predicted=Other
Actual=Embedding, Predicted=Dense
Actual=Input, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=Embedding, Predicted=Embedding
Actual=Input, Predicted=Input
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=Embedding, Predicted=Dense
Actual=Embedding, Predicted=Embedding
Actual=Embedding, Predicted=Embedding
Actual=Dense, Predicted=Dense
```

Figure 6.80Predicted values of layer type classifier for layer one

6.4.7.2 Layer type classifier for layer 02

The following configuration makes layer tipe predictor for layer two.

```
modelLType2p = Sequential()
modelLType2p.add(Dense(68, activation='relu', input_dim=68))
modelLType2p.add(Dense(34, activation='relu'))
modelLType2p.add(Dense(17, activation='relu'))
modelLType2p.add(Dense(7, activation='softmax'))
modelLType2p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 80
epochs = 100
```

Figure 6.81Neural network for layer type classifier for layer two

The training of the neural network create the following loss graph and accuracy graph.

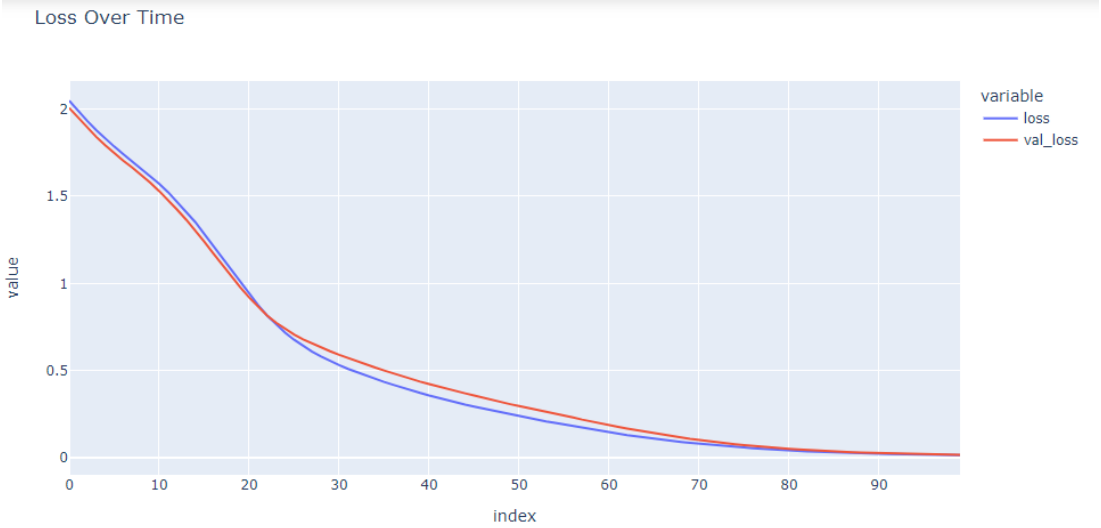


Figure 6.82Lost function for layer type classifier for layer two

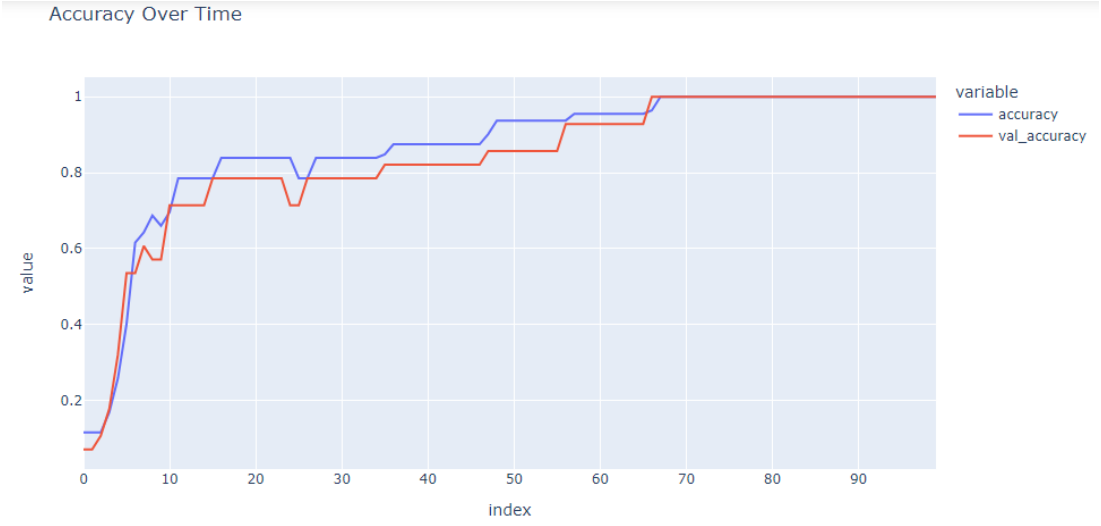


Figure 6.83Accuracy of layer type classifier for layer two

71% accuracy has reashed by the trinned model.

```
modelLType2p.evaluate(X_test, to_categorical(Y_test,7))
```

```
2/2 [==============================] - 0s 3ms/step - loss: 3.2157 - accuracy: 0.7167
[3.215651750564575, 0.7166666388511658]
```

```
y_resultLType=modelLType2p.predict(X_test)
```

```
2/2 [==============================] - 0s 3ms/step
```

Figure 6.84Evaluation and prediction of layer type classifier for layer two

When checking the values it sometimes produce duce wrong values which will be handled by the model restrictor rules set.

```
LayTypePL=[]
#le.fit(unqValues[2])
for i in range(len(X_test)):
    LayType=np.argmax(y_resultLType[i])
    LayTypePL.append(LayType)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leLType.inverse_transform(LayTypePL)
Y_testLI=leLType.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=LSTM, Predicted=LSTM
Actual=Embedding, Predicted=LSTM
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=LSTM, Predicted=Dense
Actual=LSTM, Predicted=LSTM
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=LSTM, Predicted=LSTM
```

Figure 6.85Predicted values of layer type classifier for layer two

6.4.7.3 Layer type classifier for layer 03

Layer three neural network model has designed on the following configuration.

```
modelLType3p = Sequential()
modelLType3p.add(Dense(68, activation='relu', input_dim=68))
modelLType3p.add(Dense(34, activation='relu'))
modelLType3p.add(Dense(17, activation='relu'))
modelLType3p.add(Dense(7, activation='softmax'))
modelLType3p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 80
epochs = 100
```

Figure 6.86Neural network for layer type classifier for layer three

The loss graph as results in the following manner with the accuracy graph.

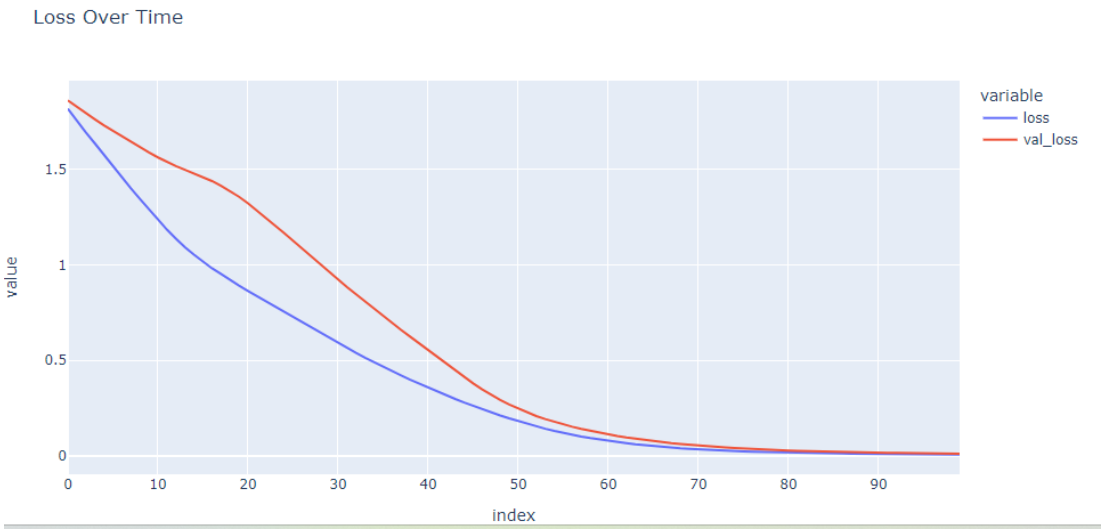


Figure 6.87Lost function for layer type classifier for layer three

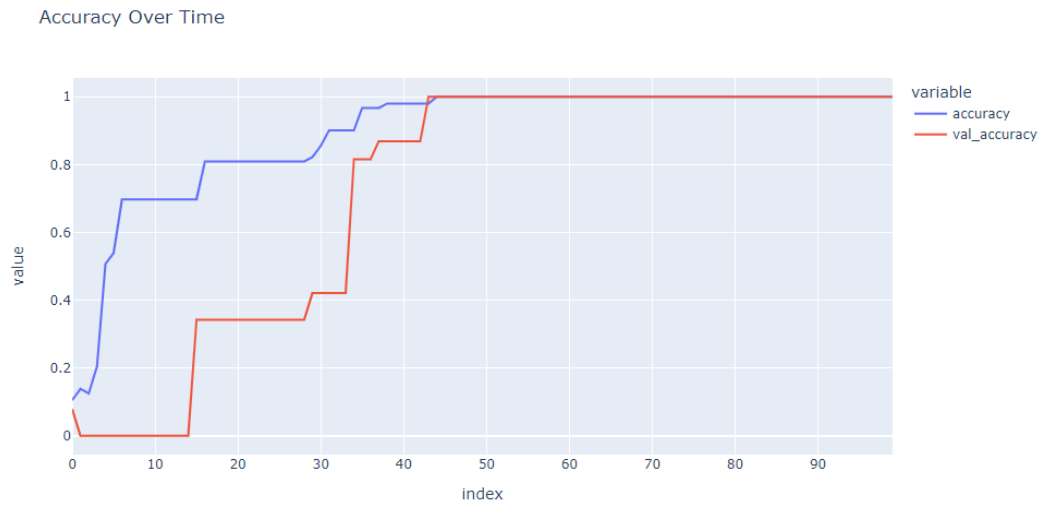


Figure 6.88Accuracy of layer type classifier for layer three

The accuracy of the predictor results in 57%

```
modelLType3p.evaluate(X_test, to_categorical(Y_test,7))
```

```
1/1 [==============================] - 0s 25ms/step - loss: 2.7488 - accuracy: 0.5714
[2.7487776279449463, 0.5714285969734192]
```

```
y_resultLType=modelLType3p.predict(X_test)
```

```
1/1 [==============================] - 0s 70ms/step
```

Figure 6.89Evaluation and prediction of layer type classifier for layer three

When checking the results most of the time it predicted correctly still.

```
LayTypePL=[]
#le.fit(unqValues[2])
for i in range(len(X_test)):
    LayType=np.argmax(y_resultLType[i])
    LayTypePL.append(LayType)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leLType.inverse_transform(LayTypePL)
Y_testLI=leLType.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=Dense, Predicted=Dense
Actual=Other, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=LSTM, Predicted=Bi-LSTM
Actual=Dense, Predicted=Dense
Actual=LSTM, Predicted=Dense
```

Figure 6.90Predicted values of layer type classifier for layer three

6.4.7.4 Layer type classifier for layer 04

Classifier configuration has end up with following classifier for the layer four in layer type.

```
modelLType4p = Sequential()
modelLType4p.add(Dense(68, activation='relu', input_dim=68))
modelLType4p.add(Dense(34, activation='relu'))
modelLType4p.add(Dense(17, activation='relu'))
modelLType4p.add(Dense(7, activation='softmax'))
modelLType4p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 80
epochs = 100
```

Figure 6.91Neural network for layer type classifier for layer four

Validation loss also become same as loss in the following loss grapgh.

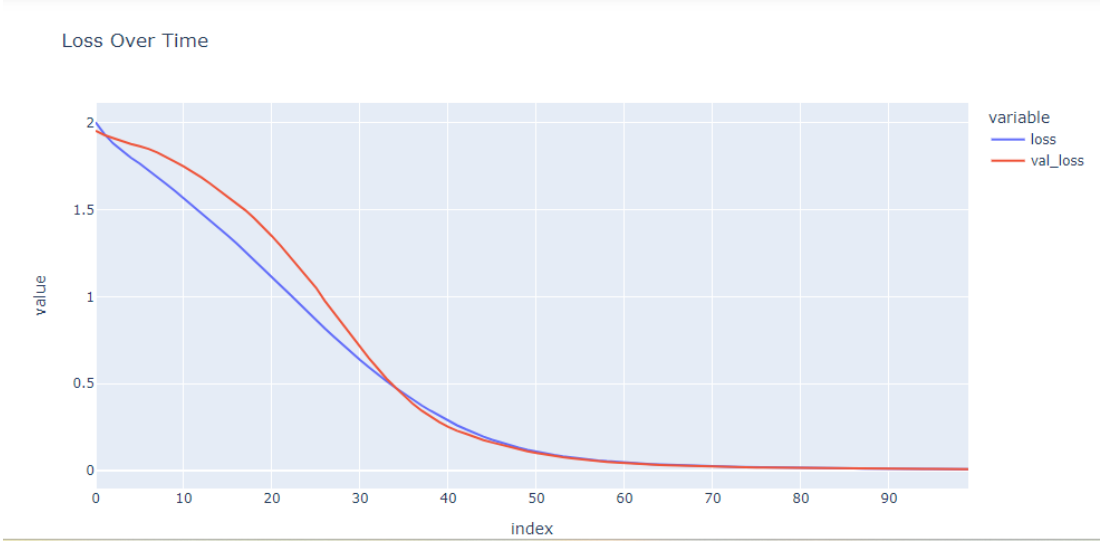


Figure 6.92Lost function for layer type classifier for layer four

For the accuracy the validation accuracy is high than the accuracy as follows.

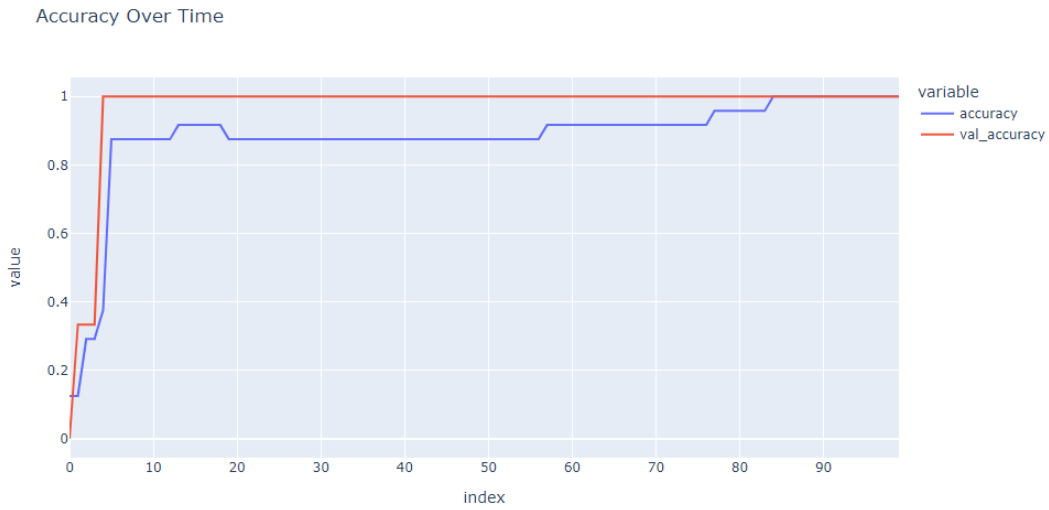


Figure 6.93Accuracy of layer type classifier for layer four

The accuracy increases only for 40% no matter what that done happens due to the low number of actual recodes. Most neural networks have 4 layers in this case all the final layers are taken separately resulting in the 4 layers in here.

```
modelLType4p.evaluate(X_test, to_categorical(Y_test,7))
```

```
1/1 [==============================] - 0s 24ms/step - loss: 1.7936 - accuracy: 0.4000
[1.7936365604400635, 0.4000000059604645]
```

```
y_resultLType=modelLType4p.predict(X_test)
```

```
1/1 [==============================] - 0s 52ms/step
```

Figure 6.94Evaluation and prediction of layer type classifier for layer four

When considering the actual and the predicted values this is reflecting too.

```
LayTypePL=[]
#le.fit(unqValues[2])
for i in range(len(X_test)):
    LayType=np.argmax(y_resultLType[i])
    LayTypePL.append(LayType)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leLType.inverse_transform(LayTypePL)
Y_testLI=leLType.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=Other, Predicted=Dense
Actual=Dense, Predicted=GlobalMaxPooling1D
Actual=Dense, Predicted=GlobalMaxPooling1D
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
```

Figure 6.95Predicted values of layer type classifier for layer four

6.4.7.5 Layer type classifier for layer 05

The nural network with 7 deferant classes for classify as previous layer type predictor as been created as follows.

```
modelLType5p = Sequential()
modelLType5p.add(Dense(68, activation='relu', input_dim=68))
modelLType5p.add(Dense(34, activation='relu'))
modelLType5p.add(Dense(17, activation='relu'))
modelLType5p.add(Dense(7, activation='softmax'))
modelLType5p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 80
epochs = 100
```

Figure 6.96Neural network for layer type classifier for layer five

Following loss over time graph has acived by traing of above neural network with the acuracy with time.

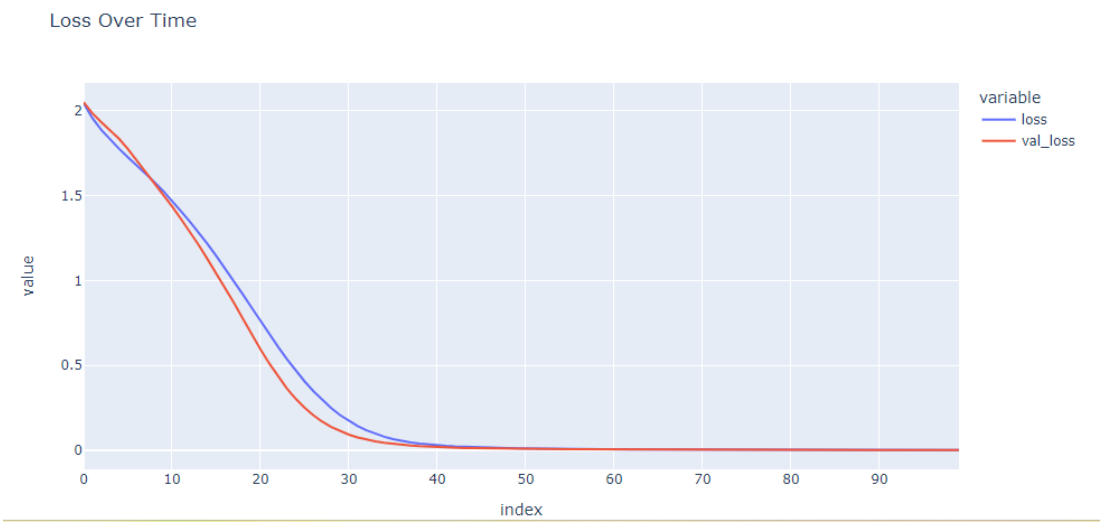


Figure 6.97Lost function for layer type classifier for layer five

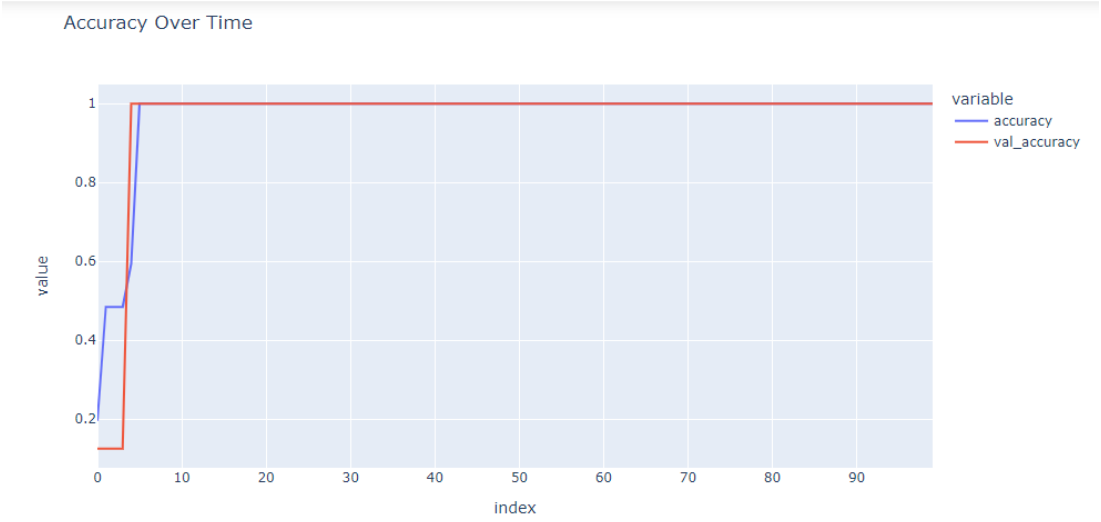


Figure 6.98Accuracy of layer type classifier for layer five

Due to less variation in the teting set accuracy is displayed as 100%

```
1 modelLType5p.evaluate(X_test, to_categorical(Y_test,7))
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.0000e+00 - accuracy: 1.0000
[0.0, 1.0]
```

```
1 y_resultLType=modelLType5p.predict(X_test)
```

```
2/2 [==============================] - 0s 4ms/step
```

Figure 6.99Evaluation and prediction of layer type classifier for layer five

The above issue is reflacting by the values. However the future situation occer due to this weakness will handle by the layer cration restictors.

```
LayTypePL=[]
#le.fit(unqValues[2])
for i in range(len(X_test)):
    LayType=np.argmax(y_resultLType[i])
    LayTypePL.append(LayType)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leLType.inverse_transform(LayTypePL)
Y_testLI=leLType.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=Other, Predicted=Other
Actual=Other, Predicted=Other
Actual=Other, Predicted=Other
```

Figure 6.100Predicted values of layer type classifier for layer five

6.4.7.6 Layer type classifier for layer 06

This is the creation of the layer type classifier for the final layer of the neural network configuration

```
modelLTypeLp = Sequential()
modelLTypeLp.add(Dense(68, activation='relu', input_dim=68))
modelLTypeLp.add(Dense(34, activation='relu'))
modelLTypeLp.add(Dense(17, activation='relu'))
modelLTypeLp.add(Dense(7, activation='softmax'))
modelLTypeLp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 80
epochs = 100
```

Figure 6.101Neural network for layer type classifier for layer six

The following two grapgh demonstrate the lost and the accuracy over time with the training of the neural network configuration.

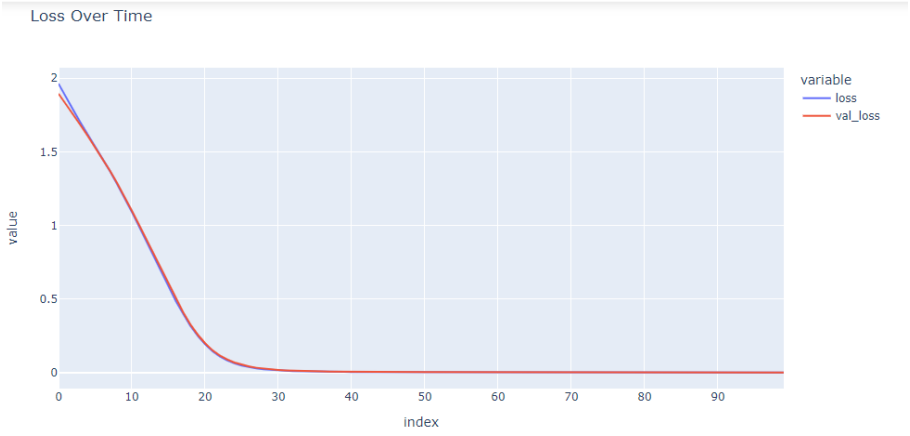


Figure 6.102Lost function for layer type classifier for layer six

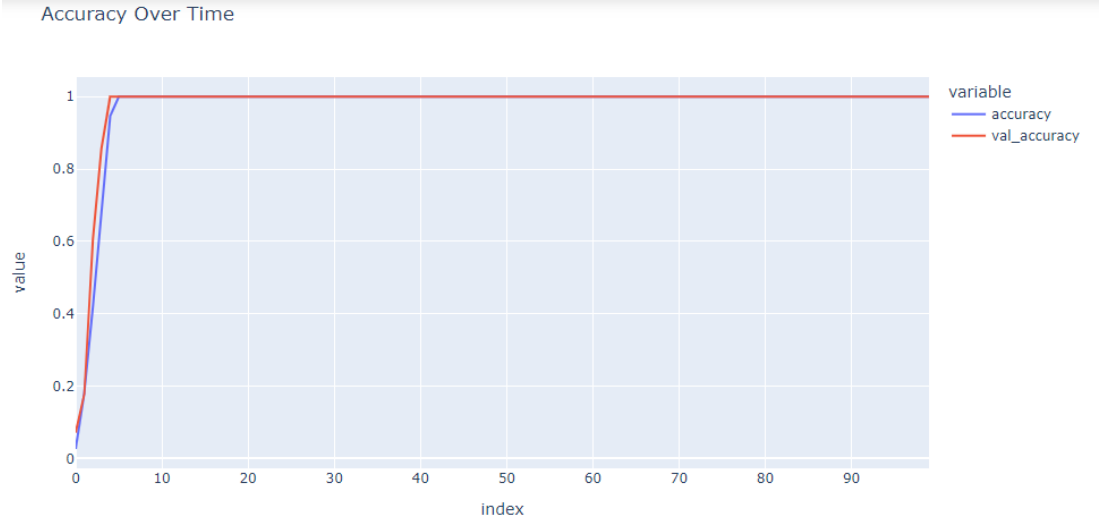


Figure 6.103Accuracy of layer type classifier for layer six

Here the accuracy of the classifier is high due to the final layers of all the 40 problems coming here. The accuracy becomes 100% due to less variation in the testing set. For almost all the neural nets the last layer is a dense layer which is a pattern.

```
1 modelLTypeLp.evaluate(X_test, to_categorical(Y_test,7))
```

```
1/1 [==============================] - 0s 22ms/step - loss: 0.0041 - accuracy: 1.0000
[0.0041272207163274 29, 1.0]
```

```
1 y_resultLType=modelLTypeLp.predict(X_test)
```

```
1/1 [==============================] - 0s 71ms/step
```

Figure 6.104Evaluation and prediction of layer type classifier for layer six

The above explaind fact is confirmed by the actual and predicted values.

```
:
1  LayTypePL=[]
2  #le.fit(unqValues[2])
3  for i in range(len(X_test)):
4      LayType=np.argmax(y_resultLType[i])
5      LayTypePL.append(LayType)
6      #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
7  y_resultLTypeLI=leLType.inverse_transform(LayTypePL)
8  Y_testLI=leLType.inverse_transform(Y_test)
9  for i in range(len(X_test)):
10
11     print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
Actual=Dense, Predicted=Dense
```

Figure 6.105Predicted values of layer type classifier for layer six

6.4.8 Number of neurons predictor

Like the layer type hyperparameter number of neurons should be predicted layer-wise. Prediction happens in up to six layers as follows.

6.4.8.1 Number of neurons Predictor for layer 01

Number of neurons predictor for the first layer happen as follows with the configuration.

```
modelnNPrdfL1 = Sequential()
modelnNPrdfL1.add(Dense(70, input_shape=(70,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelnNPrdfL1.add(Dense(18, activation='relu'))
modelnNPrdfL1.add(Dense(9, activation='relu'))
modelnNPrdfL1.add(Dense(1))#, activation='linear'
modelnNPrdfL1.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.106Neural network for number of neurons predictor for layer one

Training of the above regression approach results in following loss with the time

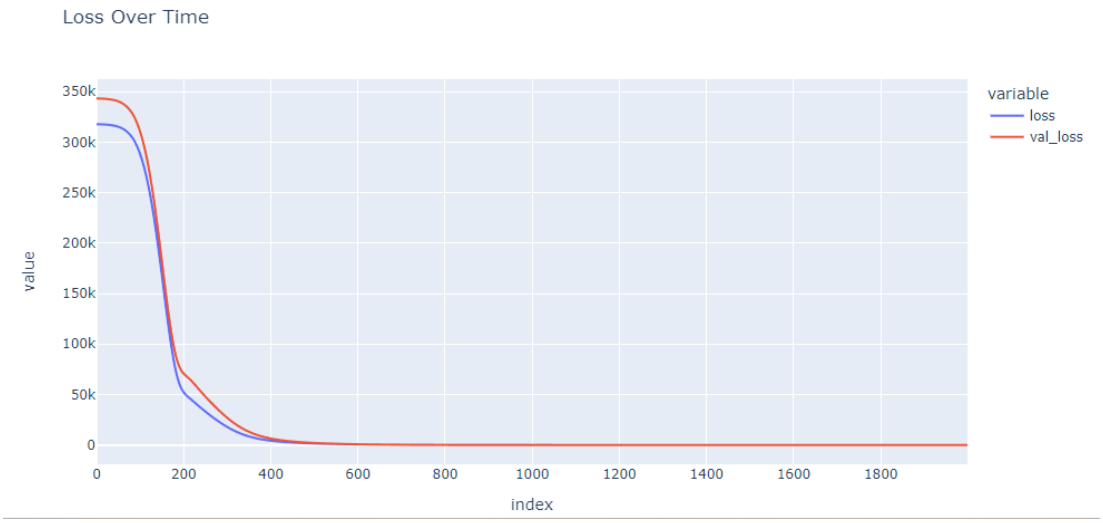


Figure 6.107Lost function for number of neurons predictor for layer one

The final loss of the trained neural network is 13526, which has been acquired because the insert values are large even though that used the min max scaler.

```
modelnNPrdfL1.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 38ms/step - loss: 13526.6182

13526.6181640625
```

```
y_result=modelnNPrdfL1.predict(X_test)
```

```
1/1 [==============================] - 0s 77ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_result[i]).astype(int)[0]))
```

```
Actual=8, Predicted=23
Actual=22, Predicted=11
Actual=0, Predicted=54
```

Figure 6.108Evaluation and prediction, prediction values of the number of neuron predictor for layer one

6.4.8.2 Number of neurons Predictor for layer 02

The number of neuron predictor develop as the following configuration for the second layer

```
modelnNPrdfL2 = Sequential()
modelnNPrdfL2.add(Dense(70, input_shape=(70,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelnNPrdfL2.add(Dense(18, activation='relu'))
modelnNPrdfL2.add(Dense(9, activation='relu'))
modelnNPrdfL2.add(Dense(1))#, activation='linear'
modelnNPrdfL2.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.109Neural network for number of neurons predictor for layer two

The trained model gives the following lost graph during the training.

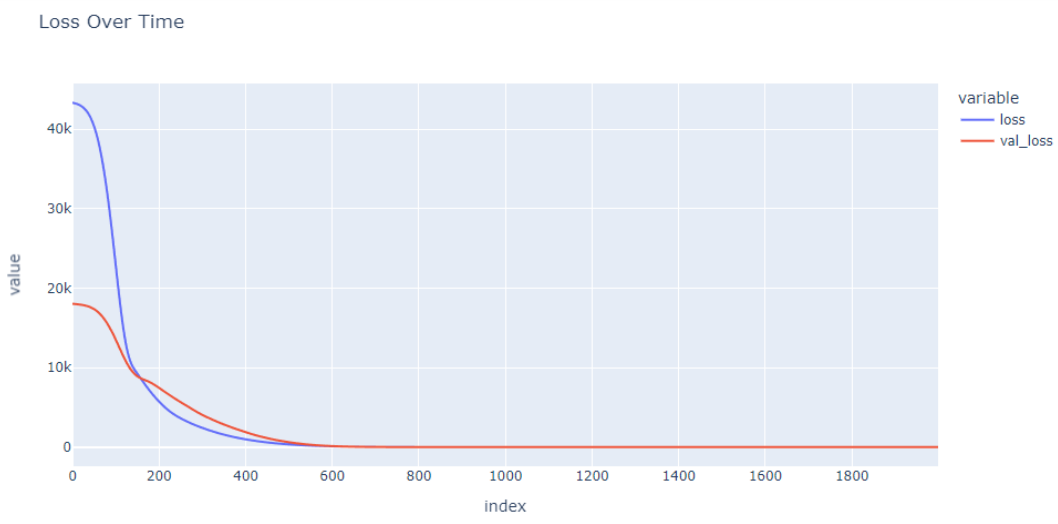


Figure 6.110Lost function for number of neurons predictor for layer two

The lost value resulting from the trained neural network is 3738. However, the resultant values are acceptable when considering the actual and predicted values.

```
modelnNPrdfL2.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 23ms/step - loss: 3738.5547

3738.5546875
```

```
y_result=modelnNPrdfL2.predict(X_test)
```

```
1/1 [==============================] - 0s 59ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_result[i]).astype(int)[0]))
```

```
Actual=100, Predicted=79
Actual=32, Predicted=29
Actual=0, Predicted=86
```

Figure 6.111Evaluation and prediction, prediction values of the number of neuron predictors for layer two

6.4.8.3 Number of neurons predictor for layer 03

The number of neurons predictor for layer three configuraton is as follows.

```
modelnNPrdfL3 = Sequential()
modelnNPrdfL3.add(Dense(70, input_shape=(70,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelnNPrdfL3.add(Dense(18, activation='relu'))
modelnNPrdfL3.add(Dense(9, activation='relu'))
modelnNPrdfL3.add(Dense(1))#, activation='linear'
modelnNPrdfL3.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.112Neural network for number of neurons predictor for layer three

The resultant loss function is as follows.

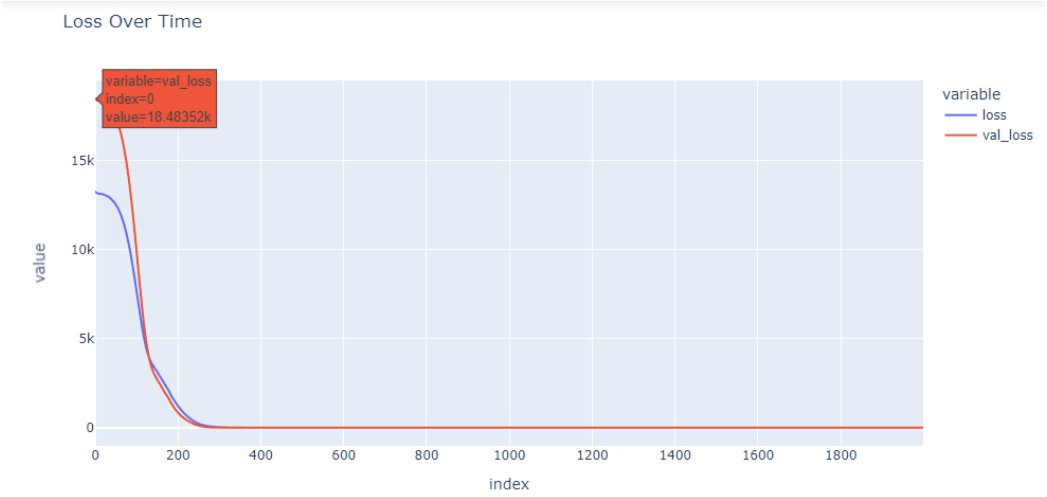


Figure 6.113Lost function for number of neurons predictor for layer three

The lost values of the tranined nural network is 900 due to the inserted lage values. Also the resultant values are acceptable.

```
1  modelnNPrdfL3.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 28ms/step - loss: 900.4302

900.4302368164062
```

```
1  y_result=modelnNPrdfL3.predict(X_test)
```

```
1/1 [==============================] - 0s 129ms/step
```

```
1  for i in range(len(X_test)):
2
3      print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_resul
```

```
Actual=24, Predicted=9
Actual=32, Predicted=34
Actual=64, Predicted=53
```

Figure 6.114Evaluation and prediction, prediction values of the number of neuron predictors for layer three

6.4.8.4 Number of neurons Predictor for layer 04

The following configuration has been applied to achieve the neurons predictor for layer four.

```
modelnNPrdfL4 = Sequential()
modelnNPrdfL4.add(Dense(70, input_shape=(70,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelnNPrdfL4.add(Dense(18, activation='relu'))
modelnNPrdfL4.add(Dense(9, activation='relu'))
modelnNPrdfL4.add(Dense(1))#, activation='linear'
modelnNPrdfL4.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.115Neural network for number of neurons predictor for layer four

The loss over time has been achieved as follows.

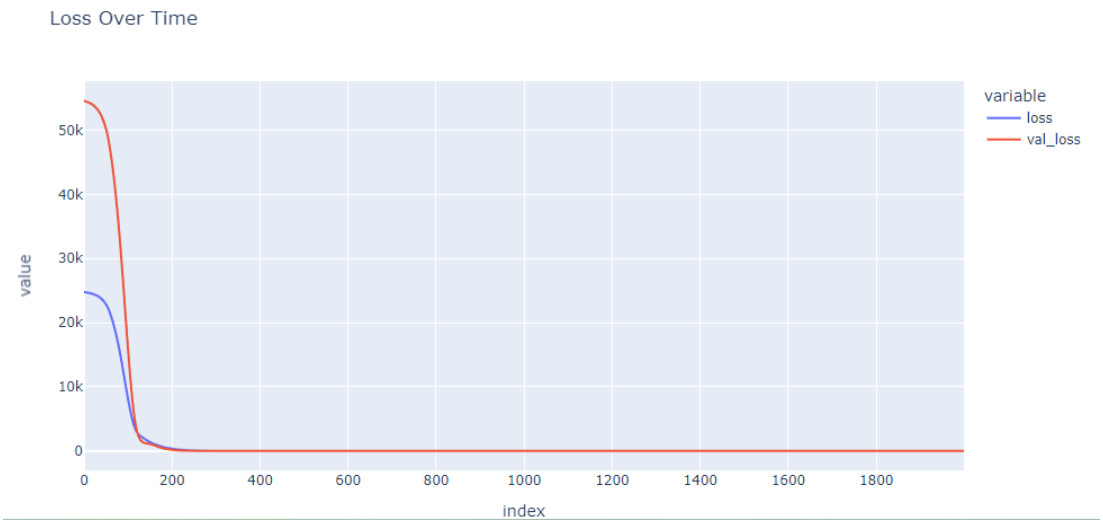


Figure 6.116Lost function for number of neurons predictor for layer four

The final lost value is achieved as follows. The predicted value is much equal to the actual values.

```
modelnNPrdfL4.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 20ms/step - loss: 12392.8076
12392.8076171875
```

```
y_result=modelnNPrdfL4.predict(X_test)
```

```
1/1 [==============================] - 0s 73ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_result[i]).astype(int)[0]))
```

```
Actual=0, Predicted=0
Actual=8, Predicted=8
Actual=256, Predicted=10
```

Figure 6.117Evaluation and prediction, prediction values of the number of neuron predictors for layer four

6.4.8.5 Number of neurons Predictor for layer 05

When considering the configuration of neuron predictor for layer five it also has 4 layers as before.

```
modelnNPrdfL5 = Sequential()
modelnNPrdfL5.add(Dense(70, input_shape=(70,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelnNPrdfL5.add(Dense(18, activation='relu'))
modelnNPrdfL5.add(Dense(9, activation='relu'))
modelnNPrdfL5.add(Dense(1))#, activation='linear'
modelnNPrdfL5.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.118Neural network for number of neurons predictor for layer five

The lost with time wile training the neural network results as follows.

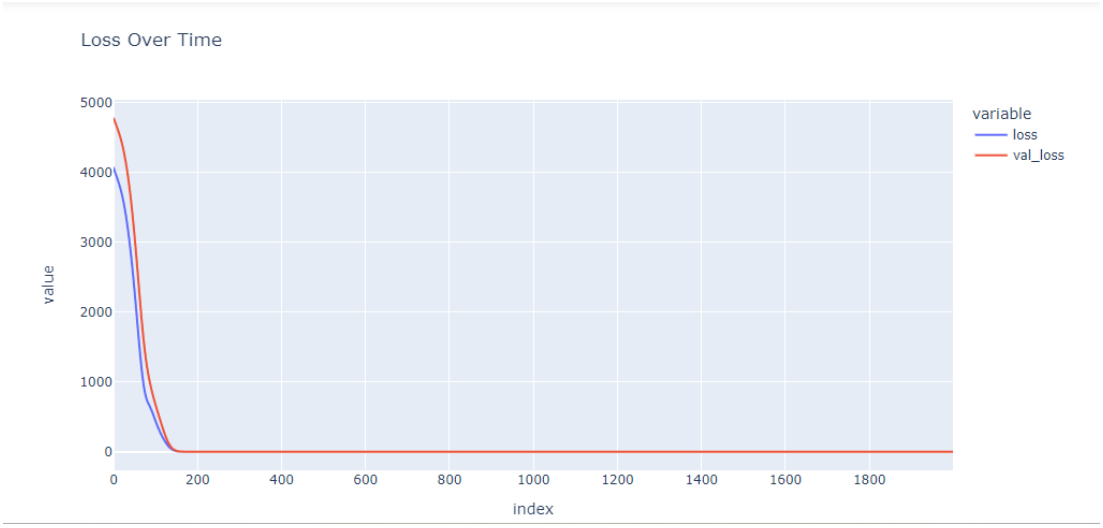


Figure 6.119Lost function for number of neurons predictor for layer five

The resultant lost value is 558 is resulting due to the large values that has inserted to the neural network.

```
modelnNPrdfL5.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 179ms/step - loss: 558.5732

558.5731811523438
```

```
y_result=modelnNPrdfL5.predict(X_test)
```

```
1/1 [==============================] - 0s 168ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_result[i]).astype(int)[0]))
```

```
Actual=0, Predicted=13
Actual=0, Predicted=36
Actual=0, Predicted=13
```

Figure 6.120Evaluation and prediction, prediction values of the number of neuron predictors for layer five

6.4.8.6 Number of neurons Predictor for layer 06

The number of neuron predictor configurations for the model is as follows.

```
modelnNPrdfLL = Sequential()
modelnNPrdfLL.add(Dense(70, input_shape=(70,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modelnNPrdfLL.add(Dense(18, activation='relu'))
modelnNPrdfLL.add(Dense(9, activation='relu'))
modelnNPrdfLL.add(Dense(1))#, activation='linear'
modelnNPrdfLL.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.121Neural network for number of neurons predictor for layer six

The lost function behavied as follows when the training happen to above model.

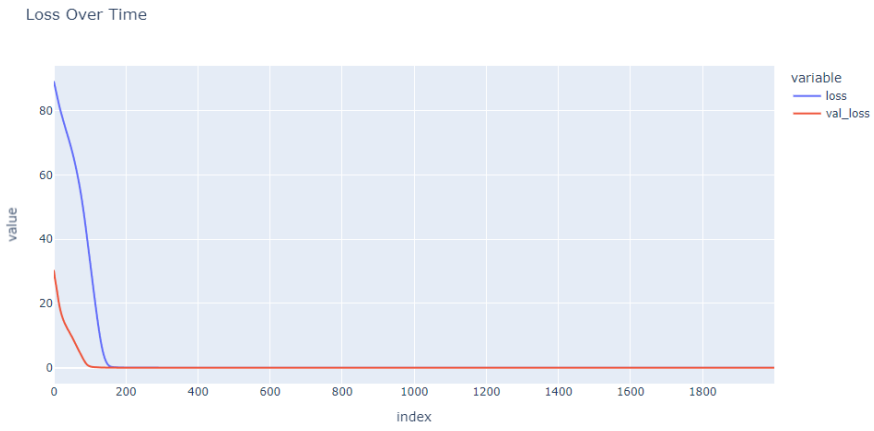


Figure 6.122Lost function for number of neurons predictor for layer six

The resultant loss is 1.92 for final layer predictor model. The predicted value and the original valus are almost same.

```
modelnNPrdfLL.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 40ms/step - loss: 1.9276
1.9275975227355957
```

```
y_result=modelnNPrdfLL.predict(X_test)
```

```
1/1 [==============================] - 0s 102ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], round_up(y_result[i]).astype(int)[0]))
```

```
Actual=2, Predicted=1
Actual=1, Predicted=2
Actual=1, Predicted=3
Actual=3, Predicted=1
Actual=1, Predicted=1
Actual=1, Predicted=1
Actual=1, Predicted=1
Actual=1, Predicted=1
Actual=1, Predicted=0
Actual=1, Predicted=1
Actual=2, Predicted=3
```

Figure 6.123Evaluation and prediction, prediction values of the number of neuron predictors for layer six

6.4.9 Dropout predictor

As previous two hyperparameters the dropout also shild predict layer wise.

6.4.9.1 Dropout predictor for layer 01

Configuraltion for the layer one dropout predictor as follows.

```
modeldropPrdfL1 = Sequential()
modeldropPrdfL1.add(Dense(70, input_shape=(71,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modeldropPrdfL1.add(Dense(18, activation='relu'))
modeldropPrdfL1.add(Dense(9, activation='relu'))
modeldropPrdfL1.add(Dense(1))#, activation='linear'
modeldropPrdfL1.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.124Neural network for dropout predictor for layer one

Behaviors of the lost function are as follows for the above configuration.

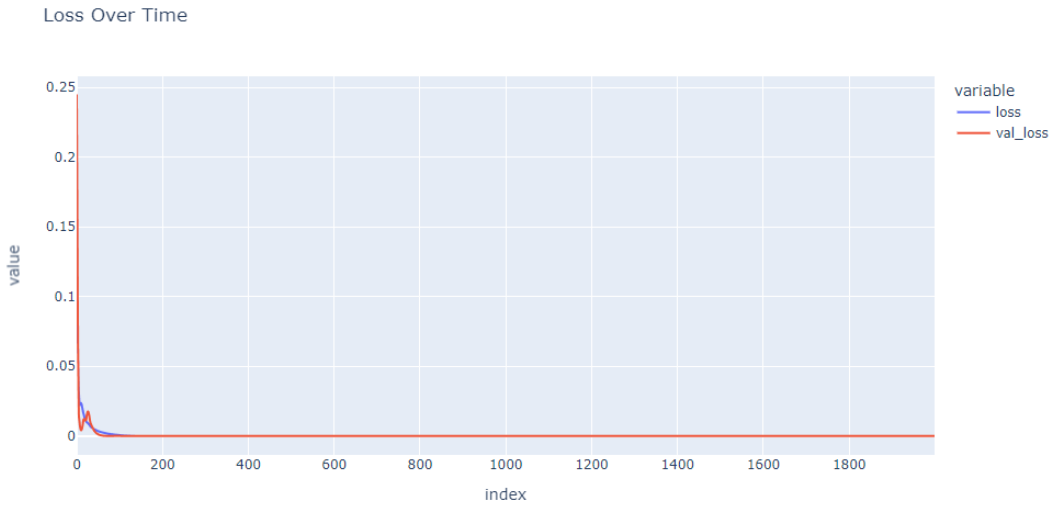


Figure 6.125Lost function for dropout predictor for layer one

Loast is 0.0071 due to small values insert to the nural network. And the predicted values are acceptable when compare with actual values.

```
: 1 modeldropPrdfL1.evaluate(X_test,Y_test)

1/1 [==============================] - 0s 45ms/step - loss: 0.0071

: 0.007131436839699745

: 1 y_result=modeldropPrdfL1.predict(X_test)

1/1 [==============================] - 0s 112ms/step

: 1 for i in range(len(X_test)):
  2
  3     print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))

Actual=0.0, Predicted=0.05
Actual=0.0, Predicted=0.03
Actual=0.0, Predicted=0.02
Actual=0.0, Predicted=0.11
Actual=0.0, Predicted=0.05
Actual=0.0, Predicted=0.1
```

Figure 6.126Evaluation and prediction, prediction values of dropout predictor for layer one

6.4.9.2 Dropout predictor for layer 02

Configuration for the layer two dropout predictor as follows.

```
modeldropPrdfL2 = Sequential()
modeldropPrdfL2.add(Dense(70, input_shape=(71,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modeldropPrdfL2.add(Dense(18, activation='relu'))
modeldropPrdfL2.add(Dense(9, activation='relu'))
modeldropPrdfL2.add(Dense(1))#, activation='linear'
modeldropPrdfL2.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.127Neural network for dropout predictor for layer two

The behavior of the lost function is as follows for the above configuration.

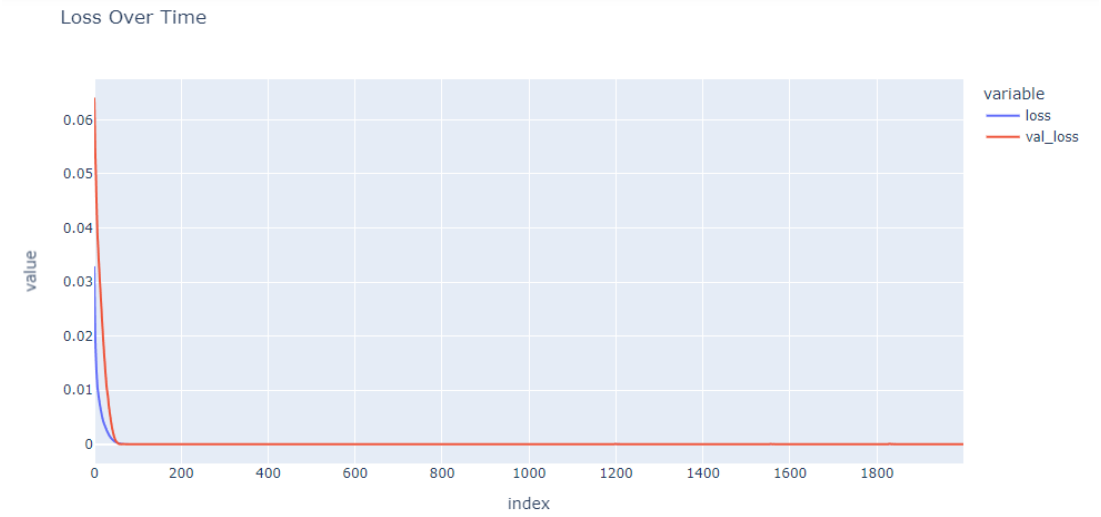


Figure 6.128 Lost function for dropout predictor for layer two

Lost is 0.0561 due to small values insert to the nural network. And the predicted values are acceptable when compare with actual values in most cases.

```
modeldropPrdfL2.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 28ms/step - loss: 0.0561

0.05610079690814018
```

```
y_result=modeldropPrdfL2.predict(X_test)
```

```
1/1 [==============================] - 0s 96ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))
```

```
Actual=0.2, Predicted=0.16
Actual=0.5, Predicted=0.2
Actual=0.0, Predicted=0.21
Actual=0.0, Predicted=0.02
Actual=0.0, Predicted=0.12
Actual=0.5, Predicted=0.18
```

Figure 6.129Evaluation and prediction, prediction values of dropout predictor for layer two

6.4.9.3 Dropout predictor for layer 03

Configuration for the layer three dropout predictor as follows.

```
modeldropPrdfL3 = Sequential()
modeldropPrdfL3.add(Dense(70, input_shape=(71,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modeldropPrdfL3.add(Dense(18, activation='relu'))
modeldropPrdfL3.add(Dense(9, activation='relu'))
modeldropPrdfL3.add(Dense(1))#, activation='linear'
modeldropPrdfL3.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.130Neural network for dropout predictor for layer three

The behavior of the lost function is as follows for the above configuration.

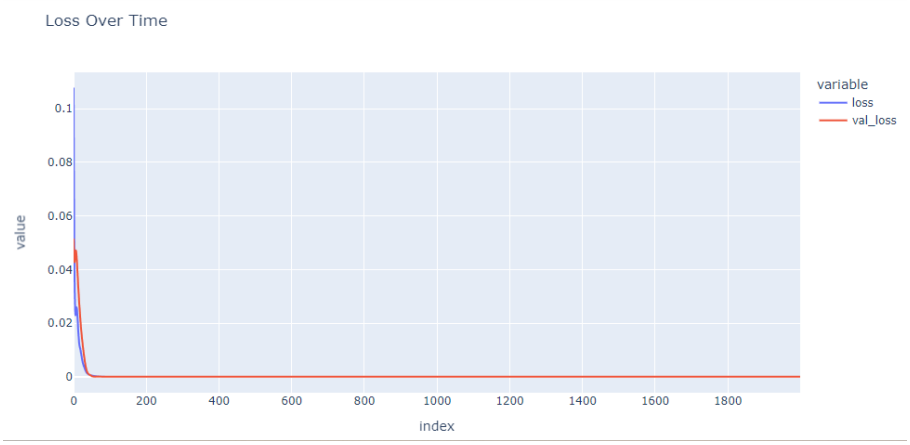


Figure 6.131Lost function for dropout predictor for layer three

Lost is 0.0198 due to small values insert to the nural network. And the predicted values are acceptable when compare with actual values in most cases.

```
1  modeldropPrdfL3.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 40ms/step - loss: 0.0198

0.019795924425125122
```

```
1  y_result=modeldropPrdfL3.predict(X_test)
```

```
1/1 [==============================] - 0s 61ms/step
```

```
1  for i in range(len(X_test)):
2
3      print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))
```

```
Actual=0.0, Predicted=0.11
Actual=0.0, Predicted=0.24
Actual=0.0, Predicted=0.11
Actual=0.0, Predicted=0.0
Actual=0.0, Predicted=0.01
Actual=0.0, Predicted=0.09
Actual=0.0, Predicted=0.27
Actual=0.0, Predicted=0.01
```

Figure 6.132Evaluation and prediction, prediction values of dropout predictor for layer three

6.4.9.4 Dropout predictor for layer 04

configuration for the layer four dropout predictor as follows.

```
modeldropPrdfL4 = Sequential()
modeldropPrdfL4.add(Dense(70, input_shape=(71,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modeldropPrdfL4.add(Dense(18, activation='relu'))
modeldropPrdfL4.add(Dense(9, activation='relu'))
modeldropPrdfL4.add(Dense(1))#, activation='linear'
modeldropPrdfL4.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.133Neural network for dropout predictor for layer four

The behavior of the lost function is as follows for the above configuration.

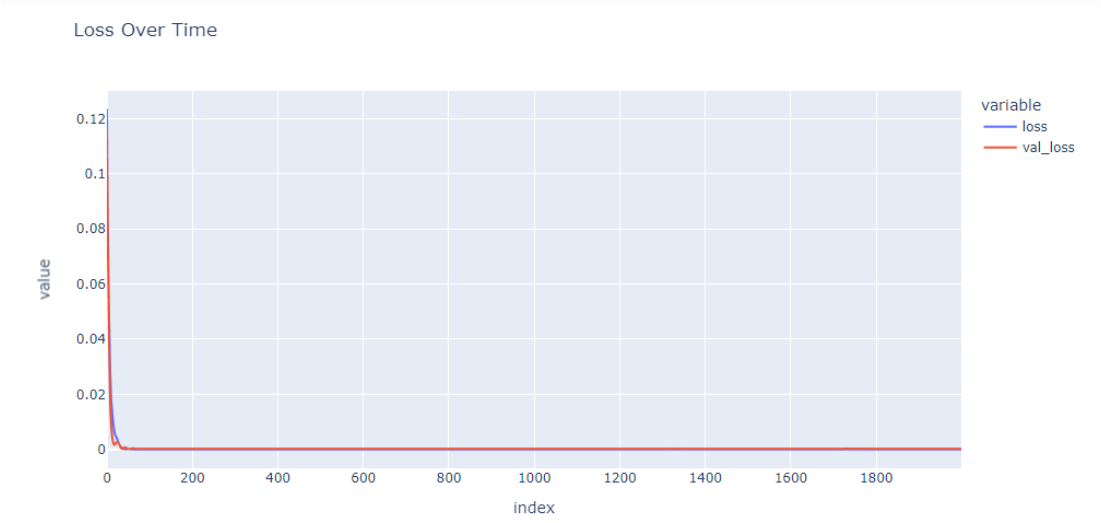


Figure 6.134Lost function for dropout predictor for layer four

Lost is 0.231 due to small values insert to the nural network. And the predicted values are acceptable when compare with actual values in most cases.

```
modeldropPrdfL4.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 22ms/step - loss: 0.2317

0.23165011405944824
```

```
y_result=modeldropPrdfL4.predict(X_test)
```

```
1/1 [==============================] - 0s 80ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))
```

```
Actual=0.35, Predicted=0.21
Actual=0.0, Predicted=0.11
Actual=0.5, Predicted=0.17
Actual=0.0, Predicted=0.16
Actual=0.0, Predicted=0.37
Actual=0.0, Predicted=0.11
```

Figure 6.135Evaluation and prediction, prediction values of dropout predictor for layer four

6.4.9.5 Dropout predictor for layer 05

Configuration for the layer five dropout predictor is as follows.

```
modeldropPrdfL5 = Sequential()
modeldropPrdfL5.add(Dense(70, input_shape=(71,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modeldropPrdfL5.add(Dense(18, activation='relu'))
modeldropPrdfL5.add(Dense(9, activation='relu'))
modeldropPrdfL5.add(Dense(1))#, activation='linear'
modeldropPrdfL5.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.136Neural network for dropout predictor for layer five

The behavior of the lost function is as follows for the above configuration.

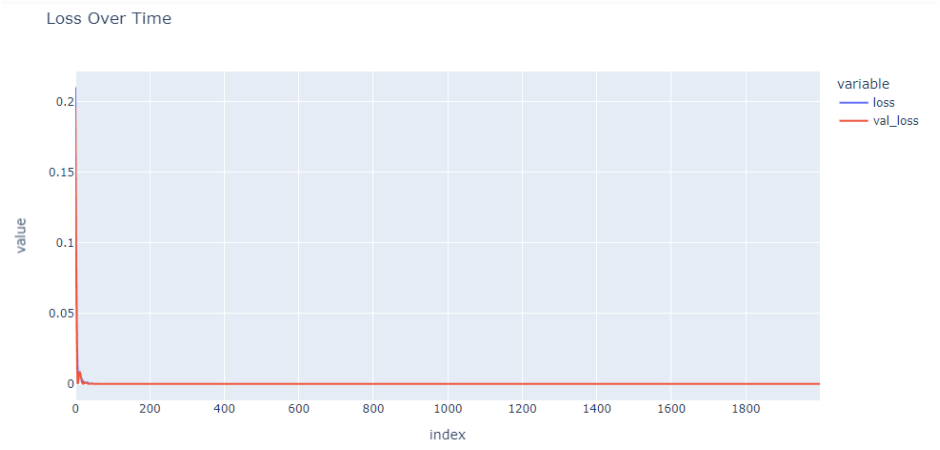


Figure 6.137Lost function for dropout predictor for layer five

Lost is 0.0109 due to small values insert to the nural network. And the predicted values are acceptable when compare with actual values in most cases.

```
modeldropPrdfL5.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 50ms/step - loss: 0.0109

0.0108580207452178
```

```
y_result=modeldropPrdfL5.predict(X_test)
```

```
1/1 [==============================] - 0s 59ms/step
```

```
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))
```

```
Actual=0.0, Predicted=0.1
Actual=0.0, Predicted=-0.12
Actual=0.0, Predicted=0.1
```

Figure 6.138Evaluation and prediction, prediction values of dropout predictor for layer five

6.4.9.6 Dropout predictor for layer 06

Configuration for the last layer dropout predictor is as follows.

```
modeldropPrdfLL = Sequential()
modeldropPrdfLL.add(Dense(70, input_shape=(71,), activation='relu'))
#modelRe.add(Dense(33, activation='relu'))
modeldropPrdfLL.add(Dense(18, activation='relu'))
modeldropPrdfLL.add(Dense(9, activation='relu'))
modeldropPrdfLL.add(Dense(1))#, activation='linear'
modeldropPrdfLL.compile(loss='mse', optimizer='adam')
```

```
batch_size = 200
epochs = 2000
```

Figure 6.139Neural network for dropout predictor for layer six

The behavior of the lost function is as follows for the above configuration.

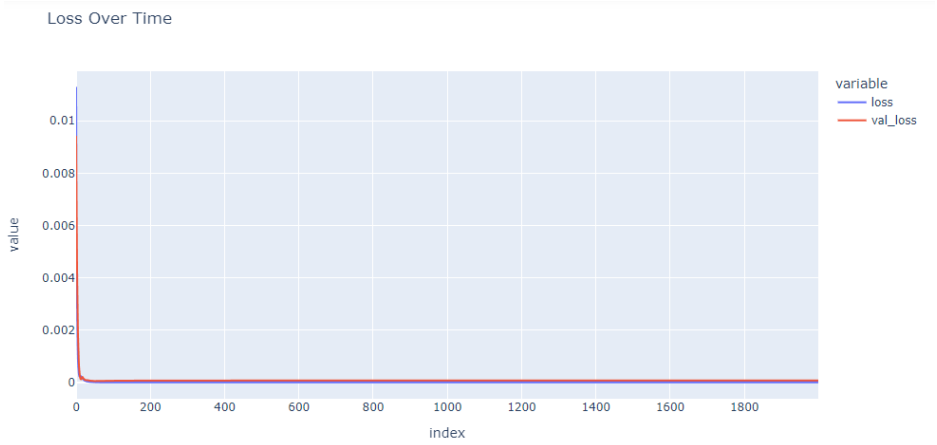


Figure 6.140Lost function for dropout predictor for layer six

Lost is 0.000404 due to small values inserted into the neural network. And the predicted values are acceptable when compare with actual values in most cases.

```
1 modeldropPrdfLL.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 34ms/step - loss: 4.0467e-04
0.00040467182407155633
```

```
1 y_result=modeldropPrdfLL.predict(X_test)
```

```
1/1 [==============================] - 0s 65ms/step
```

```
1 for i in range(len(X_test)):
2
3     print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))
```

```
Actual=0.0, Predicted=0.02
Actual=0.0, Predicted=0.01
Actual=0.0, Predicted=-0.0
Actual=0.0, Predicted=-0.0
Actual=0.0, Predicted=0.05
```

Figure 6.141Evaluation and prediction, prediction values of dropout predictor for layer six

6.4.10 Activation function classifier

A like before the activation function also should be classified for each of the 6 layers

6.4.10.1 Activation function classifier for layer 01

For the first layer configuration of the activation function has done as follows.

```
modelaFunL1p = Sequential()
modelaFunL1p.add(Dense(72, activation='relu', input_dim=72))
modelaFunL1p.add(Dense(36, activation='relu'))
modelaFunL1p.add(Dense(18, activation='relu'))
modelaFunL1p.add(Dense(9, activation='softmax'))
modelaFunL1p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 120
epochs = 500
```

Figure 6.142Neural network for activation function classifier for layer one

While the training the loss has converge as follows and the accuracy has behaved as diagram below

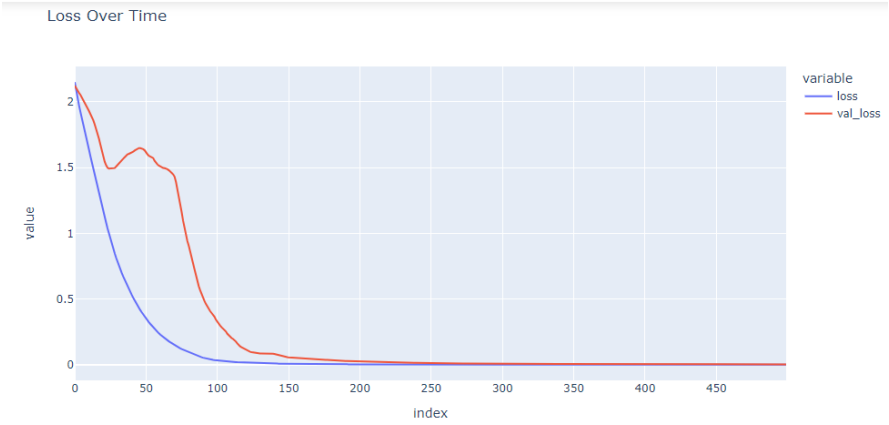


Figure 6.143Lost function for activation function classifier for layer one

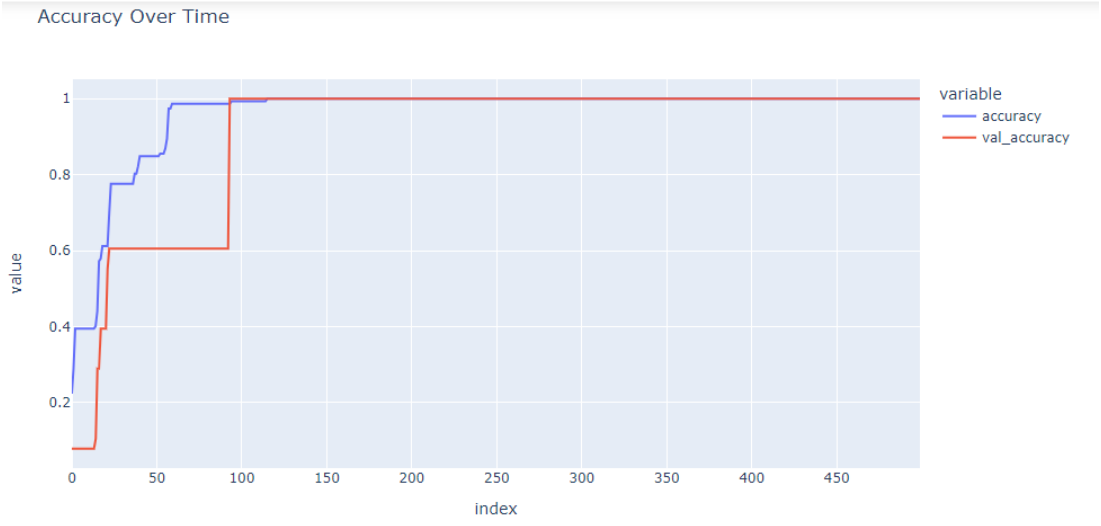


Figure 6.144Accuracy of activation function classifier for layer one

After the evaluation the loss and the accuracy achieved as follows.

```
modelaFunL1p.evaluate(X_test, to_categorical(Y_test,9))
```

```
1/1 [==============================] - 0s 61ms/step - loss: 1.8882 - accuracy: 0.8333
[1.8882094621658325, 0.8333333134651184]
```

```
y_resultLType=modelaFunL1p.predict(X_test)
```

```
1/1 [==============================] - 0s 164ms/step
```

Figure 6.145Evaluation and prediction of activation function classifier for layer one

In many of the cases actual and classified value is same and for the cases that are not are handling by the model restrictors.

```
aFunPL=[]
leGenrl.fit(unqValues[4])
for i in range(len(X_test)):
    aFun=np.argmax(y_resultLType[i])
    aFunPL.append(aFun)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
#aFunPrL.append(aFunPL)
y_resultLTypeLI=leGenrl.inverse_transform(aFunPL)
Y_testLI=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=relu, Predicted=relu
Actual=nan, Predicted=nan
Actual=nan, Predicted=nan
Actual=nan, Predicted=nan
Actual=relu, Predicted=relu
Actual=relu, Predicted=relu
Actual=nan, Predicted=nan
Actual=nan, Predicted=nan
Actual=relu, Predicted=nan
Actual=relu, Predicted=relu
```

Figure 6.146 Predicted values of activation function classifier for layer one

6.4.10.2 Activation function classifier for layer 02

For the second layer configuration of the activation function has done as follows.

```
modelaFunL2p = Sequential()
modelaFunL2p.add(Dense(72, activation='relu', input_dim=72))
modelaFunL2p.add(Dense(36, activation='relu'))
modelaFunL2p.add(Dense(18, activation='relu'))
modelaFunL2p.add(Dense(9, activation='softmax'))
modelaFunL2p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 120
epochs = 500
```

Figure 6.147Neural network for activation function classifier for layer two

While the training the loss has converge as follows and the accuracy has behaved as diagram below

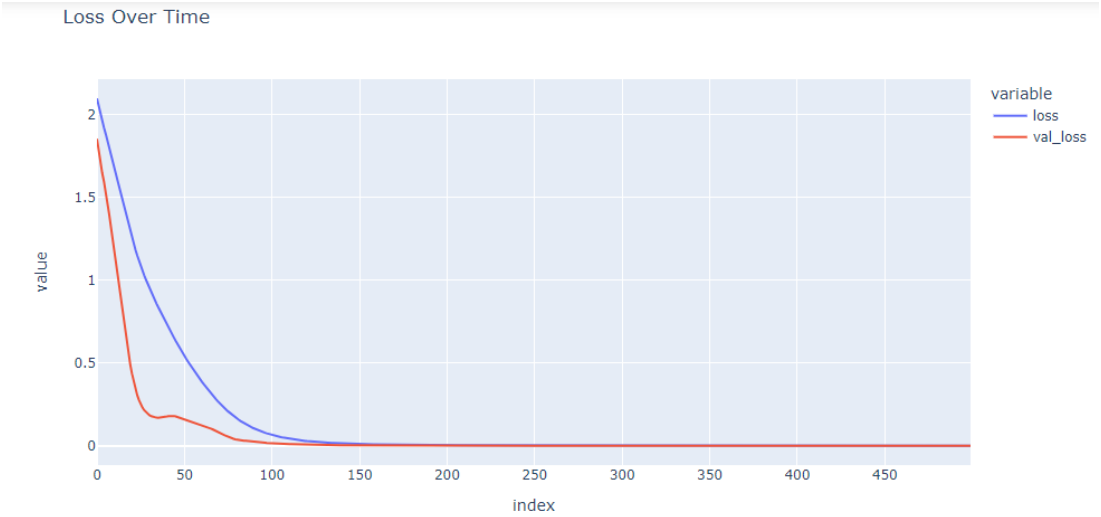


Figure 6.148Lost function for activation function classifier for layer two

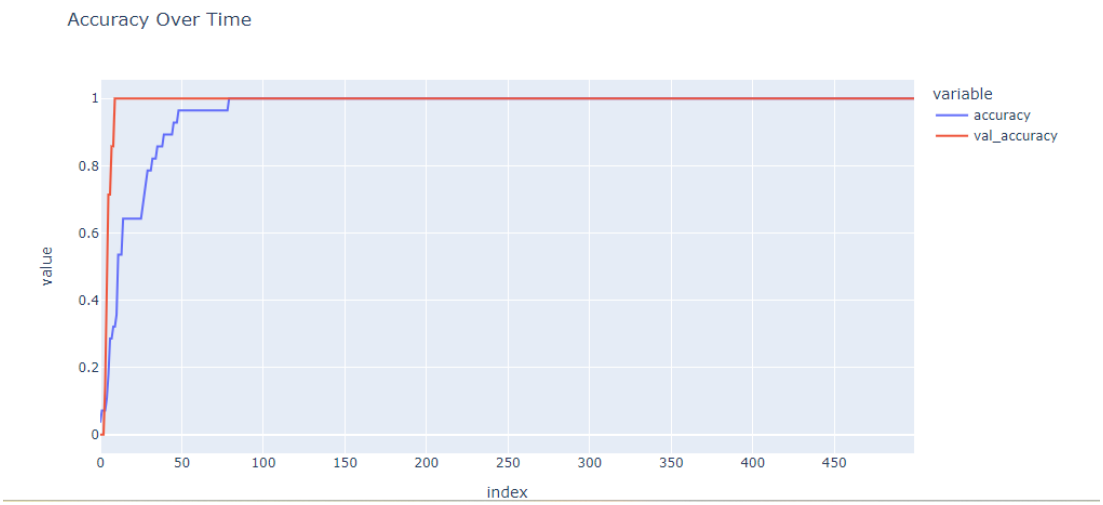


Figure 6.149Accuracy of activation function classifier for layer two

After the evaluation, the lost and the accuracy achieved as follows.

```
modelaFunL2p.evaluate(X_test, to_categorical(Y_test,9))
```

```
1/1 [==============================] - 0s 33ms/step - loss: 1.7774 - accuracy: 0.6667
[1.777388095855713, 0.6666666865348816]
```

```
y_resultLType=modelaFunL2p.predict(X_test)
```

```
1/1 [==============================] - 0s 57ms/step
```

Figure 6.150 Evaluation and prediction of activation function classifier for layer two

Many of the cases actual and classified value is same and for the cases that are not are handling by the model restrictors.

```
aFunPL=[]
#le.fit(unqValues[4])
for i in range(len(X_test)):
    aFun=np.argmax(y_resultLType[i])
    aFunPL.append(aFun)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leGenrl.inverse_transform(aFunPL)
Y_testLI=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=relu, Predicted=relu
Actual=tanh, Predicted=relu
Actual=nan, Predicted=nan
Actual=relu, Predicted=relu
Actual=relu, Predicted=relu
Actual=tanh, Predicted=relu
Actual=nan, Predicted=nan
```

Figure 6.151 Predicted values of activation function classifier for layer two

6.4.10.3 Activation function classifier for layer 03

For the third layer configuration of the activation function has done as follows.

```
modelaFunL3p = Sequential()
modelaFunL3p.add(Dense(72, activation='relu', input_dim=72))
modelaFunL3p.add(Dense(36, activation='relu'))
modelaFunL3p.add(Dense(18, activation='relu'))
modelaFunL3p.add(Dense(9, activation='softmax'))
modelaFunL3p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 120
epochs = 500
```

Figure 6.152 Neural network for activation function classifier for layer three

While the traing the loss has converge as follows and the accuracy has behaved as diagram below

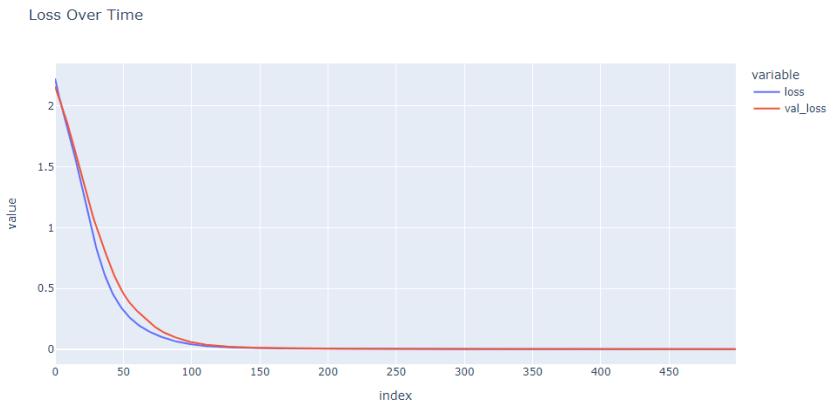


Figure 6.153Lost function for activation function classifier for layer three

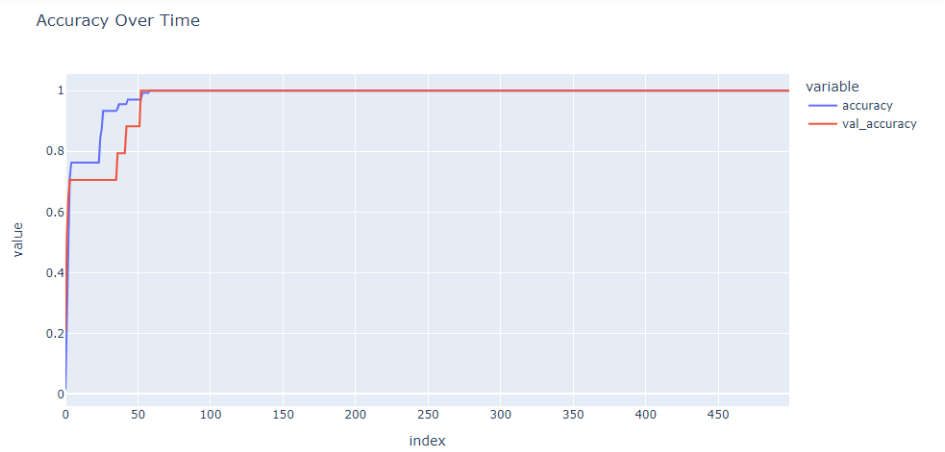


Figure 6.154 Accuracy of activation function classifier for layer three

After the evaluration the lost and the accuracy achived as follows.

```
modelaFunL3p.evaluate(X_test, to_categorical(Y_test,9))
```

```
1/1 [==============================] - 0s 62ms/step - loss: 3.4601 - accuracy: 0.5000
[3.460078477859497, 0.5]
```

```
y_resultLType=modelaFunL3p.predict(X_test)
```

```
1/1 [==============================] - 0s 67ms/step
```

Figure 6.155 Evaluation and prediction of activation function classifier for layer three

Many of the cases actual and classified value is same and for the cases that are not are handling by the model restrictors.

```
aFunPL=[]
#le.fit(unqValues[4])
for i in range(len(X_test)):
    aFun=np.argmax(y_resultLType[i])
    aFunPL.append(aFun)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leGenrl.inverse_transform(aFunPL)
Y_testLI=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=relu, Predicted=nan
Actual=relu, Predicted=relu
Actual=relu, Predicted=relu
Actual=relu, Predicted=nan
Actual=tanh, Predicted=tanh
Actual=relu, Predicted=relu
```

Figure 6.156 Predicted values of activation function classifier for layer three

6.4.10.4 Activation function classifier for layer 04

For the fourth layer configuration of the activation function has done as follows.

```
modelaFunL4p = Sequential()
modelaFunL4p.add(Dense(72, activation='relu', input_dim=72))
modelaFunL4p.add(Dense(36, activation='relu'))
modelaFunL4p.add(Dense(18, activation='relu'))
modelaFunL4p.add(Dense(9, activation='softmax'))
modelaFunL4p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 120
epochs = 500
```

Figure 6.157Neural network for activation function classifier for layer four

While the training the loss has converge as follows and the accuracy has behaved as diagram below

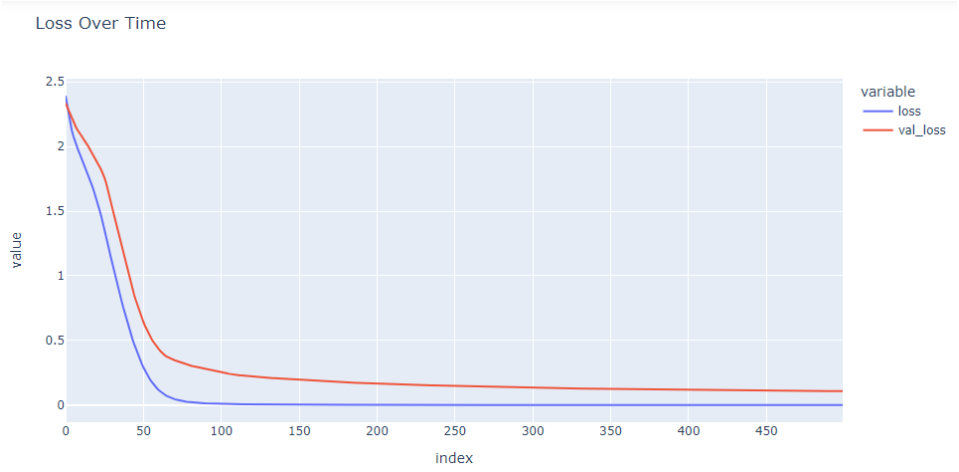


Figure 6.158 Lost function for activation function classifier for layer four

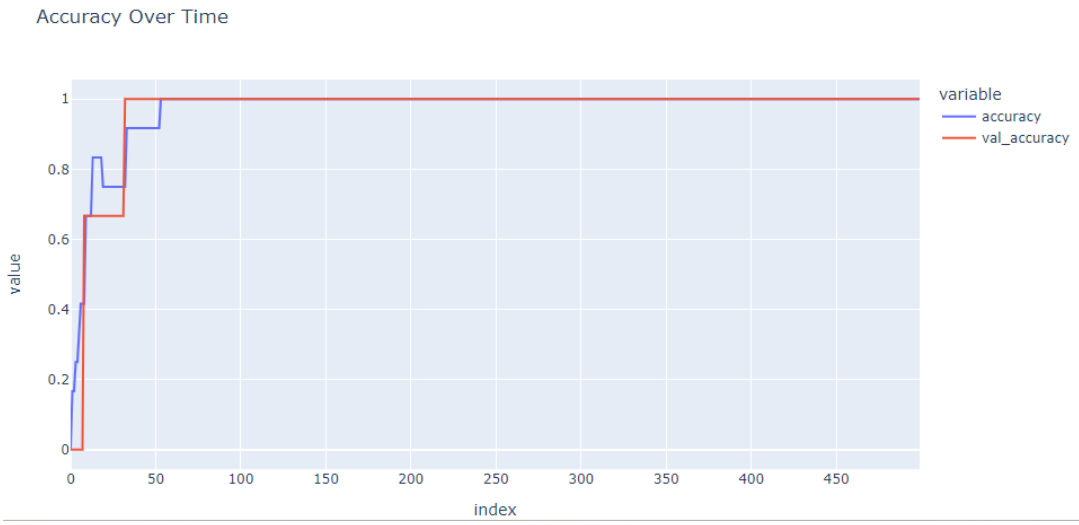


Figure 6.159 Accuracy of activation function classifier for layer four

After the evaluation the lost and the accuracy achieved as follows.

```
modelaFunL4p.evaluate(X_test, to_categorical(Y_test,9))
```

```
1/1 [==============================] - 0s 21ms/step - loss: 1.5628 - accuracy: 0.6000
[1.562762975692749, 0.6000000238418579]
```

```
y_resultLType=modelaFunL4p.predict(X_test)
```

```
1/1 [==============================] - 0s 58ms/step
```

Figure 6.160 Evaluation and prediction of activation function classifier for layer four

Many of the cases actual and classified value is same and for the cases that are not are handling by the model restrictors.

```
aFunPL=[]
#le.fit(unqValues[4])
for i in range(len(X_test)):
    aFun=np.argmax(y_resultLType[i])
    aFunPL.append(aFun)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leGenrl.inverse_transform(aFunPL)
Y_testLI=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=nan, Predicted=relu
Actual=relu, Predicted=nan
Actual=relu, Predicted=relu
Actual=relu, Predicted=relu
Actual=relu, Predicted=relu
```

Figure 6.161Predicted values of activation function classifier for layer four

6.4.10.5 Activation function classifier for layer 05

For the fifth layer configuration of the activation function has done as follows.

```
modelaFunL5p = Sequential()
modelaFunL5p.add(Dense(72, activation='relu', input_dim=72))
modelaFunL5p.add(Dense(36, activation='relu'))
modelaFunL5p.add(Dense(18, activation='relu'))
modelaFunL5p.add(Dense(9, activation='softmax'))
modelaFunL5p.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 120
epochs = 500
```

Figure 6.162 Neural network for activation function classifier for layer five

While the training the loss has converge as follows and the accuracy has behaved as diagram below

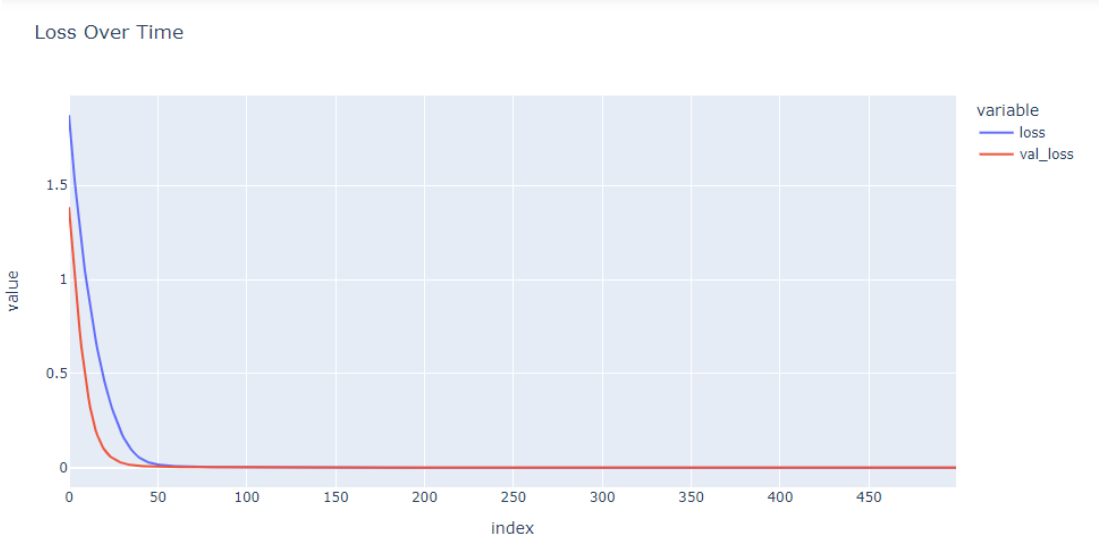


Figure 6.163 Lost function for activation function classifier for layer five

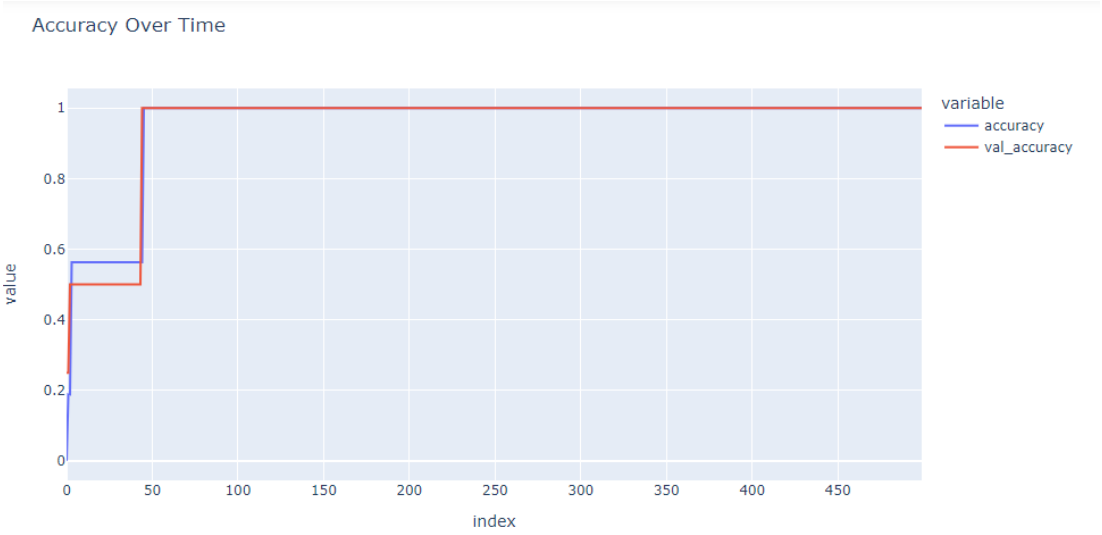


Figure 6.164 Accuracy of activation function classifier for layer five

After the evaluation, the lost and the accuracy achieved as follows.

```
1  modelaFunL5p.evaluate(X_test, to_categorical(Y_test,9))
```

```
1/1 [==============================] - 0s 22ms/step - loss: 1.7200e-04 - accuracy: 1.0000
[0.00017199519788846374, 1.0]
```

```
1  y_resultLType=modelaFunL5p.predict(X_test)
```

```
1/1 [==============================] - 0s 59ms/step
```

Figure 6.165 Evaluation and prediction of activation function classifier for layer five

Many of the cases actual and classified value is same and for the cases that are not are handling by the model restrictors.

```
aFunPL=[]
#le.fit(unqValues[4])
for i in range(len(X_test)):
    aFun=np.argmax(y_resultLType[i])
    aFunPL.append(aFun)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leGenrl.inverse_transform(aFunPL)
Y_testLI=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=nan, Predicted=nan
Actual=tanh, Predicted=tanh
```

Figure 6.166 Predicted values of activation function classifier for layer five

6.4.10.6 Activation function classifier for layer 06

For the final layer configuration of the activation function has done as follows.

```
modelaFunLLp = Sequential()
modelaFunLLp.add(Dense(72, activation='relu', input_dim=72))
modelaFunLLp.add(Dense(36, activation='relu'))
modelaFunLLp.add(Dense(18, activation='relu'))
modelaFunLLp.add(Dense(9, activation='softmax'))
modelaFunLLp.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 120
epochs = 500
```

Figure 6.167 Neural network for activation function classifier for layer six

While the training the loss has converged as follows and the accuracy has behaved as diagram below

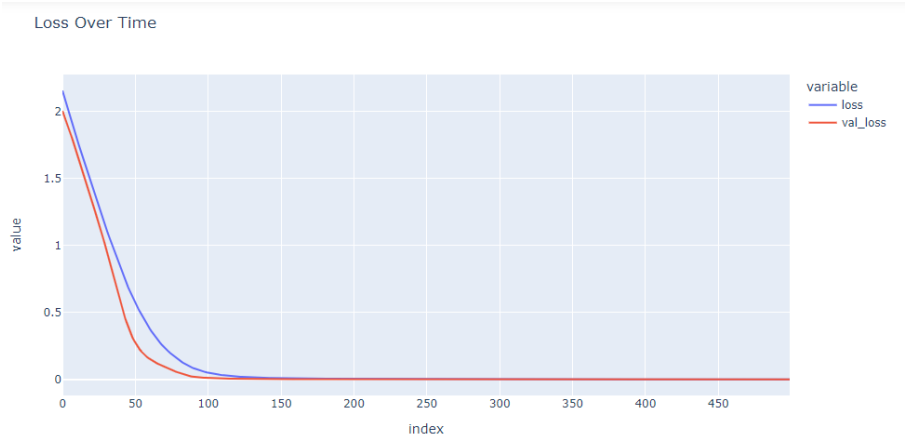


Figure 6.168 Lost function for activation function classifier for layer six

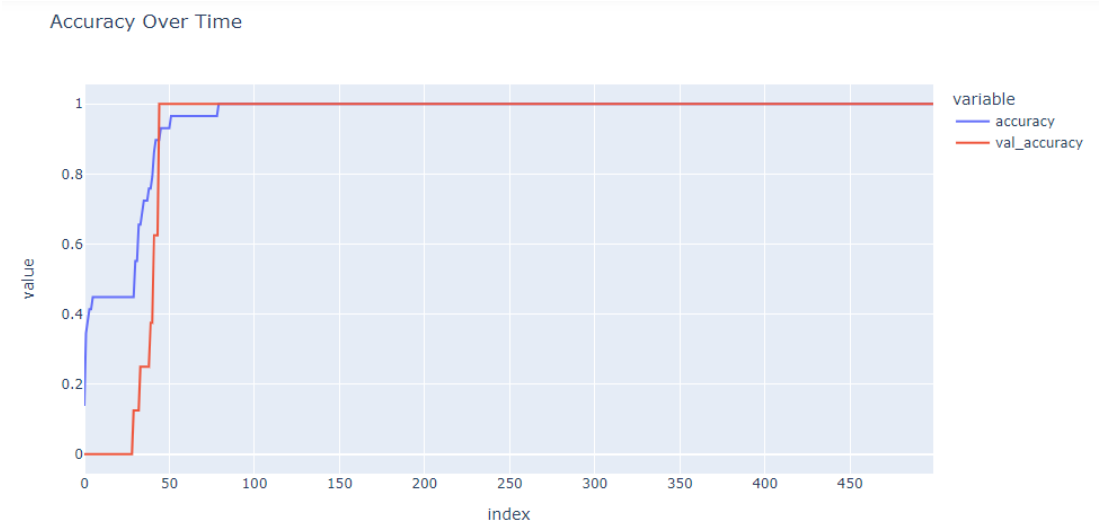


Figure 6.169 Accuracy of activation function classifier for layer six

After the evaluation the lost and the accuracy achieved as follows.

```
1  modelaFunLLp.evaluate(X_test, to_categorical(Y_test,9))
```

```
1/1 [==============================] - 0s 26ms/step - loss: 0.4525 - accuracy: 0.8333
[0.45247575640678406, 0.8333333134651184]
```

```
1  y_resultLType=modelaFunLLp.predict(X_test)
```

```
1/1 [==============================] - 0s 54ms/step
```

Figure 6.170 Evaluation and prediction of activation function classifier for layer six

In many of the cases actual and classified value is the same and for the cases that are not are handling by the model restrictors.

```
aFunPL=[]
#le.fit(unqValues[4])
for i in range(len(X_test)):
    aFun=np.argmax(y_resultLType[i])
    aFunPL.append(aFun)
    #print("Predicted Layer Type for Unseen fram row",i,":",LayType)
y_resultLTypeLI=leGenrl.inverse_transform(aFunPL)
Y_testLI=leGenrl.inverse_transform(Y_test)
for i in range(len(X_test)):

    print("Actual=%s, Predicted=%s" % (Y_testLI[i], y_resultLTypeLI[i]))
```

```
Actual=sigmoid, Predicted=sigmoid
Actual=sigmoid, Predicted=sigmoid
Actual=sigmoid, Predicted=sigmoid
Actual=sigmoid, Predicted=sigmoid
Actual=sigmoid, Predicted=sigmoid
Actual=nan, Predicted=sigmoid
Actual=sigmoid, Predicted=sigmoid
```

Figure 6.171 Predicted values of activation function classifier for layer six

6.4.11 Embedding Layer - Input dimension predictor

The following neural network structure is used to predict the input dimension of the embedding layer.

```
modelembIdPrdfL = Sequential()
modelembIdPrdfL.add(Dense(73, input_shape=(73,), activation='relu'))
modelembIdPrdfL.add(Dense(36, activation='relu'))
modelembIdPrdfL.add(Dense(18, activation='relu'))
modelembIdPrdfL.add(Dense(9, activation='relu'))
modelembIdPrdfL.add(Dense(1))#, activation='linear'
modelembIdPrdfL.compile(loss='mean_squared_logarithmic_error', optimizer='adam',metrics=['mse'])
```

```
batch_size = 200
epochs = 10000
```

Figure 6.172 Neural network for input dimension of the embedding layer

The lost function has converged after training the neural network, as in the following graph.

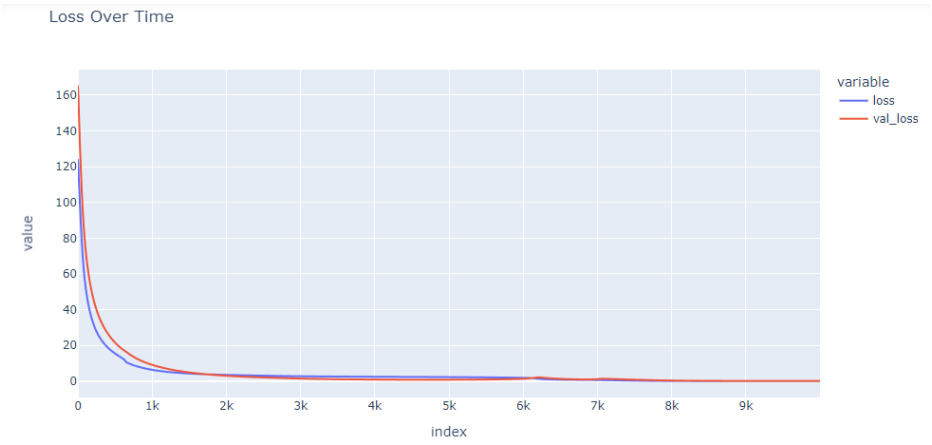


Figure 6.173 Lost function for input dimension of the embedding layer

The treatment results in the loss of 13.2

```
:  1  modelembIdPrdfL.evaluate(X_test,Y_test)

   1/1 [==============================] - 0s 25ms/step - loss: 13.2440

:  [13.244041442871094, 135090331648.0]

:  1  y_result=modelembIdPrdfL.predict(X_test)

   1/1 [==============================] - 0s 60ms/step
```

Figure 6.174 Evaluation and prediction of the input dimension of the embedding layer

6.4.12 Embedding Layer - Output dimension predictor

The following neural network structure is used to predict the output dimension of the embedding layer.

```
1  modelembOdPrdfL = Sequential()
2  modelembOdPrdfL.add(Dense(74, input_shape=(74,), activation='relu'))
3  modelembOdPrdfL.add(Dense(37, activation='relu'))
4  modelembOdPrdfL.add(Dense(18, activation='relu'))
5  modelembOdPrdfL.add(Dense(9, activation='relu'))
6  modelembOdPrdfL.add(Dense(1))#, activation='linear'
7  modelembOdPrdfL.compile(loss='mean_squared_logarithmic_error', optimizer='adam',metrics=['mse'])
```

```
1  batch_size = 200
2  epochs = 10000
```

Figure 6.175 Neural network for output dimension of the embedding layer

The lost function has converged after training the neural network, as in the following graph.

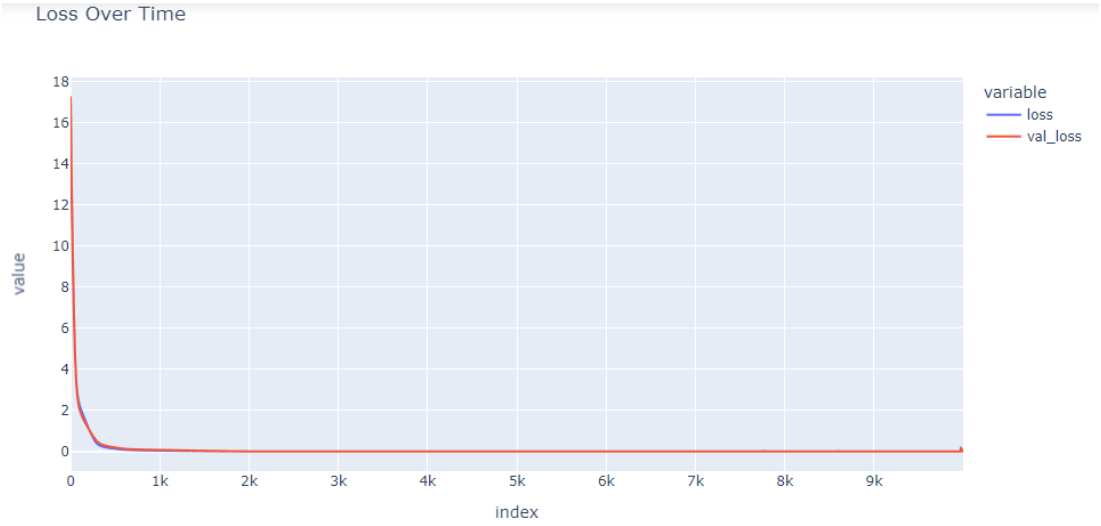


Figure 6.176 Lost function for output dimension of the embedding layer

The treaing results in th lost of 2.9

```
1 modelembOdPrdfL.evaluate(X_test,Y_test)
```

```
1/1 [==============================] - 0s 66ms/step - loss: 2.9506 - mse: 142947.1250
[2.950613260269165, 142947.125]
```

```
1 y_result=modelembOdPrdfL.predict(X_test)
```

```
1/1 [==============================] - 0s 78ms/step
```

```
1 for i in range(len(X_test)):
2
3     print("Actual=%s, Predicted=%s" % (Y_test[i], y_result[i].round(2)[0]))
```

```
Actual=300, Predicted=221.05
Actual=1000, Predicted=121.41
Actual=128, Predicted=239.89
```

Figure 6.177 Evaluation and prediction, prediction values of output dimension of the embedding layer

6.4.13 Final Hyperparameter Predictor Module

The following module is the hyperparameter predictor that has 34 methods withing the Python class

```
hppredictor():
ef __init__(self,genFra=pd.DataFrame,genDictionary={}):
   self.genDictionary=genDictionary
   self.genFra=genFra
   self.totFiles=0
ef modelSelector(self,modelIndex=0,modelPath='PredictorsClassifies/1.Modelselection/'): #/PredictorsClassifies/1.Modelselect

   filelist=os.listdir(modelPath)
   totFiles=len(filelist)
   filename =modelPath+filelist[modelIndex]
   modelMS = keras.models.load_model(filename)
   print("filename:",filename,"Nooffiles:",totFiles)
   return modelMS
ef NoLayersPredictor(self,modelIndex=0,modelPath='PredictorsClassifies/2.NoLayersPrediction/'): #/PredictorsClassifies/1.Mod

   filelist=os.listdir(modelPath)
   totFiles=len(filelist)
   filename =modelPath+filelist[modelIndex]
   modelNLPr = keras.models.load_model(filename)
   print("filename:",filename,"Nooffiles:",totFiles)
   return modelNLPr
ef OptimizerClassifire(self,modelIndex=0,modelPath='PredictorsClassifies/3.OptimizerClassifire/'): #/PredictorsClassifies/1.
```

Figure 6.178 The hyperparameter prediction class

6.5 Module 05: General module

The main module of the general model is the general module. As per the design, according to the predictions, the selection of the ANN or RNN model layers by the inbuilt model templates happens here.

6.5.1 Create a general model

As described in the design section, the following code is used to make predictions for ANN and RNN according to the problem based on the ANN-RNN predictor.

```
def CreateGeneralModel(self,nnType=-1):
    #LocalDf=pd.DataFrame(['cam','pam'])
    self.tempFraANN=pd.DataFrame
    self.tempFraRNN=pd.DataFrame
    if 'nature_of_NN'in self.genFra.columns:
        self.genFra=self.genFra.drop('nature_of_NN',axis=1)
    LocalDf=self.genFra.copy()
    selctVal,skip=self.Predictor.modelSelector(0,self.genFra)
    ind=0

    for itm in selctVal:
        fixV=self.annModelRestrictor("ModSe",[itm,"Nominal"])[0][0]
        selctVal[ind]=fixV
        #print("fixV:",fixV)
        ind+=1
    LocalDf.insert(60,'nature_of_NN',selctVal,True)
    self.genFra.insert(60,'nature_of_NN',selctVal,True)
    #print("type:",type(LocalDf),type(self.genFra))
    #print("ANN:",0 in LocalDf.nature_of_NN.tolist())
    if 0 in LocalDf.nature_of_NN.tolist() and (nnType==-1 or nnType==0):
        annDf=LocalDf[LocalDf.nature_of_NN==0]
        annDf=self.annLayerPredictor(annDf)
        self.tempFraANN=annDf
```

Figure 6.179 Create General Model

First, if the “nature_of_NN” column already exists, the prevent rerunning error will drop. Then, by the condition, it filters the ANN problems predicted by the model selector predictor, as in the above figure.

```
    if 1 in LocalDf.nature_of_NN.tolist() and (nnType==-1 or nnType==1):
        rnnDf=LocalDf[LocalDf.nature_of_NN==1]
        rnnDf=self.rnnLayerPredictor(rnnDf)
        self.tempFraRNN=rnnDf
    if nnType==0:
        return self.tempFraANN,1
    elif nnType==1:
        return 0,self.tempFraRNN
    else:
        return self.tempFraANN,self.tempFraRNN
```

Figure 6.180 RNN part of the creat general model

Filtering happens similarly for RNN problems, as in the above figure.

6.5.2 ANN & RNN Layer Predictors

ANN and RNN layer predictors work as follows.

6.5.2.1 ANN Layer Predictor

In the first several predictors predicted directly by the latest model found as shown in the figure below.

```
def annLayerPredictor(self,annLocalDF=pd.DataFrame):
    print("ANN predict Layers...")
    unpY=[25,96,50]
    noLPra,skip=self.Predictor.NoLayersPredictor(0,annLocalDF)
    indNL=0
    for nL in noLPra:
        fixV=self.annModelRestrictor("NLPr",[nL.round(0),"Nominal"])[0]
        #print("NL:",nL,"fix",fixV)
        noLPra[indNL]=fixV
        indNL+=1
    annLocalDF.insert(61,self.predictionSequance[1],noLPra,True)
    optCla,skip=self.Predictor.OptimizerClassifire(0,annLocalDF)
    optClaValL=[]
    for opt in optCla:
        optimizer=np.argmax(opt)
        optClaValL.append(optimizer)
    annLocalDF.insert(62,self.predictionSequance[2],optClaValL,True)
    lostf,skip=self.Predictor.LostFunctionClassifire(0,annLocalDF)
    lostClaValL=[]
    for lost in lostf:
        optimizer=np.argmax(lost)
        lostClaValL.append(optimizer)
    self.leblFit(1)
    annLocalDF.insert(63,self.predictionSequance[3],lostClaValL,True)
    metrixCla,skip=self.Predictor.MetricsClassifire(0,annLocalDF)
    metrixClaValL=[]
```

Figure 6.181starting part of the ANN layer predictor

Each hyperparameter takes the predicted values to go through the restrictor class to check for the anomalies and directly enter the fixed values without looking for further models to predict, as in the above figure 6.181.

```
LaTypCol=[]
annLocalDFLaTypL1=annLocalDF[annLocalDF.layer_no==0]
run=False
indLaTyp1=0
modelS=0
rounds=0
LaTyp=0
LaTypClaL1=[]
LaTypClaValL1=[]
skip=False
while(not(run)):
    LaTypClaL1,skip=self.Predictor.LayerTypeClassifireL1(modelS,annLocalDFLaTypL1,skip)
    modelS+=1


    print("State1:",run)
    while LaTyp <len(LaTypClaL1):
        lLaTyp=np.argmax(LaTypClaL1[LaTyp])
        #print("Layer1Items:",len(dropPreL1),":",inddrop1,"nNClaLLitems:",len(nNClaLL),":",LnN)
        run=self.annModelRestrictor("LaTyp",[lLaTyp,"Nominal"],0,0,LaTyp)[1]
        #run=True
        #print("currentLV:",LnN,"previousLV",self.curPreValueMapper(nNCol,nNClaLL,indNN1),"run:",run)
        if rounds>5:
            run=True
            lLaTyp=self.annModelRestrictor("LaTyp",[lLaTyp,"Nominal"],0,0,LaTyp)[0]
            #indNN1-=1
        elif not(run) :
            break
        if run:
            modelS=0
            rounds=0
        LaTypClaValL1.append(lLaTyp)

        indLaTyp1+=1
        LaTyp+=1
        #print("State:",run)
    rounds+=1
LaTypCol.append(LaTypClaValL1)
```

Figure 6.182code for loading different predictors when anomalies happen in predicted values in the latest model

When we consider the layer-wise predictions in addition to the previous process, the code will try to acquire the predictions of other available predictors if there are anomalies before. As in the above figure, the layer type of layer one is achieved by two nested while loops.

6.5.2.2 RNN Layer Predictor

Like the ANN Layer predictor, the RNN layer predictor also predicts the initial hyperparameters with the latest predictor with model restrictor for assigned values if abnormal values are predicted, as in the following figure.

```
def rnnLayerPredictor(self,rnnLocalDF=pd.DataFrame):
    print("RNN create Layers...")
    unpY=[25,96,50]
    noLPra,skip=self.Predictor.NoLayersPredictor(0,rnnLocalDF)
    indNL=0
    for nL in noLPra:
        fixV=self.annModelRestrictor("NLPr",[nL.round(0),"Nominal"])[0]
        #print("NL:",nL,"fix",fixV)
        noLPra[indNL]=fixV
        indNL+=1
    rnnLocalDF.insert(61,self.predictionSequance[1],noLPra,True)
    optCla,skip=self.Predictor.OptimizerClassifire(0,rnnLocalDF)
    optClaVall=[]
    for opt in optCla:
        optimizer=np.argmax(opt)
        optClaVall.append(optimizer)
    rnnLocalDF.insert(62,self.predictionSequance[2],optClaVall,True)
    lostf,skip=self.Predictor.LostFunctionClassifire(0,rnnLocalDF)
    lostClaVall=[]
    for lost in lostf:
        optimizer=np.argmax(lost)
        lostClaVall.append(optimizer)
    self.leblFit(1)
    rnnLocalDF.insert(63,self.predictionSequance[3],lostClaVall,True)
    metrixCla,skip=self.Predictor.MetricsClassifire(0,rnnLocalDF)
    metrixClaVall=[]
    for met in metrixCla:
        metrix=np.argmax(met)
        metrixClaVall.append(metrix)
    self.leblFit(2)
    rnnLocalDF.insert(64,self.predictionSequance[4],metrixClaVall,True)
    nEPr,skip=self.Predictor.NoEpochesPredictior(0,rnnLocalDF)
    indNE=0
    #epPrL=[]
    for nE in nEPr:
        #fixV=self.annModelRestrictor("NLPr",[nE.round(0),"Nominal"])[0]
        #print("NL:",nE,"fix",fixV)
        nEPr[indNE]=nE.round(0)
        indNE+=1
    rnnLocalDF.insert(65,self.predictionSequance[5],nEPr,True)
    #LblResult=self.leGenrL.inverse_transform(metrixClaVall)
    bSPr,skip=self.Predictor.BatchSizePredictor(0,rnnLocalDF)
    indBS=0
```

Figure 6.183 Starting code of the RNN layer predictor

In the same way, as in ANN, RNN layer-wise prediction will look for other available predictors that are trained to predict the same hyperparameter before handling abnormal values by assigning fix values as following figure.

```
LaTypCol=[]
rnnLocalDFLaTypL1=rnnLocalDF[rnnLocalDF.layer_no==0]
run=False
indLaTyp1=0
modelS=0
rounds=0
LaTyp=0
LaTypClaL1=[]
LaTypClaValL1=[]
skip=False
while(not(run)):
    LaTypClaL1,skip=self.Predictor.LayerTypeClassifireL1(modelS,rnnLocalDFLaTypL1,skip)
    modelS+=1


    print("State1:",run)
    while LaTyp <len(LaTypClaL1):
        lLaTyp=np.argmax(LaTypClaL1[LaTyp])
        #print("Layer1Items:",len(dropPreL1),":",inddrop1,"nNCLaLLitems:",len(nNCLaLL),":",lnN)
        run=self.rnnModelRestrictor("LaTyp",[lLaTyp,"Nominal"],0,LaTyp)[1]
        #run=True
        #print("currentLV:",lnN,"previousLV",self.curPreValueMapper(nNCol,nNCLaLL,indNN1),"run:",run)
        if rounds>5:
            run=True
            lLaTyp=self.rnnModelRestrictor("LaTyp",[lLaTyp,"Nominal"],0)[0]
            #indNN1-=1
        elif not(run) :
            break
        if run:
            modelS=0
            rounds=0
        LaTypClaValL1.append(lLaTyp)

        indLaTyp1+=1
        LaTyp+=1
        #print("State:",run)
    rounds+=1
LaTypCol.append(LaTypClaValL1)
self.temp.append(LaTypClaValL1)
```

Figure 6.184 RNN load other predictor when anomalies in the prictor before assign fix value

6.5.3 ANN & RNN layer restrictors

Layer restrictors use for handle anomalies of prediction. For example, for ANN it can't have LSTM layers.

6.5.3.1 ANN layer restrictors

In ANN layer restrictors it will first detect the anomaly by the rule and then "predictionAcc" Boolean variable is False it will continually predict by taking one by one predicter that has trained to predict a specific hyperparameter until all the list ends. If the solution for anomaly is found before the end of the list because no rule accepts the prediction the value of "predictionAcc" will be keep True resulting in sending the predicted hyperparameters. Otherwise, it reached end then fix value will assign and keep the "predictionAcc" value false allowing make the value of "predictionAcc" keep true in next iteration because of no rule is accepted so anomaly solved and hyperparameter sending happen to the ANN general dataframe.

However, if when the current prediction happened it could also find according to the anomalies of previously predicted hyperparameters such as activation function result

"relu" but the previous type can be "input" crates anomaly so it can be select current value is wrong checking the previous layer type value for the layer 1. The current rules for restriction are indicated in the following codes as in the figure below.

```
def annModelRestrictor(self,hppNme="",hppVal=[-1,"Nominal"],LayerNo=0,hppValcmp=0,hppValcmp2=0):

    predictionAcc=True
    fixVal=-1
    fixNme=""
    if hppNme=="ModSe":
        out = hppVal[0] + 0.5
        np.floor(out, out=out)
        out=out.astype(int)
        fixVal=out.ravel()

    if hppNme=="NLPr":
        if hppVal[0]<2:
            fixVal=2
            predictionAcc=False
        elif hppVal[0]>6:
            fixVal=6
            predictionAcc=False
        else:
            fixVal=hppVal[0]

    if hppNme=="nN" and hppValcmp<hppVal[0] and hppValcmp>0:
        fixVal=hppValcmp
        predictionAcc=False
        #print("hppValcmp:",hppValcmp,"hppVal",hppVal[0])
    if hppNme=="nN" and LayerNo==0 and hppVal[0]<3:
        print("layer0 Rest Check",hppVal[0])
        #Layer1lst=self.tempFraANN[self.tempFraANN.layer_no==0].index.tolist()
        if self.tempFraANN.loc[Layer1lst[hppValcmp2]][68] in [1]:
            fixVal=5
            predictionAcc=False
    if  hppNme=="drop":
        if hppVal[0]<0 or hppVal[0]== -0:
            fixVal=0
            predictionAcc=False
            #print('fixed...')
        else:
            fixVal=hppVal[0]
```

*Figure 6.185ANN model restrictor,layer restericto*r

6.5.3.2 RNN layer restrictors

Previously explain concepts are same for RNN and in addition to that the previously predicted neural network type can be LSTM but if the current activation function is null then ither current activation function can be change in to "tanh" or previous layer type can be change to "input" type. Some rules are written in such a way that previously predicted hyperparameter to change can be identified as backward prediction. So, such rules can be identified by the following figure.

```
def rnnModelRestrictor(self,hppNme="",hppVal=[-1,"Nominal"],LayerNo=0,hppValcmp=0,hppValcmp2=0):

    predictionAcc=True
    fixVal=-1
    fixNme=""

    if hppNme=="NLPr":
        if hppVal[0]<2:
            fixVal=2
            predictionAcc=False
        elif hppVal[0]>6:
            fixVal=6
            predictionAcc=False
        else:
            fixVal=hppVal[0]

    if hppNme=="nN" :


        if LayerNo==0 and hppVal[0]==0:
            Layer1lst=self.tempFraRNN[self.tempFraRNN.layer_no==0].index.tolist()
            self.tempFraRNN.loc[Layer1lst[hppValcmp2],'layer_type']=4
            fixVal=hppVal[0]
            predictionAcc=True
        elif LayerNo==0 and hppVal[0]!=0:
            Layer1lst=self.tempFraRNN[self.tempFraRNN.layer_no==0].index.tolist()
            if self.tempFraRNN.loc[Layer1lst[hppValcmp2]][68] in [2,4]:
                fixVal=0
                predictionAcc=False
        elif LayerNo==1 and hppVal[0]!=0:
            Layer1lst=self.tempFraRNN[self.tempFraRNN.layer_no==1].index.tolist()
            if self.tempFraRNN.loc[Layer1lst[hppValcmp2]][68] in [2]:
                fixVal=0
                predictionAcc=False
                #print("emb Detected..")
            elif hppValcmp<hppVal[0] and LayerNo!=0 and hppValcmp!=0:

                fixVal=hppValcmp
                predictionAcc=False
                self.preValSt=False
                #print("cmpAssogened...")
            elif hppVal[0]<(hppValcmp/4):
                fixVal=(hppValcmp/4)
                predictionAcc=False
                self.preValSt=False
```

Figure 6.186 RNN model restrictor, layer resterctor

6.5.4 ANN & RNN layer creators

The TensorFlow layers can't be generated by splitting the part of the layers. So, the TensorFlow layers has created with the variables for the parameters for TensorFlow as following figures. Moreover, the layers have been arranged into two templates. One is sequential model and other one is non sequential input model. The following figure shows the ANN layer creator class.

```
def annLayerCreator(self,mdelTL="BC",LayerNo=0,LayerTyp='Dense',inputSh=5,nNeur=20,drop=0,afun='relu',inputswitch=Fals

    #print("mdelTL:",mdelTL)
    if LayerNo==0 and LayerTyp=='Input'or inputswitch:
        #print("Input")
        inputswitch=True

        if LayerNo==0:
            self.inputs = tf.keras.Input(shape=(inputSh,))
            print("inputs = tf.keras.Input(shape=(",inputSh,",))")
        elif LayerNo==1:
            if drop==0 and LayerTyp=='Dense':
                connector = tf.keras.layers.Dense(nNeur, activation=afun)(self.inputs)
                print("connector = tf.keras.layers.Dense(",nNeur,", activation='",afun,"')(inputs)")
            elif drop!=0  and LayerTyp=='Dense':
                connector = tf.keras.layers.Dense(nNeur, activation=afun)(self.inputs)
                connector = tf.keras.layers.Dropout(drop)(connector)
                print("connector = tf.keras.layers.Dense(",nNeur,", activation='",afun,"')(inputs) \n connector = tf.k

        elif LayerNo>1 and LayerNo<100:
            if drop==0 and LayerTyp=='Dense':
                connector = tf.keras.layers.Dense(nNeur, activation=afun)(connector)
                print("connector = tf.keras.layers.Dense(",nNeur,", activation='",afun,"')(connector)")
            elif drop!=0  and LayerTyp=='Dense':
                connector = tf.keras.layers.Dense(nNeur, activation=afun)(connector)
                connector = tf.keras.layers.Dropout(drop)(connector)
                print("connector = tf.keras.layers.Dense(",nNeur,", activation='",afun,"')(connector) \n connector = t
        elif LayerNo==100 and mdelTL=="BC":
            outputs = tf.keras.layers.Dense(1, activation=afun)(connector)
            model = tf.keras.Model(inputs, outputs)
            inputswitch=False
            print("outputs = tf.keras.layers.Dense(",1,", activation='",afun,"')(connector) \n model = tf.keras.Model(
        elif LayerNo==100 and mdelTL=="MC":
            outputs = tf.keras.layers.Dense(nNeur, activation='softmax')(connector)
            model = tf.keras.Model(inputs, outputs)
            inputswitch=False
            print("outputs = tf.keras.layers.Dense(",nNeur,", activation='softmax')(connector) \n model = tf.keras.Mod
        elif LayerNo==100 and mdelTL=="Re":
            outputs = tf.keras.layers.Dense(1)(connector)
            model = tf.keras.Model(inputs, outputs)
            inputswitch=False
            print("outputs = tf.keras.layers.Dense(1)(connector) \n model = tf.keras.Model(inputs, outputs)")
```

Figure 6.187ANN layer creator method

Further the running code layers had indicated to the user by using the print. Everything is same for the RNN layer creator except there are more layer types and arrangements due to the LSTM and embedding layers in addition to the input and dense layers found in the ANN layer creator as following figure.

```
def rnnLayerCreator(self,mdelTL="BC",LayerNo=0,LayerTyp='Dense',inputSh=tuple,nNeur=20,drop=0,afun='relu',embInputDi=1

    if LayerNo==0 and LayerTyp=='Input'or inputswitch:
        #print("Input")
        inputswitch=True

        if LayerNo==0:
            self.inputs = tf.keras.Input(shape=inputSh)#(inputSh,1,)
            print("inputs = tf.keras.Input(shape=(",inputSh,",1))")
        elif LayerNo==1:
            if drop==0 and LayerTyp=='LSTM':
                connector = tf.keras.layers.LSTM(nNeur, activation=afun,return_sequences=True)(self.inputs)
                print("connector = tf.keras.layers.LSTM(",nNeur,", activation='",afun,"')(inputs)")
            elif drop!=0  and LayerTyp=='LSTM':
                connector = tf.keras.layers.LSTM(nNeur, activation=afun,return_sequences=True)(self.inputs)
                connector = tf.keras.layers.Dropout(drop)(connector)
                print("connector = tf.keras.layers.LSTM(",nNeur,", activation='",afun,"')(inputs) \n connector = tf.ke

            if drop==0 and LayerTyp=='Embedding':
                connector = tf.keras.layers.Embedding(embInputDi, embOutputDi)(self.inputs)
                print("connector = tf.keras.layers.Embedding(",embInputDi,",",embOutputDi,")(inputs)")
            elif drop!=0  and LayerTyp=='Embedding':
                connector = tf.keras.layers.Embedding(embInputDi, embOutputDi)(self.inputs)
                connector = tf.keras.layers.Dropout(drop)(connector)
                print("connector = tf.keras.layers.Embedding(",embInputDi,",",embOutputDi,")(inputs) \n connector = tf.
        elif LayerNo==2:
            if drop==0 and LayerTyp=='LSTM':
                connector = tf.keras.layers.LSTM(nNeur, activation=afun,return_sequences=True)(connector)
                print("connector = tf.keras.layers.LSTM(",nNeur,", activation='",afun,"')(connector)")
            elif drop!=0  and LayerTyp=='LSTM':
                connector = tf.keras.layers.LSTM(nNeur, activation=afun,return_sequences=True)(connector)
                connector = tf.keras.layers.Dropout(drop)(connector)
                print("connector = tf.keras.layers.LSTM(",nNeur,", activation='",afun,"')(connector) \n connector = tf.
            if drop==0 and LayerTyp=='Bi-LSTM':
                connector = tf.keras.layers.Bidirectional(Layer.LSTM(nNeur, return_sequences = True, activation=afun))(
                print("connector = tf.keras.layers.Bidirectional(Layer.LSTM(",nNeur,",return_sequences = True, activat
            elif drop!=0  and LayerTyp=='Bi-LSTM':
                connector = tf.keras.layers.Bidirectional(Layer.LSTM(nNeur, return_sequences = True, activation=afun))(
                connector = tf.keras.layers.Dropout(drop)(connector)
                print("connector = tf.keras.layers.Bidirectional(Layer.LSTM(",nNeur,", activation='",afun,"'))(connecte

            if drop==0 and LayerTyp=='Dense':
                connector = tf.keras.layers.Dense(nNeur, activation=afun)(connector)
```

Figure 6.188 RNN layer creator method

6.5.5 Configuration loader

The first coding is allocated to identify the problem as regression problem or multiclass or binary classification as following code.

```
def configurationLoader(self,problem=0,indpX=np.ndarray,depY=np.ndarray,unpY=pd.DataFrame,unpX=pd.DataFrame):
    self.indpX=indpX
    self.depY=depY
    self.unpY=unpY
    self.unpX=unpX
    tf.keras.backend.clear_session()
    unprItemTypeList=self.unpY.dtypes
    uniqueCount=self.unpY
    if str(uniqueCount)[20:25] !="frame":
        uniqueCount=len(self.unpY.unique())
        self.dtSetCon=True

    else:
        uniqueCount=0
    #print("unprItemTypeList:",str(uniqueCount)[20:25])
    if unprItemTypeList is float or str(unprItemTypeList)[:5] == str(np.float64)[14:19] or str(unprItemTypeList)[:3] =
        #print("Number Process...!!!")

        if uniqueCount<=2:
            self.mdelTL="BC"
        elif uniqueCount<=15:
            self.mdelTL="MC"
        else:
            self.mdelTL="Re"
    else:
        if uniqueCount<=2:
            self.mdelTL="BC"
        elif uniqueCount<=15:
```

Figure 6.189 Starting of configuration loader method

Then according to the ANN-RNN model selection result the ANN or RNN code will be run. According to the ANN code section, the restrictions according to the characteristics of the dataset is written in this code by extracting each layer related to each problem as in following figure.

```
        annModel= self.model
        con=tf.keras
        problemIndexList=annDfHp[annDfHp.layer_no==100].index.tolist()
        #print("problemList",problemIndexList)
        if len(problemIndexList)-1>(problem-self.annInVCount-1) and problem-self.annInVCount!=0:
            row=problemIndexList[problem-self.annInVCount -1]+1
        inputswitch=False
        #print('row:',row,"self.annInVCount:",self.annInVCount)
        while(run):
            #print('rowin loop:',row)
            hppList=annDfHp.loc[row].to_list()
            #print("dfV:",hppList,"indexV:",row)
            myList1=[hppList[1]]
            hppList[1]=self.leLType.inverse_transform([int(myList1) for myList1 in myList1])
            #print("hppList[1]:",hppList[1])
            self.leblFit(3)
            myList2=[hppList[5]]
            hppList[5]=self.leGenrl.inverse_transform([int(myList2) for myList2 in myList2])
            #print("hppList[5]:",hppList[5])
            #print("tyComp:",hppList[2])
            if int(hppList[0])==0 and str(type(self.indpX))[8:12]!="type":
                if type(hppList[0]) is float:
                    hppList[2]=int(self.indpX.shape[1])
                    #print("flType:",hppList[2])
                else:
                    hppList[2]=self.indpX.shape[1]
                    #print("noneflType:",hppList[2])
            else:
                if type(hppList[0]) is float:
                    hppList[2]=int(hppList[2])
                    #print("flType:",hppList[2])
                else:
                    hppList[2]=hppList[2]
                    #print("noneflType:",hppList[2])
            #print("input fixed:", hppList[2])
            if self.mdelTL=="MC" and hppList[0]==100:
                hppList[3]=float(self.unqC)

            annModel,con,inputswitch=self.annLayerCreator(self.mdelTL,int(hppList[0]),hppList[1][0],hppList[2],int(hppl
            if annDfHp.loc[row][0]==100:
                #print("stop..")
                run=False
            row+=1
        self.model=annModel
        return annModel
```

Figure 6.190 ANN code part of the configuration loader method, at the end of the code part

This will include the detection of whether the dataset is connected or not for applying the restrictions related to dataset for each problem. For example, in the multiclass classification the number of neurons in the final layer should match with the number of classes in the dependent variable column. Then by a single code line it call the layer creator in two ways according to sequence model and according to the input layer model.

When considering the RNN code that also structure as the same as ANN code section. The following figure shows the restriction section for RNN which is more complex because more neural network layer types are involve in.

```
elif modelList[problem]==1:
    #self.annInVCount+=1
    rnnDf=self.tempFraRNN
    rnnDfHp=rnnDf.iloc[:, 67:75]
    row=0
    run=True
    dropShft=0
    rnnModel= self.model
    con=tf.keras
    problemIndexList=rnnDfHp[rnnDfHp.layer_no==100].index.tolist()
    #print("problemList",problemIndexList)
    if len(problemIndexList)-1>(problem-self.rnnInVCount-1) and problem-self.rnnInVCount!=0:
        row=problemIndexList[problem-self.rnnInVCount -1]+1
    inputswitch=False
    #print('row:',row,"self.annInVCount:",self.rnnInVCount)
    while(run):
        #print('rowin loop:',row)
        hppList=rnnDfHp.loc[row].to_list()


        if row>0 and hppList[0]!=100:
            hppListN=rnnDfHp.loc[row+1].to_list()
            if hppListN[1]==3:
                dropShft=hppList[4]
                #print("dropShft:",dropShft)
                hppList[4]=0
            #hppListP=rnnDfHp.loc[row-1].to_list()
            if hppList[1]==3:
                hppList[4]=dropShft
                #print("as-dropShft",dropShft)
        if hppList[1]==2 and str(type(self.indpX))[8:12]!="type":
            token = Tokenizer()
            token.fit_on_texts(self.unpX)
            hppList[6]=float(len(token.word_index)+1)
        myList1=[hppList[1]]
        hppList[1]=self.leLType.inverse_transform([int(myList1) for myList1 in myList1])
        #print("hppList[1]:",hppList[1])
        self.leblFit(3)
        myList2=[hppList[5]]
        hppList[5]=self.leGenrl.inverse_transform([int(myList2) for myList2 in myList2])
        #print("hppList[5]:",hppList[5])
        #print("tyComp:",hppList[2])
        if int(hppList[0])==0 and str(type(self.indpX))[8:12]!="type":
            if type(hppList[0]) is float:
```

Figure 6.191 RNN code part of the configuration loader that acquires parameters, including embedding layers

As in the above figure 6.191, it acquires the restrictions such as input dimension of the embedding layer by Tokenizer class with fit_on_text method. Also, like ANN code, RNN code consists of different input shape methods according to the data set, as shown in the following figure.

```
            if int(hppList[0])==0 and str(type(self.indpX))[8:12]!="type":
                if type(hppList[0]) is float:
                    hppList[2]=int(self.indpX.shape[1])
                    #print("flType:",hppList[2])
                else:
                    hppList[2]=self.indpX.shape[1]
                    #print("noneflType:",hppList[2])
            else:
                if type(hppList[0]) is float:
                    hppList[2]=int(hppList[2])
                    #print("flType:",hppList[2])
                else:
                    hppList[2]=hppList[2]
                    #print("noneflType:",hppList[2])
            #print("input fixed:", hppList[2])
            if self.mdelTL=="MC" and hppList[0]==100:
                hppList[3]=float(self.unqC)
            if hppList[0]==0:
                hppListN=rnnDfHp.loc[row+1].to_list()
                if hppListN[1]==2 or hppList[1]==2:
                    inputSh=(hppList[2],)

                else:
                    inputSh=(hppList[2],1)
                #print("input shape:",inputSh)
            #print("dfV:",hppList,"indexV:",row)
            rnnModel,con,inputswitch=self.rnnLayerCreator(self.mdelTL,int(hppList[0]),hppList[1][0],inputSh,int(hppLis
            if rnnDfHp.loc[row][0]==100:
                #print("stop..")
                run=False
            row+=1
        self.model=rnnModel#"""
        return rnnModel
```

Figure 6.192 RNN code part of the configuration loader including different input shapes for different situations

The first input shape decision is the input shape when the dataset is connected and disconnected. The second situation is the input shape, based on the layer type, such as embedding layers. The configuration loader class loads the neural network structures according to the selected problem

6.5.6 Model runner

The model runner is the class that trains the model using the predicted hyperparameters. It also consists of ANN and RNN code that works according to the nature of the problem from ANN and RNN. In the ANN code will consist with restrictions for handling training anomalies in the hyperparameters such as performance optimizer in the following figure.

```
    optLocalDF=self.tempFraANN[self.tempFraANN.columns[:62]].loc[row]# .loc[row]
    #print(optLocalDF)
    run=False
    indOpt=0
    modelS=0
    rounds=0
    opt=0
    optCla=[]
    optClaVal=[]
    skip=False
    firstRun=True
    while(not(run)):
        if firstRun:
            optCla1=hppList[0]
            firstRun=False
            if self.annModelRestrictor("opt",[optCla1,"Nominal"],0,opt)[1]:
                break
        else:
            optCla,skip=self.Predictor.OptimizerClassifire(modelS,optLocalDF,skip)
            modelS+=1
        print("State1:",run)
        while opt <len(optCla):
            lopt=np.argmax(optCla[opt])
            #print("layer1Items:",len(dropPreL1),":",inddrop1,"nNClaLLitems:",len(nNClaLL),":",lnN)
            run=self.annModelRestrictor("opt",[lopt,"Nominal"],0,opt)[1]
            #run=True
            #print("currentLV:",lnN,"previousLV",self.curPreValueMapper(nNCol,nNClaLL,indNN1),"run:",run)
            if rounds>5:
                run=True
                lopt=self.annModelRestrictor("opt",[lopt,"Nominal"],0)[0]
                #indNN1-=1
            elif not(run) :
                break
            if run:
                modelS=0
```

Figure 6.193ANN part of the model runner consisting of restriction code to lead different models in prediction anomalies

As you can see, the structure is the same as the layer-wise hyperparameter predictor applied to the optimizer, performance matrix, batch size, and number of epochs. The code also shows the applied code by the print method as following diagram.

```
#...predictor reload end
self.leblFit(0)
myList1=[hppList[0]]
hppList[0]=self.leGenrl.inverse_transform([int(myList1) for myList1 in myList1])
self.leblFit(1)
myList2=[hppList[1]]
hppList[1]=self.leGenrl.inverse_transform([int(myList2) for myList2 in myList2])
self.leblFit(2)
myList3=[hppList[2]]
hppList[2]=self.leGenrl.inverse_transform([int(myList3) for myList3 in myList3])
if(hppList[2][0]=="auc"):
    hppList[2]=keras.metrics.AUC()
else:
    hppList[2]=hppList[2][0]
self.model.compile(optimizer=hppList[0][0], loss=hppList[1][0], metrics=[hppList[2]])
print("model.compile(optimizer=",hppList[0][0],", loss=",hppList[1][0],", metrics=[",hppList[2],"])")
if not(direct):

    if self.mdelTL=="MC":
        print("model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=",int(hppLi
        history=self.model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=int(h
    else:
        print("model.fit(self.indpX,self.depY,validation_split=0.2,epochs=",int(hppList[3]),",batch_size=",int(
        history=self.model.fit(self.indpX,self.depY,validation_split=0.2,epochs=int(hppList[3]),batch_size=int(
else:
    if self.mdelTL=="MC":
        print("model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=",sEpoch,",
        history=self.model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=sEpoc
    else:
        print("model.fit(self.indpX,self.depY,validation_split=0.2,epochs=",sEpoch,",batch_size=",sbatchZ,",ver
        history=self.model.fit(self.indpX,self.depY,validation_split=0.2,epochs=sEpoch,batch_size=sbatchZ,verbo

return history
```

Figure 6.194 Beside the numerical values converted to nominal values, print is used to give the code output for the user in the ANN part

For the RNN code in the model runner is also structured as ANN code as in following diagrams.

```
elif modelList[problem]==1:
    rnnDf=self.tempFraRNN
    rnnDfHp=rnnDf.iloc[:, 62:68]
    row=0
    run=True
    mdelTL=self.mdelTL
    rnnModel= self.model
    problemIndexList=rnnDfHp[rnnDfHp.layer_no==100].index.tolist()
    #print("problemList",problemIndexList)
    if len(problemIndexList)-1>(problem-self.rnnInVCount-1) and problem-self.rnnInVCount!=0:
        row=problemIndexList[problem-self.rnnInVCount -1]+1
    inputswitch=False
    #print('row:',row,"self.rnnInVCount:",self.rnnInVCount)
    hppList=rnnDfHp.loc[row].to_list()
    #...predictor reload start

    optLocalDF=self.tempFraRNN[self.tempFraRNN.columns[:62]].loc[row]# .loc[row]
    #print(optLocalDF)
    run=False
    indOpt=0
    modelS=0
    rounds=0
    opt=0
    optCla=[]
    optClaVal=[]
    skip=False
    firstRun=True
    while(not(run)):
        if firstRun:
            optCla1=hppList[0]
            firstRun=False
            if self.rnnModelRestrictor("opt",[optCla1,"Nominal"],0,opt)[1]:
                break
        else:
            optCla,skip=self.Predictor.OptimizerClassifire(modelS,optLocalDF,skip)
            modelS+=1
        print("State1:",run)
        while opt <len(optCla):
            lopt=np.argmax(optCla[opt])
            #print("Layer1Items:",len(dropPrel1),":",inddrop1,"nNClaLLitems:",len(nNClaLL),":",lnN)
            run=self.rnnModelRestrictor("opt",[lopt,"Nominal"],0,opt)[1]
```

Figure 6.195Similer RNN restrictors usage in the RNN part of the model loader

```
        #print("State:",run)
    rounds+=1
#...predictor reload end"""
self.lblFit(0)
myList1=[hppList[0]]
hppList[0]=self.leGenrl.inverse_transform([int(myList1) for myList1 in myList1])
self.lblFit(1)
myList2=[hppList[1]]
hppList[1]=self.leGenrl.inverse_transform([int(myList2) for myList2 in myList2])
self.lblFit(2)
myList3=[hppList[2]]
hppList[2]=self.leGenrl.inverse_transform([int(myList3) for myList3 in myList3])
if(hppList[2][0]=="auc"):
    hppList[2]=keras.metrics.AUC()
else:
    hppList[2]=hppList[2][0]
self.model.compile(optimizer=hppList[0][0], loss=hppList[1][0], metrics=[hppList[2]])
print("model.compile(optimizer=",hppList[0][0],", loss=",hppList[1][0],", metrics=[",hppList[2],"])")
if not(direct):

    if self.mdelTL=="MC":
        print("model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=",int(hppL
        history=self.model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=int(
    else:
        print("model.fit(self.indpX,self.depY,validation_split=0.2,epochs=",int(hppList[3]),",batch_size=",int
        history=self.model.fit(self.indpX,self.depY,validation_split=0.2,epochs=int(hppList[3]),batch_size=int
else:
    if self.mdelTL=="MC":
        print("model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=",sEpoch,"
        history=self.model.fit(self.indpX,to_categorical(self.depY,self.unqC),validation_split=0.2,epochs=sEpo
    else:
        print("model.fit(self.indpX,self.depY,validation_split=0.2,epochs=",sEpoch,",batch_size=",sbatchZ,",ve
        history=self.model.fit(self.indpX,self.depY,validation_split=0.2,epochs=sEpoch,batch_size=sbatchZ,verb

return history
```

Figure 6.196 Simler data conversion and print usage in the RNN coding part of the model runner

6.5.7 Evaluator and Predictor

Evaluator and predictor menthod created sperarly because when predict for multiclass classification it should go through to_categorical method as in following diagram.

```
def evaluate(self,Xind=np.ndarray,Ydep=np.ndarray):
    if self.mdelTL=="MC":
        self.model.evaluate(Xind,to_categorical(Ydep,self.unqC))
    else:
        self.model.evaluate(Xind,Ydep)
def predict(self,Xind=np.ndarray):
    return self.model.predict(Xind)
```

Figure 6.197implimentation of evaluator and predictor methods due to use of to_categorical method

6.5.8 User process for HPPGeneral model

User should first generate the general frames by the code in the following diagram.

```
1 uSdependentList=['is_canceled','rating','hotel','hotel','is_canceledy','hotel','is_canceled']#
```

```
1 hppGen=HPPGeneralModel("unseenData/","BaseFiles/",uSdependentList)
```

```
unSeenGeneralFrameWithIndVars.csv  reading...
1.hotel_booking.csv allready in...

2.tripadvisor_hotel_reviews.csv allready in...

3.hotel_bookings.csv allready in...

4.hotel_bookings3_FinFe.csv allready in...

5.hotel_booking1FinFe.csv allready in...

6.hotel_bookings2.csv allready in...

7.hotel_bookings3RCP.csv allready in...

Data Frame Saved...
dfDrop: 7 dfAppend: 7
depVeriables: ['depV01', 'depV02', 'depV03', 'depV04', 'depV05', 'depV06', 'depV07', 'depV08', 'depV09', 'depV10']
length: 7
idx_first_empty_row: 7
```

Figure 6.198 Generate general frames and apply encoding by the user

If user wanted to rebuild the encoding then use, True after dependent list, if the use wanted to regenerate all the general freames user thould use another True value follows by previous true value.

To predict the hyperparameters for each problem user should use the following code. With separate variable to capture ANN general frame and RNN general frame as in following diagram.

```
1 ann,rnn=hppGen.CreateGeneralModel()
```

```
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches.sav Nooffiles: 7
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001AB77A37A60> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 48ms/step
filename: PredictorsClassifies/7.BatchSizePredictor/finalized_modelnbatchSizes.sav Nooffiles: 4
1/1 [==============================] - 0s 52ms/step
filename: PredictorsClassifies/8.LayerTypeClassifire/8.1LayerTypeClassifireForLayer01/finalized_modelLayTypeL1M5.sav Nooffiles: 6
1/1 [==============================] - 0s 51ms/step
State1: False
filename: PredictorsClassifies/8.LayerTypeClassifire/8.2LayerTypeClassifireForLayer02/finalized_modelLayTypeL2M4.sav Nooffiles: 5
1/1 [==============================] - 0s 70ms/step
State1: False
filename: PredictorsClassifies/8.LayerTypeClassifire/8.2LayerTypeClassifireForLayer02/finalized_modelLayTypeL2M3.sav Nooffiles: 5
```

Figure 6.199 Use of create general model for run the hyperparameter prediction to inserted problems by the user

In the debugging the developers can explicitly restict running ANN part by giving 1 and restict running ANN part by giving 0 to as paramerter to CreateGeneralModel() method.

After predict output frames can be seen as following diagrams.

```
1 ann
```

	IndV01	IndV02	IndV03	IndV04	IndV05	IndV06	IndV07	IndV08	IndV09	IndV10	...	lost_fun	metrics	no_of_epochs	batch_size	layer_no
0	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	642.0	23.0	0
1	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	642.0	23.0	1
2	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	642.0	23.0	2
3	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	642.0	23.0	3
4	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	642.0	23.0	4
5	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	642.0	23.0	100
6	0.00000	0.32585	0.254769	0.229453	0.213853	0.131590	0.145256	0.352067	0.335369	0.404670	...	1	0	391.0	2409.0	0
7	0.00000	0.32585	0.254769	0.229453	0.213853	0.131590	0.145256	0.352067	0.335369	0.404670	...	1	0	391.0	2409.0	100
8	0.00000	0.32585	0.254769	0.229453	0.213853	0.000000	0.145256	0.352067	0.335369	0.404670	...	1	0	7.0	1837.0	0
9	0.00000	0.32585	0.254769	0.229453	0.213853	0.000000	0.145256	0.352067	0.335369	0.404670	...	1	0	7.0	1837.0	1
10	0.00000	0.32585	0.254769	0.229453	0.213853	0.000000	0.145256	0.352067	0.335369	0.404670	...	1	0	7.0	1837.0	100
11	0.00000	0.32585	0.254769	0.229453	0.213853	0.131590	0.145256	0.352067	0.335369	0.404670	...	1	0	391.0	2409.0	0
12	0.00000	0.32585	0.254769	0.229453	0.213853	0.131590	0.145256	0.352067	0.335369	0.404670	...	1	0	391.0	2409.0	100
13	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	224.0	717.0	0
14	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	224.0	717.0	1
15	0.32585	0.00000	0.348012	0.211276	0.205345	0.134556	0.152472	0.338209	0.317697	0.268237	...	1	0	224.0	717.0	100

16 rows × 73 columns

Figure 6.200 Generated ANN general frame output

```
1 rnn
```

	IndV01	IndV02	IndV03	IndV04	IndV05	IndV06	IndV07	IndV08	IndV09	IndV10	...	no_of_epochs	batch_size	layer_no	layer_type	input_shape	no_of_n
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	61.0	5.0	0	4	246	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	61.0	5.0	1	2	0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	61.0	5.0	2	0	0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	61.0	5.0	3	3	0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	61.0	5.0	4	1	0	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	61.0	5.0	100	1	0	
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	75.0	151.0	0	4	246	
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	75.0	151.0	1	5	0	
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	75.0	151.0	2	5	0	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	75.0	151.0	100	1	0	

10 rows × 75 columns

Figure 6.201Generated RNN general frame output

After the prediction happens for each problem the configuration is loaded as in the following two examples. The configuration loader has parameters such as problem number, final independent variable, final dependent variable, unencoded dependent column, and unencoded text column.

```
annPr01M=hppGen.configurationLoader(0,x_train,y_train,unpCols[unpCols.columns[0]])
```

```
ANN con...
model = Sequential()
 model.add(Dense( 509 , activation=' relu ',input_shape=( 5 ,)))
model.add(Dense( 214 , activation=' relu '))
 model.add(Dropout( 0.05000000074505806 ))
model.add(Dense( 153 , activation=' relu '))
 model.add(Dropout( 0.05000000074505806 ))
model.add(Dense( 146 , activation=' relu '))
 model.add(Dropout( 0.14000000059604645 ))
model.add(Dense( 53 , activation=' softplus '))
 model.add(Dropout( 0.38999998569488525 ))
model.add(Dense(1))
```

Figure 6.202 Configuration loader usage by the user for a regression problem

```
rnnPr02M=hppGen.configurationLoader(1,textTrain,ratingTrain,hotel["EncodedRating"],hotel["CleanReview"])
```

```
inputs = tf.keras.Input(shape=( (1931,) ,1))
connector = tf.keras.layers.Embedding( 81486 , 90 )(inputs)
connector = tf.keras.layers.Bidirectional(Layer.LSTM( 256 ,return_sequences = True, activation=' tanh '))(connector)
connector = tf.keras.layers.GlobalMaxPool1D()(connector)
 connector = tf.keras.layers.Dropout( 0.09000000357627869 )(connector)
connector = tf.keras.layers.Dense( 64 , activation=' relu ')(connector)
 connector = tf.keras.layers.Dropout( 0.009999999776482582 )(connector)
outputs = tf.keras.layers.Dense( 3 , activation='softmax')(connector)
 model = tf.keras.Model(inputs, outputs)
```

Figure 6.203 Configuration loader usage for text classification problem by the user

After the user loads the configuration, the user can run the model runner to train the neural network, as shown in the following diagrams.

```
annPr01MHis=hppGen.modelRunner(20,1,True)
```

```
ANN con...
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 2
1/1 [==============================] - 0s 52ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 54ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 2
1/1 [==============================] - 0s 51ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 45ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 52ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 48ms/step
```

Figure 6.204Model runner method usage by the user for a regression problem

```
1 rnnPr02MHis=hppGen.modelRunner(3,32)
```

```
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 2
1/1 [==============================] - 0s 49ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 48ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 2
1/1 [==============================] - 0s 44ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 55ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 43ms/step
State1: False
filename: PredictorsClassifies/4.LostFunctionClassifire/finalized_modelLostFnp.sav Nooffiles: 1
1/1 [==============================] - 0s 52ms/step
State1: False
State1: False
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches.sav Nooffiles: 7
1/1 [==============================] - 0s 54ms/step
State1: False
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches6.sav Nooffiles: 7
1/1 [==============================] - 0s 61ms/step
State1: False
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches5.sav Nooffiles: 7
1/1 [==============================] - 0s 55ms/step
State1: False
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches4.sav Nooffiles: 7
1/1 [==============================] - 0s 54ms/step
State1: False
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches3.sav Nooffiles: 7
1/1 [==============================] - 0s 48ms/step
State1: False
filename: PredictorsClassifies/6.NoEpochesPrediction/finalized_modelnEpoches2.sav Nooffiles: 7
1/1 [==============================] - 0s 57ms/step
State1: False
```

Figure 6.205Model runner method usage for a text classification problem by the user

If you use modelRunner method use without any parameters to obtain the result with predicted hyperparameters. If you use the parameters with the number of epochs and batch size it will select the prediction results closer to the inserted two values for the number of epochs and batch size. If the user uses the method with True values the manual values will be directly assigned to the training parameters.

As the final step, the user can use the model evaluator and predictor method to evaluate the model and predict the values as in the following figure.

```
1 hppGen.evaluate(textTest,ratingTest)
```

```
212/212 [==============================] - 222s 1s/step - loss: 0.4398 - accuracy: 0.8486
```

```
1 y_resultGM=hppGen.predict(textTest)
```

```
212/212 [==============================] - 239s 1s/step
```

Figure 6.206 Usage of evaluate and predict method by the user

6.6 Summary

In this chapter, see how the general model will enable us to solve. Both ANN and RNN problems buy handling without user intervention for the neural network configuration mainly. The next chapter will look to evaluate the general model with ordinary separate.

CHAPTER 7

EVALUATION

7.1 Introduction

No matter how successful the research is, the output of the study should be verified whether the project achieves promising results. However, these research projects have succeeded here to some extent. This approach evaluates the hyperparameter prediction with the 6 existing problems in the hotel domain. One problem is repeated to test the impact of the row number in the general frame. So, seven problems are together, as indicated in the following table.

Table 7.1 Problem table for testing the solution

Problem No	Problems/data sets
01	Booking prediction problem with initial total columns(regression problem)
02	Review rating problem (text classification problem/multiclass classification)
03	Hotel classification problem (binary classification)
04	Hotel classification problem with few features (dropping:arrival_date_week_number)
05	Booking prediction problem with few features (final feature columns (2 columns:is_canceledx, is_canceledy) (regression problem)
06	Hotel classification problem (binary classification) (repeat run)
07	The hotel classification problem changed to boking recode cancellation prediction (binary classification)

Problems one and five are the same problem. The first problem is identified as an ANN problem, and the second problem is recognized as an RNN problem, which is achieved by inserting the problem's complete feature set and the problem's final feature set into the general model. The second problem is the text classification problem, as the table indicates. The 3rd and 4th problems are the same problems that measure the impact of removing a column. So even though the problem is the same, one feature is dropped in the 4th problem. The 6th problem and the 3rd problem are the same problems that measure the impact of arranging the features in the rows in the general frame. In the general frame, after adding the first row, the features of the second row are added by measuring the NLP distance for each feature with the first row. When we add the 3rd row, we measure the NLP distance for each feature with all the two already existing. So, there can be a dependency because we place the current row's features relative to previous rows. The 7th problem and the 3rd problem are also two problems that take the same feature set but different dependent variables by sending the hotel feature that indicates the country hotel and the city hotel to the independent variable and take is_cancelled as the dependent variable, which was taken as an independent variable in problem 3. Previous measures of the same feature set's impact but take different independent and dependent variables. In AutoML, it uses meta-features to select the optimized model structures. In this situation, using the same model structures to generate the solution by model ensembling is possible.

7.2 Training results comparison table

The final performances achieved are as follows. The code for taking the results is explained by using problem one for the regression problem and problem three for the classification problem.

First, the values of the expert configuration and the HPPgenral model are taken as shown in the figure on the next page, taking the difference between the actual value and the value provided by the HPPGenral model and then taking the difference between the user configuration and the exact value. If the HPP general model obtains the minor difference, it is indicated as “actual-GM” and otherwise indicated as “actual-UC.” Then, the count has taken how much actual-GM and the actual-UC are present in the results.

```
compPro01=pd.DataFrame()
compPro01.insert(0,"Actual",acVal,True)
compPro01.insert(1,"Genaral_model",gmVal,True)
compPro01.insert(2,"User config",uSerVal,True)
compPro01["actual-GM"]=compPro01.apply(lambda row: abs(row['Actual'] - row['Genaral_model']), axis=1)
compPro01["actual-UC"]=compPro01.apply(lambda row: abs(row['Actual'] - row['User config']), axis=1)
compPro01["Result"]=compPro01.apply(lambda row: "actual-UC" if row["actual-UC"]<row["actual-GM"] else "actual-GM", axis=1)
compPro01
```

	Actual	Genaral_model	User config	actual-GM	actual-UC	Result
0	699.0	484.196822	555.889771	214.801178	143.110229	actual-UC
1	491.0	484.638580	568.782349	6.361420	77.782349	actual-GM

Figure 7.1 Evaluating a regression problem for the HPPGeneral Model

The final result is indicated in the following figure.

```
compPro01["Result"].value_counts()
```

```
actual-UC    10
actual-GM     7
Name: Result, dtype: int64
```

Figure 7.2result of the HPPGeneral model for a regression problem

Then, the same process is performed with AutoML and user configuration. Because AutoML can't run in Windows, the Ubuntu operation system is used. The following figure shows the code and the result.

```
compPro01=pd.DataFrame()
compPro01.insert(0,"Actual",acVal,True)
compPro01.insert(1,"AutoML_model",amlVal,True)
compPro01.insert(2,"User config",uSerVal,True)
compPro01["actual-AML"]=compPro01.apply(lambda row: abs(row['Actual'] - row['AutoML_model']), axis=1)
compPro01["actual-UC"]=compPro01.apply(lambda row: abs(row['Actual'] - row['User config']), axis=1)
compPro01["Result"]=compPro01.apply(lambda row: "actual-UC" if row["actual-UC"]<row["actual-AML"] else "actual-AML", axis=1)
compPro01
```

	Actual	AutoML_model	User config	actual-AML	actual-UC	Result
0	699.0	545.670688	550.957397	153.329312	148.042603	actual-UC
1	491.0	541.695663	563.494507	50.695663	72.494507	actual-AML
2	745.0	524.207160	558.422058	220.792840	186.577942	actual-UC
3	537.0	555.894301	567.833252	18.894301	30.833252	actual-AML

Figure 7.3 Evaluating the AutoML for a regression problem in deep learning

The final result is shown in the following figure.

```
compPro01["Result"].value_counts()
```

```
Result
actual-UC     9
actual-AML    8
Name: count, dtype: int64
```

Figure 7.4 Result of the AutoML for a regression problem

Regarding classification, there is a value between true and false. So, we compare the value relative to the user configuration and HPPGenral Model as in the following figure.

```
compPro03=pd.DataFrame()
```

```
compPro03.insert(0,"Actual",acVal,True)
compPro03.insert(1,"Genaral_model",gmVal,True)
compPro03.insert(2,"User config",uSerVal,True)
```

```
compPro03["actual-GM"]=compPro03.apply(lambda row: True if row['Actual']==row['Genaral_model'] else False, axis=1)
compPro03["actual-UC"]=compPro03.apply(lambda row: True if row['Actual']==row['User config'] else False, axis=1)
```

```
compPro03
```

	Actual	Genaral_model	User config	actual-GM	actual-UC
0	0	0	0	True	True
1	0	0	0	True	True
2	0	0	0	True	True
3	0	0	0	True	True
4	1	1	1	True	True
...	...	...	...	...	...
35812	0	0	0	True	True
35813	0	0	0	True	True
35814	0	0	0	True	True

Figure 7.5Evaluating HPPGegeral model for a classification problem

Then, count the true values of actual value and HPPGenral Model value and actual and user configuration variables. It shows the following result.

```
compPro03["actual-GM"].value_counts()
```

```
True     34851
False      966
Name: actual-GM, dtype: int64
```

```
compPro03["actual-UC"].value_counts()
```

```
True     34989
False      828
Name: actual-UC, dtype: int64
```

Figure 7.6 Result of the HPPGeneral model for a classification problem

The same is done between the user and the actual value and the AutoML and actual value, as shown in the following diagram on the next page.

```
compPro03=pd.DataFrame()
compPro03.insert(0,"Actual",acVal,True)
compPro03.insert(1,"AutoML_model",amlVal,True)
compPro03.insert(2,"User config",uSerVal,True)
compPro03["actual-AML"]=compPro03.apply(lambda row: True if row['Actual']==row['AutoML_model'] else False, axis=1)
compPro03["actual-UC"]=compPro03.apply(lambda row: True if row['Actual']==row['User config'] else False, axis=1)
compPro03["Result"]=compPro03.apply(lambda row: True if row["actual-UC"]==row["actual-AML"] else False, axis=1)
compPro03
```

	Actual	AutoML_model	User config	actual-AML	actual-UC	Result
0	0	0	0	True	True	True
1	0	0	0	True	True	True
2	0	0	0	True	True	True

Figure 7.7 Evaluating AutoML for a classification problem in deep learning

Similarly, we got the count result between actual values and AutoML, followed by the user value and the actual value results in the following diagram.

```
]: compPro03["actual-AML"].value_counts()
]: actual-AML
   True     33973
   False     1844
   Name: count, dtype: int64

]: compPro03["actual-UC"].value_counts()
]: actual-UC
   True     35032
   False      785
   Name: count, dtype: int64
```

Figure 7.8 Result of AutoML for the classification problem in deep learning

Similar calculations have been done for other problems, such as regression or classification, as shown in the following table.

Table 7.2performance comparison between expert and hyperparameter prediction by AutoML and HPPGeneral Model

Problems/data sets	Expert configuration W-windows U-ubuntu	HPPGeneral Model W-windows	AutoML U-ubuntu
01 (booking prediction-regression) ANN	actual-UC =10(W)	actual-GM= 7	
	actual-UC= 9 (U)		actual-AML = 8
02 (Review rating)	True : 5675 False : 1088(W)	True : 5739 False : 1024	
	True: 5612 False :1151 (U)		True: 4955 False: 1808
03(hotel classification)	True: 34989 False: 828 (W)	True: 34851 False: 966	
	True : 35032 False :785 (U)		True: 33973 False : 1844
04 (hotel classification by dropping 1 col)	True:23628 False: 12189(W)	True: 23628 False: 12189	
	True: 23628 False:12189		True:30894 False: 4923
05(booking prediction-regression-2columns) RNN	actual-UC =10 (W)	actual-GM= 7	
	actual-UC = 10 (U)		actual-AML = 7
06 (hotel classification rerun)	True: 34989 False: 828 (W)	True: 34851 False: 966	
	True : 35032 False :785(U)		True: 33973 False : 1844
07 booking cancellation by hotel classification DS	True: 35803 False : 14	True: 35805 False: 12	
	True: 35798 False : 19		True 35817 False :0

In the above table, the double-underlined values indicate the best value per each problem regardless of regression or classification. A single line in the value for the classification problem indicates the best selection in two AutoML and HPPGenral Model user configurations. The issues indicated in the dashed underline with the pattern as in problem 6 in the column Problems/data sets indicate significant differences in the HPPGeneral model. In contrast, the pattern in problem 4 indicates considerable differences to AutoML.

In general, the results obtained can be considered similar. However, occasionally, they give significant results based on the dataset and the problem. The purpose of the AutoML platforms is to make machine learning easy for those who don't have additional knowledge of ML or DL. One can check the results using both the HPPGeneral Model approach and AutoML. Due to the separate installation of an operating system by allocation of 40GB or more, hard disk and 8GB or more RAM can be worthless for use AutoML.

AutoML	HPPGeneral Model
can't run on Windows	It can run on Windows.
The black-box nature of the solution	It shows all the steps as manual configuration but doesn't explain the results in the same way as AutoML or LLM approaches.
run of up to 25 models for a single problem	Single run per problem

Figure 7.9 Comparison of the HPPGeneral model with AutoML

7.3 Summary

This chapter attempted to evaluate the general model and now focuses on the final chapter, elaborating on what outcomes have been made, what will be possible in the future, and the recommendations.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Introduction

This final chapter will discuss the solution's final interpretation of the solution, its future developments, and contributions. Considering this feature selection approach, the system is trained with low data, such as 40 data sets. Because there is no publicly available hyperparameter data related to ANN, RNN, or any other models, the dataset must be manually created by the experts' manual observation of configurations. On the other hand, different experts configure neural networks differently. So, putting them on a standard table is a hectic task sometimes because it has to take the differently configured networks to a standard format. Population-based algorithms handle this by generating configurations with restrictions[29].

When considering the fulfilment of the objectives of this research project, it can be realized that the objectives have been achieved in the following ways. Objective 01: A critical review of existing hyperparameter prediction and optimization approaches extending to hotel classification & review rating classification achieved by 46 literature reviews. Objective 02 In-depth study of the latest research on customized neural networks and ANN, RNN-based Technologies achieved by studying and developing ANN and Vanilla LSTM tutorials. Objective 03 Design and develop a neural network-based system that can automate the neural network type selection for ANN and RNN with hyperparameter prediction for the hotel domain achieved by Python and TensorFlow. Objective 04 Evaluate the Solution to Hotel domain problems with datasets achieved by six issues—objective 05 Preparation of final documentation achieved by Word.

8.2 Other Important Research Outputs

The general frame used to accomplish the general model will be helpful in other sectors, such as natural language processing activities, as a clustering technique in the future. NLP encoding can be used when you have diverse data other than label encoding because NLP encoding energizes semantic and syntactic meanings while encoding, giving more promising results.

Hyperparameter tuning has already been an excellent approach to building a good neural network. There are many hyperparameter tuning frameworks, such as OPTUNA [43]. However, there is no approach for hyperparameter prediction due to the lack of a method in connection with data to predict them, which is because of the discovery of the general frame by this research. This hyperparameter prediction will powerfully energize the AGI in the future because it will make the neural network creation and development a fully automated task.

8.3 Recommendations

The neural network results in performing the NLP encoding were not stable. In other words, when one time is trained, it gives one good result, and, at another time, it provides a different result that will be established with more training data introduced, so adding more data to stable results is recommended.

The evaluation section shows that the HPPGeneral model gives better results than AutoML when considering deep learning. However, AutoML can also provide better results on some occasions. As explained in the last part of the evaluation, users can enjoy the advantage of using the HPPGeneral model. If the results do not seem reasonable, they can try AutoML because it doesn't work for Windows operating systems.

As future developments, it can stabilise all 34 predictors and classifiers with more data up to 200 with 6-layer problems. Moreover, test solutions with more problems, further develop the restrictors and Automate data preprocessing. Further, create a solution for issues with more than one dependent variable and develop the solution for other types, such as CNN.

The model training with more data records than the newly trained predictors and classifiers easily inserted into the system results in far better results in the future that may supersede the expert configuration like in LLM [42]. Previous approaches were achieved while keeping the lightweight structure.

8.4 Summary

This final chapter summarises everything, including the potential of the results of this research and the places to develop more in the future. So, in the future, many hyperparameter prediction approaches will fuse to achieve high-quality hyperparameter prediction solutions. I hope this approach will be part of them.

References

[1] “analytixlabs-data-science-skills-survey-report-2024.pdf.” Accessed: Jul. 16, 2024. [Online]. Available: https://api.analytixlabs.co.in/image/brochure/analytixlabs-data-science-skills-survey-report-2024.pdf

[2] P. P. Ray, “ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope,” *Internet Things Cyber-Phys. Syst.*, vol. 3, pp. 121–154, Jan. 2023, doi: 10.1016/j.iotcps.2023.04.003.

[3] P. Shankar, “A Review on Artificial Neural Networks,” vol. 3, pp. 166–169, Apr. 2022.

[4] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J. Big Data*, vol. 8, no. 1, Art. no. 1, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[5] J. Zhou *et al.*, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, Jan. 2020, doi: 10.1016/j.aiopen.2021.01.001.

[6] C. Chopra, S. Sinha, S. Jaroli, A. Shukla, and S. Maheshwari, “Recurrent Neural Networks with Non-Sequential Data to Predict Hospital Readmission of Diabetic Patients,” Oct. 2017, pp. 18–23. doi: 10.1145/3155077.3155081.

[7] E. Nisioti, K. C. Chatzidimitriou, and A. L. Symeonidis, “Predicting hyperparameters from meta-features in binary classification problems”.

[8] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, “Auto-sklearn: Efficient and Robust Automated Machine Learning,” in *Automated Machine Learning*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., in The Springer Series on Challenges in Machine Learning. , Cham: Springer International Publishing, 2019, pp. 113–134. doi: 10.1007/978-3-030-05318-5_6.

[9] M. Sarker, S. Noor, and U. Acharjee, “Basic Application and Study of Artificial Neural Networks,” *SK Int. J. Multidiscip. Res. Hub*, vol. 4, pp. 1–12, May 2017.

[10] M. Sazli, “A brief review of feed-forward neural networks,” *Commun. Fac. Sci. Univ. Ank.*, vol. 50, pp. 11–17, Jan. 2006, doi: 10.1501/0003168.

[11] G. Van Houdt, C. Mosquera, and G. Nápoles, “A Review on the Long Short-Term Memory Model,” *Artif. Intell. Rev.*, vol. 53, Dec. 2020, doi: 10.1007/s10462-020-09838-1.

[12] B. Goertzel, “Artificial General Intelligence: Concept, State of the Art, and Future Prospects,” *J. Artif. Gen. Intell.*, vol. 0, Jan. 2014, doi: 10.2478/jagi-2014-0001.

[13] D. Jilk, C. Lebiere, R. O’Reilly, and J. Anderson, “SAL: An explicitly pluralistic cognitive architecture,” *J Exp Theor Artif Intell*, vol. 20, pp. 197–218, Sep. 2008, doi: 10.1080/09528130802319128.

[14] T. Achler, “Towards Bridging the Gap Between Pattern Recognition and Symbolic Representation Within Neural Networks,” p. 7.

[15] A. Nestor and B. Kokinov, “TOWARDS ACTIVE VISION IN THE DUAL COGNITIVE ARCHITECTURE,” *Int. J.*, p. 7.

[16] J. Bach, “Principles of Synthetic Intelligence PSI An Architecture of Motivated Cognition,” *Princ. Synth. Intell. PSI Archit. Motiv. Cogn.*, pp. 1–400, Nov. 2009, doi: 10.1093/acprof:oso/9780195370676.001.0001.

[17] M. Haenlein and A. Kaplan, “A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence,” *Calif. Manage. Rev.*, vol. 61, p. 000812561986492, Jul. 2019, doi: 10.1177/0008125619864925.

[18] A. M. Kalteh, P. Hjorth, and R. Berndtsson, “Review of the Self-Organizing Map (SOM) approach in water resources: analysis, modelling and application,” *Environ. Model. Softw.*, vol. 23, pp. 835–845, Jul. 2008, doi: 10.1016/j.envsoft.2007.10.001.

[19] J. Larsen, C. Svarer, L. N. Andersen, and L. K. Hansen, “Adaptive Regularization in Neural Network Modeling,” in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., Berlin, Heidelberg: Springer, 2012, pp. 111–130. doi: 10.1007/978-3-642-35289-8_8.

[20] Y. Bengio, “Gradient-Based Optimization of Hyperparameters,” *Neural Comput.*, vol. 12, no. 8, pp. 1889–1900, Aug. 2000, doi: 10.1162/089976600300015187.

[21] C.-S. Foo, C. B. Do, and A. Y. Ng, “A majorization-minimization algorithm for (multiple) hyperparameter learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, Montreal Quebec Canada: ACM, Jun. 2009, pp. 321–328. doi: 10.1145/1553374.1553415.

[22] C. Rasmussen, O. Bousquet, U. Luxburg, and G. Rätsch, “Gaussian Processes in Machine Learning,” *Adv. Lect. Mach. Learn. ML Summer Sch. 2003 Canberra Aust. Febr. 2 - 14 2003 Tüb. Ger. August 4 - 16 2003 Revis. Lect. 63-71 2004*, vol. 3176, Sep. 2004, doi: 10.1007/978-3-540-28650-9_4.

[23] T. Sufi and S. Singh, “Hotel Classification Systems : A Case Study,” *Prabandhan Indian J. Manag.*, vol. 11, p. 52, Jan. 2018, doi: 10.17010/pijom/2018/v11i1/120823.

[24] R. Chowdhury and A. Deshpande, “An Analysis of the Impact of Reviews on the Hotel Industry,” *Ann. Trop. Med. Public Health*, vol. 23, Jan. 2020, doi: 10.36295/ASRO.2020.231742.

[25] T. Yu and H. Zhu, “Hyper-Parameter Optimization: A Review of Algorithms and Applications,” Mar. 12, 2020, *arXiv*: arXiv:2003.05689. doi: 10.48550/arXiv.2003.05689.

[26] T. SERIZAWA and H. FUJITA, “Optimization of Convolutional Neural Network Using the Linearly Decreasing Weight Particle Swarm Optimization,” 2022, *一般社団法人 人工知能学会*: 0. doi: 10.11517/pjsai.JSAI2022.0_2S4IS2b03.

[27] R. Vilalta, C. Giraud-Carrier, and P. Brazdil, “Meta-Learning - Concepts and Techniques,” in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and

L. Rokach, Eds., Boston, MA: Springer US, 2009, pp. 717–731. doi: 10.1007/978-0-387-09823-4_36.

[28] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, “Ensemble selection from libraries of models,” in *Twenty-first international conference on Machine learning - ICML ’04*, Banff, Alberta, Canada: ACM Press, 2004, p. 18. doi: 10.1145/1015330.1015432.

[29] B. Panda, *Hyperparameter Tuning*. 2019. doi: 10.13140/RG.2.2.11820.21128.

[30] R. S. Segall, “Some mathematical and computer modelling of neural networks,” *Appl. Math. Model.*, vol. 19, no. 7, Art. no. 7, Jul. 1995, doi: 10.1016/0307-904X(95)00021-B.

[31] A. Bielecki, “Mathematical model of architecture and learning process of artificial neural networks,” *TASK Q.*, vol. 7, pp. 93–114, Jan. 2003.

[32] P. Zhang, “Neural Networks for Classification: A Survey,” *Syst. Man Cybern. Part C Appl. Rev. IEEE Trans. On*, vol. 30, pp. 451–462, Dec. 2000, doi: 10.1109/5326.897072.

[33] K. Potdar, T. Pardawala, and C. Pai, “A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers,” *Int. J. Comput. Appl.*, vol. 175, pp. 7–9, Oct. 2017, doi: 10.5120/ijca2017915495.

[34] “Review Based on Different Deep Learning Architecture and their Application ijariie7462.” Accessed: Nov. 18, 2022. [Online]. Available: http://ijariie.com/AdminUploadPdf/Review_Based_on_Different_Deep_Learning_Architecture_and_their_Application_ijariie7462.pdf

[35] M. Hassan *et al.*, “Neuro-Symbolic Learning: Principles and Applications in Ophthalmology,” Jul. 31, 2022, *arXiv*: arXiv:2208.00374. Accessed: Nov. 17, 2022. [Online]. Available: http://arxiv.org/abs/2208.00374

[36] X. Chen and W. Wang, “An overview of Hierarchical Temporal Memory: A new neocortex algorithm,” presented at the Proceedings of 2012 International

Conference on Modelling, Identification and Control, ICMIC 2012, Jan. 2012, pp. 1004–1010.

[37] G. Edelman, “Neural Darwinism,” *New Perspect. Q.*, vol. 21, pp. 62–64, Jul. 2004, doi: 10.1111/j.1540-5842.2004.00685.x.

[38] M. Minsky, *The society of mind*. New York: Simon and Schuster, 1986.

[39] L. Shastri and V. Ajjanagadde, “From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony,” *Behav. Brain Sci.*, vol. 16, no. 3, pp. 417–451, Sep. 1993, doi: 10.1017/S0140525X00030910.

[40] S. Liu, C. Gao, and Y. Li, “Large Language Model Agent for Hyper-Parameter Optimization,” Feb. 06, 2024, *arXiv*: arXiv:2402.01881. Accessed: Apr. 28, 2024. [Online]. Available: http://arxiv.org/abs/2402.01881

[41] S. Zhang, C. Gong, L. Wu, X. Liu, and M. Zhou, “AutoML-GPT: Automatic Machine Learning with GPT,” May 03, 2023, *arXiv*: arXiv:2305.02499. Accessed: May 03, 2024. [Online]. Available: http://arxiv.org/abs/2305.02499

[42] L. Zhang, Y. Zhang, K. Ren, D. Li, and Y. Yang, “MLCopilot: Unleashing the Power of Large Language Models in Solving Machine Learning Tasks,” in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, Y. Graham and M. Purver, Eds., St. Julian’s, Malta: Association for Computational Linguistics, Mar. 2024, pp. 2931–2959. Accessed: May 04, 2024. [Online]. Available: https://aclanthology.org/2024.eacl-long.179

[43] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019, p. 2631. doi: 10.1145/3292500.3330701.

[44] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, *Google Vizier: A Service for Black-Box Optimization*. 2017, p. 1495. doi: 10.1145/3097983.3098043.

[45] H. Shaziya and R. Zaheer, “Impact of Hyperparameters on Model Development in Deep Learning,” 2021, pp. 57–67. doi: 10.1007/978-981-15-8767-2_5.

[46] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, “Hyperopt: a Python library for model selection and hyperparameter optimization,” *Comput. Sci. Discov.*, vol. 8, no. 1, p. 014008, Jul. 2015, doi: 10.1088/1749-4699/8/1/014008.

[47] “Welcome to SigOpt! | SigOpt Documentation.” Accessed: Apr. 30, 2024. [Online]. Available: https://docs.sigopt.com

[48] *autonomio/talos*. (Apr. 30, 2024). Python. Autonomio. Accessed: Apr. 30, 2024. [Online]. Available: https://github.com/autonomio/talos

[49] “Pandas Introduction.” Accessed: May 07, 2023. [Online]. Available: https://www.w3schools.com/python/pandas/pandas_intro.asp

[50] S. Yegulalp, “What is TensorFlow? The machine learning library explained,” InfoWorld. Accessed: May 07, 2023. [Online]. Available: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[51] “Rain Prediction: ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/karnikakapoor/rain-prediction-ann

[52] “Intro to Keras with breast cancer data[ANN].” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/thebrownviking20/intro-to-keras-with-breast-cancer-data-ann

[53] “Single RNN with 4 folds (CLR).” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/shujian/single-rnn-with-4-folds-clr

[54] “Google stock price prediction - RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/ptheru/google-stock-price-prediction-rnn

[55] “Heart Failure Prediction: ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/karnikakapoor/heart-failure-prediction-ann

[56] “Mushroom Classification with ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/gulsahdemiryurek/mushroom-classification-with-ann

[57] “Predicting BTC Price Using RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/microtang/predicting-btc-price-using-rnn

[58] “Associated Model RNN + Ridge.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/nvhbk16k53/associated-model-rnn-ridge

[59] “Customer Churn Prediction Using ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/niteshyadav3103/customer-churn-prediction-using-ann

[60] “Predicting House Prices (Keras - ANN).” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/tomasmantero/predicting-house-prices-keras-ann

[61] “Fake News Detection Using RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/therealcyberlord/fake-news-detection-using-rnn

[62] “Lyrics Generator: RNN🎵 🎤 👩‍🎤 🎵.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/karnikakapoor/lyrics-generator-rnn

[63] “💳Credit Card Fraud 💸Detection 🚨 ANNs vs XGBoost.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/faressayah/credit-card-fraud-detection-anns-vs-xgboost

[64] “ANN starter.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/thebrownviking20/ann-starter

[65] “Deep Learning For NLP: Zero To Transformers & BERT.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/tanulsingh077/deep-learning-for-nlp-zero-to-transformers-bert

[66] “RNN_Detailed Explanation_[0.2246].” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/shanth84/rnn-detailed-explanation-0-2246

[67] “Who said this line[EDA/Classification/Keras/ANN].” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/thebrownviking20/who-said-this-line-eda-classification-keras-ann

[68] “[ANN/SLP] Making Model for Multi-Classification.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/mirichoi0218/ann-slp-making-model-for-multi-classification

[69] “PLAsTiCC RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/zerrxy/plasticc-rnn

[70] “Deep Learning Multivariate RNN / LSTM Network.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/minhajulhoque/deep-learning-multivariate-rnn-lstm-network

[71] “ANN on Electricity Consumption.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/utathya/ann-on-electricity-consumption

[72] “Minimizing Risks for Loan Investments Keras - ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/tomasmantero/minimizing-risks-for-loan-investments-keras-ann

[73] “Hourly energy consumption time series RNN, LSTM.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/msripooja/hourly-energy-consumption-time-series-rnn-lstm

[74] “Sarcasm Detection : RNN-LSTM.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/tanumoynandy/sarcasm-detection-rnn-lstm

[75] “Titanic Prediction - ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/farizhaykal/titanic-prediction-ann

[76] “healthcare-dataset-stroke-data.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/rishabh057/healthcare-dataset-stroke-data

[77] “Comparison of ML models with RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/jaswanthhbadvelu/comparison-of-ml-models-with-rnn

[78] “Botnet Host Prediction using RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/asindico/botnet-host-prediction-using-rnn

[79] “Autistic Patients EDA + Classification Using ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/muhammadshahrayar/autistic-patients-eda-classification-using-ann

[80] “Heart Attack Prediction | EDA | ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/anubhavgoyal10/heart-attack-prediction-eda-ann

[81] “Sentiment Analysis with RNNs - Disaster Tweets.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/dariocioni/sentiment-analysis-with-rnns-disaster-tweets

[82] “Transformer-RNN-Tool-Box-for-Text-Classification.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/emmanuelpintelas/transformer-rnn-tool-box-for-text-classification

[83] “hotel-reservations.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/chaitnyapol/hotel-reservations

[84] “ml_ann_ch.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/yunchuangao/ml-ann-ch

[85] “Sentiment Analysis(ML & RNN).” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/smitshah00/sentiment-analysis-ml-rnn

[86] “Simple LSTM for text classification.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/kredy10/simple-lstm-for-text-classification

[87] “Travel Customer Churn Prediction using simple ANN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/venkatganesh98/travel-customer-churn-prediction-using-simple-ann

[88] “Credit analysis with KNN/DTree/RF/Bagging/ANN 🏦.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/fahadmehfoooz/credit-analysis-with-knn-dtree-rf-bagging-ann

[89] “Hate speech detection : RNN.” Accessed: May 13, 2023. [Online]. Available: https://kaggle.com/code/nayansakhiya/hate-speech-detection-rnn

Appendices