B.TECH. PROJECT REPORT

On

Multi-label Classification of Genome Data using Soft Computing

BY

Sahaj Khandelwal, 160001052
&
Niranjan Joshi, 160001026

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

November, 2019

Multi-label Classification of Genome Data using Soft Computing

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Sahaj Khandelwal, 160001052
&
Niranjan Joshi, 160001026,

Discipline of Computer Science and Engineering,

Indian Institute of Technology, Indore

Guided by:

Dr. Aruna Tiwari,

Associate Professor,

Computer Science and Engineering,

IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November, 2019

CANDIDATES' DECLARATION

We hereby declare that the project entitled **"Multi-label Classification of Genome Data using Soft Computing"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Aruna Tiwari, Associate Professor, Computer Science and Engineering**, IIT Indore is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Sahaj Khandelwal **Niranjan Joshi**

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge.

Dr. Aruna Tiwari,
Associate Professor,
Discipline of Computer Science and Engineering,
IIT Indore

PREFACE

This report on "Multi-label Classification of Genome Data using Soft Computing" is prepared under the guidance of Dr. Aruna Tiwari, Associate Professor, Computer Science and Engineering, IIT Indore.

Through this report, we have tried to provide a detailed description of our approach, design, and implementation of an innovative method to perform Multi-label Classification for Genome Data. We tried to analyze the dataset and perform filtering through various measures. While proposing improvements in the existing models to perform the classification, we also devised a heuristic approach to handle the data imbalance encountered during the classification of a multi-label dataset. The weight modulation techniques used to improvise on the precision and recall of the individual label samples have been thoroughly described.

We have tried our best to explain the proposed solution, along with the detailed analysis of our devised heuristic approach to handle the data imbalance found in the multi-label dataset used for classification.

ACKNOWLEDGEMENTS

We want to thank our B.Tech Project supervisor **Dr. Aruna Tiwari** for her guidance and constant support in structuring the project and providing valuable feedback throughout the course of this project. Her overseeing the project meant there was a lot that we learnt while working on it. We thank her for her time and efforts.

We are grateful to **Mr. Vikas Chauhan**, without whom this project would have been impossible. He provided valuable guidance to handle the delicacies involved in the project and also taught us how to write a scientific paper.

We are grateful to the Institute for the opportunity to be exposed to systemic research, especially Dr. Aruna Tiwari's Lab, for providing the necessary hardware utilities to complete the project.

Lastly, we offer our sincere thanks to everyone who helped us complete this project, whose name we might have forgotten to mention.

Sahaj Khandelwal & Niranjan Joshi,
B.Tech. 4^{th} **Year**
Discipline of Computer Science and Engineering,
IIT Indore

ABSTRACT

Biological data mining aims to extract meaningful information from DNA, RNA and proteins. The information could pertain to clustering and classification rules between functionalities and gene families. Classification of genome data is naturally a widely studied and essential area of research nowadays, with a whole field of molecular biology and functional genomics dedicated to understanding the gene functions from genome data.

Through this work, we propose an efficient method to classify the proteins into their functional classes through multi-label classification using soft-computing techniques. Along with achieving high efficiency in classifying the proteins, we propose a heuristic approach to improve the precision and recall in predicting the individual functional classes. The effectiveness of the proposed heuristics is evaluated through testing and comparing with various models based on ANN architecture, using the performance metrics such as precision, recall, AUC and subset accuracy. The proposed approach is found to result in a substantial increase in the precision and recall of the individual functional classes, also called labels.

Artificial neural networks (ANNs) have climbed up to popularity among various machine learning tools, owing to the recent success achieved in image and sound processing classification problems. Here, we apply ANNs to predict the functional classes the proteins belong to, knowing their residue sequence. Through various experimentation in the model architecture, we studied the variation of prediction effectiveness with the properties of ANN. Here we present a new ANN with multi-label classification ability, showing impressive accuracy when classifying protein sequences into 1665 Gene Ontology classes (AUC=99.94%).

The greatest challenge with multi-label classification is to handle the data imbalance, which appears due to variance in frequencies of the labels. This is handled through weight modulation in the loss function, to influence the learning process. The heuristic approach proposed can further be extended to construct methodologies for performing a better multi-label classification.

Contents

List of Tables

List of Figures

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
TN	**T**rue **N**egative
FN	**F**alse **N**egative
AUC	**A**rea **U**nder **C**urve
ROC	**R**eceiver **O**perating **C**haracteristic
SA	**S**ubset **A**ccuracy
avg	**a**verage

Chapter 1

Introduction

1.1 Background

Genomics is an interdisciplinary field of biology focusing on the structure, function, evolution, mapping, and editing of genomes. A genome is an organism's complete set of DNA, including all of its genes. Genes may direct the production of proteins with the assistance of enzymes and messenger molecules. In turn, proteins make up body structures such as organs and tissues as well as control chemical reactions and carry signals between cells. Genomics also involves the sequencing and analysis of genomes through uses of high throughput DNA sequencing and bioinformatics to assemble and analyze the function and structure of entire genomes.

Proteins are vastly studied in highly sophisticated laboratories, using advanced computational approaches. One of the most important problems, revolving around protein studies is the functional annotation/classification of proteins, using only structural information[1]. The problem of functionally annotating the proteins refers to classifying proteins into various functional categories, based upon the available structural information of the proteins. Classification of proteins based on the structural organization takes place on four different levels: primary, secondary, tertiary, and quaternary structure. The sequence of amino acids in the polypeptide chain is referred to by the primary structure of the protein. The secondary structure represents the highly regular local substructures on the polypeptide backbone chain. There are two main types of secondary substructure classification categories: the α-helix and the β-strand or β-sheets. Tertiary structure refers to the three-dimensional geometry of the folded substructures. Quaternary structure is the three-dimensional structure consisting of the aggregation of two or more individual polypeptide chains (subunits) that operate as a single functional unit. Hence, the complexity of the protein structure increases from the primary structure to the quaternary structure.

The problem of functional annotation of proteins elevates with a deficiency in the amount of structural information available about the protein. With the knowledge of tertiary and quaternary structures of protein, it is possible to find certain distinguishing characteristics of the protein, that dictate its main functionalities. Whereas, only the knowledge of the primary structure of the proteins makes it more challenging to make a correct functional annotation to the macro-molecules. This can be viewed as knowing only a high-dimensional representation of the protein structure, where the dimension corresponds to the length of the amino acid sequence.

One of the possible methods to do this annotation would be to apply a sequence alignment based similarity search between the input sequence and a properly chosen functionally annotated database[1]. Many algorithms, such as the exact Smith-Waterman algorithm[2, 3], BLAST[4], or hidden Markov-model based search[5, 6, 7] can be used for sequence alignment. Once the most similar sequence to the input sequence is found in the database, its functional annotation is assigned to the input sequence. In other words, the input is assigned the function of the most similar sequence in a reference database. One of the significant problems with this approach is that protein sequences have varying relevance in similarity based on the extent of how conservative their sub-sequences are. Also, the 3-dimensional structure of the proteins is more conserved during the evolution as compared to the primary structure, as a result of which two sequences could have the same three-dimensional geometry, and hence, the same functionalities, but have different primary structures. Due to this, the above approach would falter[8].

This marks the need for more sophisticated methods to perform classification than the conventional sequence alignment search. A very fast-growing field of research in protein classification is the use of ANNs(Artificial Neural Networks). ANNs have been applied frequently in numerous image and sound processing and classification problems[9, 10, 11, 12]. The basic building blocks of the ANNs are the artificial neurons[13], that initially compute the weighted sum of the inputs, and then apply a non-linear function(also called the activation function) to the resultant. The output of one neuron is fed as the input of the other. These neurons, usually work in layers, such that the output of a layer is the input of the other. While applying to a classification problem, the output classes are assigned a distinct neuron in the final layer, which is activated if the input is successfully classified into the corresponding class. ANNs learn by modifying the weights assigned to each neuron, in every cycle of training, eventually intending to minimize the loss function. Specifying an appropriate activation function, proper architecture, and suitable loss function can help in better training of the neural network, hence increasing the classification accuracy. Weights are updated through backward propagation mechanisms, some of which are Stochastic Gradient Descent(SGD)[14], RMSprop or Adam[15, 16, 17]. When the input can be classified into multiple classes all at once, it is a multi-label classification problem. Here, the protein can be classified into multiple functional classes at once, and hence, the classification of proteins with their amino acid sequence as the input, and the functional classes as outputs is a multi-label classification problem.

1.2 Problem Specification

Proteins can be classified into four different structural levels based on the spatial arrangements of the constituent amino acids. The classification is into the primary structure, secondary structure, tertiary structure and quaternary structure. The tertiary and quaternary properties refer to the three-dimensional organization of the constituting amino acids, which is crucial in determining the functional properties of the corresponding protein. With the prior knowledge of the three-dimensional structural organization of the protein, it is possible to find the distinguishing characteristics of the protein with certainty. Whereas, with only the primary structural information available, it becomes more challenging to functionally annotate the protein.

Hence, the problem statement can be specified as functionally annotating the protein, based on the available structural information (the primary structure, essentially the amino acid sequencing of the protein).

The problem statement is thus broken down into the objectives described as in the following section.

1.3 Objectives

Multi-label classification is a generalization of multi-class classification, which is the single-label problem of categorizing instances into precisely one of more than two classes; in the multi-label problem, there is no constraint on how many of the classes the instance can be assigned to. Formally, multi-label classification is the problem of finding a model that maps inputs x to binary vectors y (assigning a value of 0 or 1 for each element (label) in y).

The task of functionally annotating the proteins into various functional classes, relates to being a multi-label classification, for a protein can be performing many functions, and hence, being classified into multiple classes at once.
The main objectives of this project are:

- Understanding the Genome Data(SwissProt subset of the UniProt Dataset[18]) to gain insights on the useful features which determine the functions of the proteins. Finding the appropriate information is necessary for the task of classification in order to incorporate the features of the proteins while still optimizing the amount of data used.

- Once the dataset is understood and the suitable information identified, further pre-processing needs to be done in order to make the data ready to be applied to the neural network. For this, various filtering methods were applied before encoding the protein sequence for mathematical realization.

- After the pre-processing is done, and the sequences have been mathematically encoded, 1-D CNN (Convolutional Neural Network) is applied for the desired multi-label classification of Genome Data.

The rest of the report is organized as follows. In chapter 3, we propose the Design and Architecture of the 1-D CNN to be applied, discussing the advantages of using convolutional neural networks and the suitability of CNN for the problem statement. Chapter 4 describes the experimental setup, mentioning the pre-processing methods, various models along with their performance metrics. The report concludes in Section 5.

Chapter 2

Literature Survey

The following chapter discusses literature pertaining to previously known methods of protein classification/functional annotation, providing a short description of the methodology and the dataset used, filtering, and the accuracy attained.

2.1 Multi-Label Classification

The binary classification is a type of supervised learning problem, where the input is classified into one out of two classes. Whenever there are more than two classes to choose from, it becomes a multi-class classification problem. Here, the input can be classified into one of many classes. Multi-label classification is the problem of assigning the input to multiple classes all at once.

In the protein classification performed using neural networks in [19], the proteins were stored in 20 x 20 bi-peptide matrices. The training was performed in an unsupervised manner, as a result of which, self-organization of the activation of the neurons took place into a topologically ordered map, such that the proteins belonging to a known family were associated with the same neuron or one neighbouring it. This self-organization into topologically ordered maps made the classification fast for new inputs. Filtering: protein sequences with length greater than 50, dataset after filtering: 1758 protein sequences.

In [20], the protein sequences were encoded into the input vectors to the neurons by applying the n-gram hashing or SVD(singular value decomposition) method. The annotated PIR(Protein Identification Resource) database was used, and the input was applied to a three-layered, feed-forward neural network that employed the back-propagation learning algorithm. The target classes were pairwise disjoint, and hence, this solution did not solve the multi-label classification problem, whereas our work does that.

In [21], a hybrid neural network-sequence alignment search was applied for gene family classification, whereas we use pure ANNs for achieving this. Using classification algorithms over database search helps in improving the speed, as the search time now grows linearly with the functional classes rather than the number of sequence entries. It also helps in improving the sensitivity. The work in [22], divides the proteins in transmembrane and non-transmembrane groups using ANNs and, subsequently, [23] divides the non-transmembrane proteins into 3 further classes. Hence, the eventual classification takes place into 4 different functional classes. Here, we are classifying the whole dataset into 1665.

In [24], the proteins were classified into 4 superfamilies only, and in [25], the classification took place for 10 superfamilies through a neural network, for the proteins belonging to the PIR database. In [26], a binary classification was performed into globin or non-globin classes. In [27], the input was chosen from the Protein Data Bank[28] and was applied to multiple fully-connected multilayer perceptrons(MLPs) for function prediction, with accuracy around 75%. The work of [29, 30] classifies into a maximum of 7 different protein classes. Recently, the authors of [31] used proteins from the PIR database, and classified them into 10 superfamilies with an accuracy of 93.69%.

The work in [32], deployed an ANN-based Gene Ontology functional classification, with AUC less than 90% for one class, and 80% AUC for further 2 classes. Dataset: 30k sequences for Bos taurus and 15k sequences for Gallus gallus. In [33], DNA sequences were classified into a smaller number of classes(less than 10).

The work in [34] trained the neural network on 80% of the sequences of the SwissProt subset of the UniProt Dataset and tested the performance on the remaining 20%, attaining a nearly 100% accuracy, classifying the proteins into only 4 different classes. Filtering: sequence length limited between 10-1000 or 10-2000 as per the classes.

2.2 Loss Function

Let y be the target output vector and $\hat{y}$ be the output vector predicted by the model. The vectors y and $\hat{y}$ both have length $n = 1665$. Let the total number of samples be m. The loss function for multi-label classification if given by

$$L(y, \hat{y}) = \sum_{i=1}^{m} \sum_{j=1}^{n} -y_{ij} \log \hat{y}_{ij} - (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \tag{2.1}$$

The above loss function penalizes the training algorithm if the output is FP or FN. The next section will cover more about these performance metrics.

2.3 Model Performance Evaluation

The analysis of how the model has performs on a multi-label output requires a different approach than binary or multi-class output. The conventional method of calculating *accuracy* fails in multi-label classification when the number of TN is much greater than the sum of TP, FP, FN.

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.2}$$

The accuracy in this case is always close to 100%.This gives the motivation SA. For m samples and n labels,

$$SA = \frac{1}{m} \sum_{i=1}^{m} (y_i = \hat{y}_i) \tag{2.3}$$

Therefore, only exact match of y and $\widehat{y}$ are considered for each sample. The evaluation of model based on *precision* and *recall* can be done by averaging over *samples* or *labels*. For i^{th} sample, let Y_i be the labels of the target and Z_i be the labels of the prediction. Here,

$$Y_i = \{j : y_{ij} = 1, j \in \{1,2,...n\}\}$$
$$Zi = \{j : \widehat{y}_{ij} = 1, j \in \{1,2,...n\}\}$$

The *precision* and *recall* averaged on *samples* is given by,

$$precision(''samples'') = \frac{1}{m}\sum_{i=1}^{m}\frac{|Y_i \cap Z_i|}{|Z_i|} \tag{2.4}$$

$$recall(''samples'') = \frac{1}{m}\sum_{i=1}^{m}\frac{|Y_i \cap Z_i|}{|Y_i|} \tag{2.5}$$

The *HammingLoss* is given by,

$$HammingLoss = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n}\sum_{j=1}^{n}(y_{ij} \oplus \widehat{y}_{ij}) \tag{2.6}$$

Let c_j be the set of labels of all the samples for label j.

$$c_j = \{y_{ij}, i \in \{1,2,...m\}\}$$

TP_j, FP_j, FN_j, TN_j for j^{th} label is calculated. The *precision* and *recall* averaged on *weights*, also called *support*, is given by,

$$s_j = TP_j + FN_j$$

$$precision(''weighted'') = \frac{\sum_{j=1}^{n} s_j \left(\frac{TP_j}{TP_j+FP_j}\right)}{\sum_{j=1}^{n} s_j} \tag{2.7}$$

$$recall(''weighted'') = \frac{\sum_{j=1}^{n} TP_j}{\sum_{j=1}^{n} s_j} \tag{2.8}$$

The *weighted precision* and *recall* takes into account the data imbalance. Along with this the *precision* and *recall* for individual labels is also calculated. The *AUC* is used as an evaluation metric by [1].

Chapter 3

Analysis and Design

3.1 Data Representation

Before designing the neural network architecture for multi-label classification, we need to first understand the genome data. Genome data contains **Gene Ontology** classes, and the sequence of constituent **amino acids** which form the given protein. The amino acid sequence is obtained from DNA using transcription and translation mechanisms by the cell. The naturally occurring proteins generally contain 20 essential amino acids. These are represented by one letter each as follows-

Amino Acid	1-Letter code	Amino Acid	1-Letter code
Alanine	A	Methionine	M
Cysteine	C	Asparagine	N
Aspartic Acid	D	Proline	P
Glutamic Acid	E	Glutamine	Q
Phenylalanine	F	Arginine	R
Glycine	G	Serine	S
Histidine	H	Threonine	T
Isoleucine	I	Valine	V
Lysine	K	Tryptophan	W
Leucine	L	Tyrosine	Y

TABLE 3.1: Essential Amino Acids

The structure and functions of the protein depend on the above amino acids and their properties. These properties are-

- Charge
- Hydrophobicity
- Polarity
- Aromaticity
- Presence of Hydroxyl
- Presence of Sulphur

The amino acids can be encoded mathematically into a one-hot vector of length 20. The purpose of using one-hot encoding is to retain the individuality of each representation, which would otherwise be called into question if the bit-wise representation would be used. This is because batch normalization, which is being used further in the models, calculates mean and variance over the batches. The presence of 6 properties can be represented by a 6 length vector where 1 denotes that property is present, whereas 0 denotes that the property is absent. Therefore for every constituent amino acid in the sequence of amino acids, we have a 26 length vectors. If the input sequence has length l, the input to the model is a matrix of size $26 \times l$. The protein sequences can be associated with 1665 *Gene Ontology* classes. An input sequence can belong to multiple classes (hence, multi-label). The target output vector is encoded as 1665 length vector with 0 for absence and 1 for the presence of the label.

3.2 1-D Convolution Algorithm

Consider a sequence of amino acid

$$SPEYFREGLFSAKS...DVFSFGV$$

The kernel used is also 1 dimensional. The number of initial channels is 26, as each amino acid is encoded as a vector of length 26.

$\boxed{SPEYF}$REGLFSAKS...DVFSFGV
S$\boxed{PEYFR}$EGLFSAKS...DVFSFGV
SP$\boxed{EYFRE}$GLFSAKS...DVFSFGV
SPEYFREGLFSAKS...DV$\boxed{FSFGV}$

TABLE 3.2: 1-D convolution with kernel size 5 stride 1

The 1-D convolution takes into consideration the type of amino acid as well as the six properties which determine the protein structure while doing the convolution operation. The effect produced by certain constituent amino acid only lasts in its neighborhood and is not observed beyond it. The choice of the kernel size is hence used to restrict the learning only to the neighborhood. The information extracted by this type of convolution operation effectively calculates the aggregated effect of groups of amino acids. With each pass through a convolution layer, the length of the sequence decreases depending upon the kernel size and stride. Therefore, in this process, complex information on neighborhood influence can be learned. This is then passed on to the fully connected layers of the neural network.

Chapter 4

Experiments and Results

4.1 Datasets

4.1.1 Data Extraction

The SwissProt subset of the UniProt database[18] was used after being acquired from http://uniprot.org as starting point (using the query "goa:(*) AND reviewed:yes"), containing 535,119 sequences having Gene Ontology IDs at the date of download of 5 August 2019.
The raw data acquired from UniProt has been stated as follows-

Sequence Samples : 535119
Label Samples : 535119
Max Sequence Length : 35213
Min Sequence Length : 2
Max Label Length : 258
Min Label Length : 1
Total Labels : 2970815
Total Unique Labels : 28234

Clearly, the amount of data was large enough to become a challenge for any kind of deep neural networks. Hence, we had to filter and pre-process the data suitably. After the raw data was ready, it was shuffled and split into train and test data, with the test data containing 5000 entries, and the rest of the data being used for training purposes.

4.1.2 Data Pre-Processing

After obtaining the raw data, pre-processing had to be done, for which some filtering measures were applied.
The filtering criteria for the train data were -

- Constraining the sequence length between 162 and 2000. This was largely determined by the available video memory on our GPU(Nvidia GP102 TITAN Xp 12 GB GPU). The lower limit was set to 162 so that the output of the last pooling layer was at least one amino acid.
- The starting 'M' (Methionine) character was removed from all the sequences.
- Once, protein sequences were filtered, the corresponding labels were filtered such that every label had at least one protein sequence belonging to it.

The filtering criteria for the test set were -

- Only sequences with length greater than 162 were considered, and longer sequences were cropped to a max length of 2000.

4.2 Experimental Setup

All the experiments have been conducted on Intel(R) Xeon(R) Gold 6142 CPU @ 2.60GHz, 196 GB RAM, Nvidia GP102 TITAN Xp 12 GB GPU. Python v3.6.8 is used. The framework used is PyTorch v1.2.0. The framework takes advantage of CUDA parallel computing platform by Nvidia. CUDA compilation tools v9.1.85 are used. Supporting libraries like torchvision(0.4.0), numpy(1.16.5), sklearn are also required.

4.2.1 Model Architecture

The initial model architecture [1] was -

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
spatial pyramid pool (levels=3, divs per level=4)
fully connected 1 (units=1024, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.1: model 0

The Adam optimization algorithm was used for training the model.

4.2.2 Experimentation with model architecture

Experimental changes in the models were as follows -

- model 1: In each layer, the number of output channels is reduced by half. The number of neurons in fully connected 1 layer is increased to 5000 units. Spatial pyramid pooling layer is removed.

conv (size=6, stride=1, depth=64, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=64, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
fully connected 1 (units=5000, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.2: model 1

- model 2: Batch norm layer is removed from model 0.

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
spatial pyramid pool (levels=3, divs per level=4)
fully connected 1 (units=1024, activation=prelu)
dropout (p=0.5)
fully connected 2 (units=1665)

TABLE 4.3: model 2

- model 3: *relu* is used as activation function instead of *prelu* in model 0.

conv (size=6, stride=1, depth=128, padding=VALID, activation=relu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
spatial pyramid pool (levels=3, divs per level=4)
fully connected 1 (units=1024, activation=relu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.4: model 3

- model 4: *leaky_relu* is used as activation function instead of *prelu* in model 0.

conv (size=6, stride=1, depth=128, padding=VALID, activation=leaky_relu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=leaky_relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=leaky_relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=leaky_relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=leaky_relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=leaky_relu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
spatial pyramid pool (levels=3, divs per level=4)
fully connected 1 (units=1024, activation=leaky_relu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.5: model 4

- model 5: In model 0, the number of neurons in fully connected 1 layer is increased to 4096 units. 1 fully connected layer with 2048 units and prelu activation function is added after fully connected 1 along with dropout 0.5 and batch norm.

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
spatial pyramid pool (levels=3, divs per level=4)
fully connected 1 (units=4096, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=2048, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 3 (units=1665)

TABLE 4.6: model 5

- model 6: 1 fully connected layer with 2048 units and prelu activation function is added after fully connected 1 along with dropout 0.5 and batch norm in model 1.

conv (size=6, stride=1, depth=64, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=64, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
fully connected 1 (units=5000, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=2048, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 3 (units=1665)

TABLE 4.7: model 6

- model 7: model 1 with number of output channels in each layer as model 0.

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
fully connected 1 (units=5000, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.8: model 7

- model 8: 1 fully connected layer with 2048 units and prelu activation function is added after fully connected 1 along with dropout 0.5 and batch norm in model 7.

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
fully connected 1 (units=5000, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 2 (units=2048, activation=prelu)
dropout (p=0.5)
batch norm (scale=True)
fully connected 3 (units=1665)

TABLE 4.9: model 8

- model 9: In model 0, the number of neurons in fully connected 1 layer is increased to 4096 units. Dropout is changed to 0.25.

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
spatial pyramid pool (levels=3, divs per level=4)
fully connected 1 (units=4096, activation=prelu)
dropout (p=0.25)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.10: model 9

- model 10: In model 7, the dropout is changed to 0.25.

conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
batch norm (scale=False)
conv (size=6, stride=1, depth=128, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=256, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
conv (size=5, stride=1, depth=512, padding=VALID, activation=prelu)
max pool (size=2, stride=2, padding=VALID)
batch norm (scale=False)
fully connected 1 (units=5000, activation=prelu)
dropout (p=0.25)
batch norm (scale=True)
fully connected 2 (units=1665)

TABLE 4.11: model 10

The results of each of the models are given below.

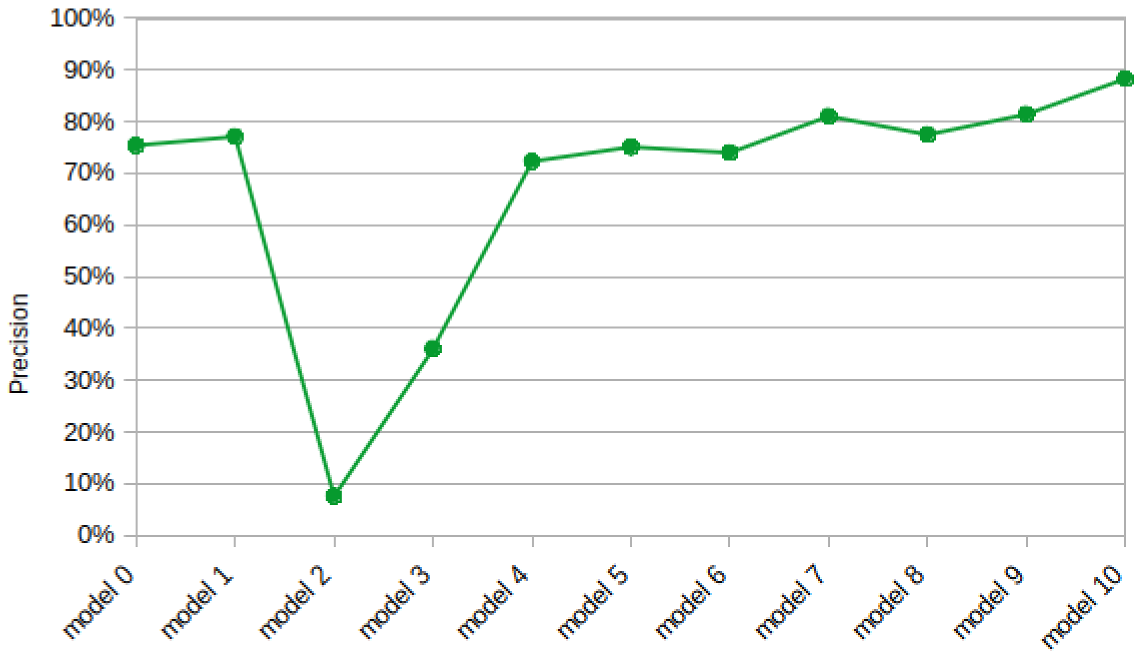

FIGURE 4.1: Precision of models

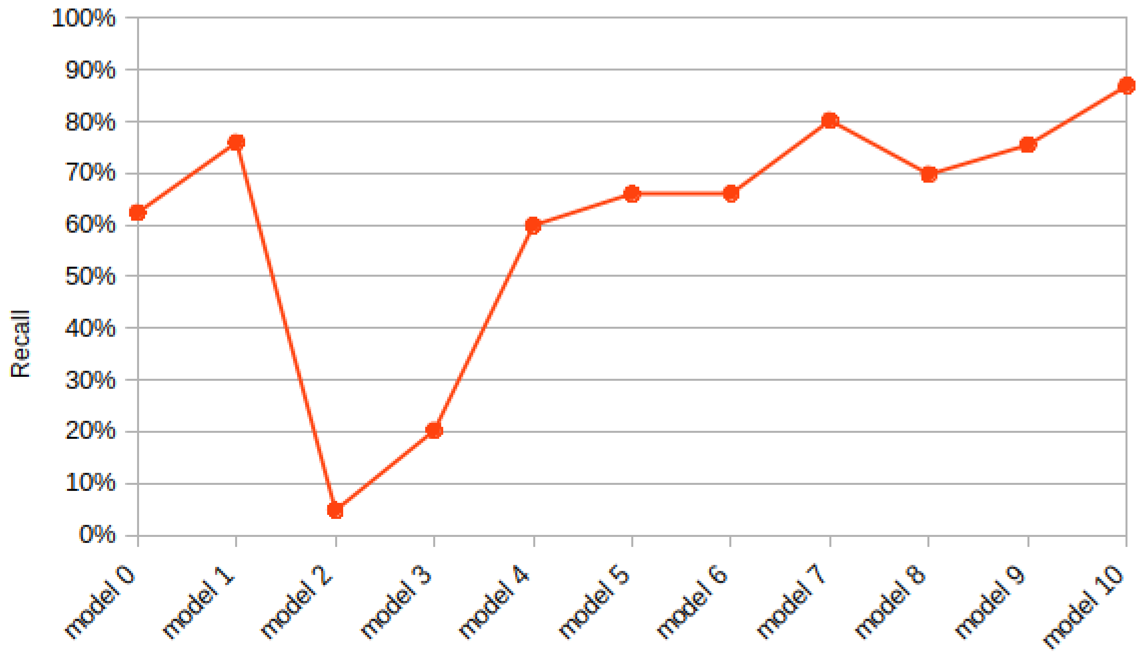

FIGURE 4.2: Recall of models

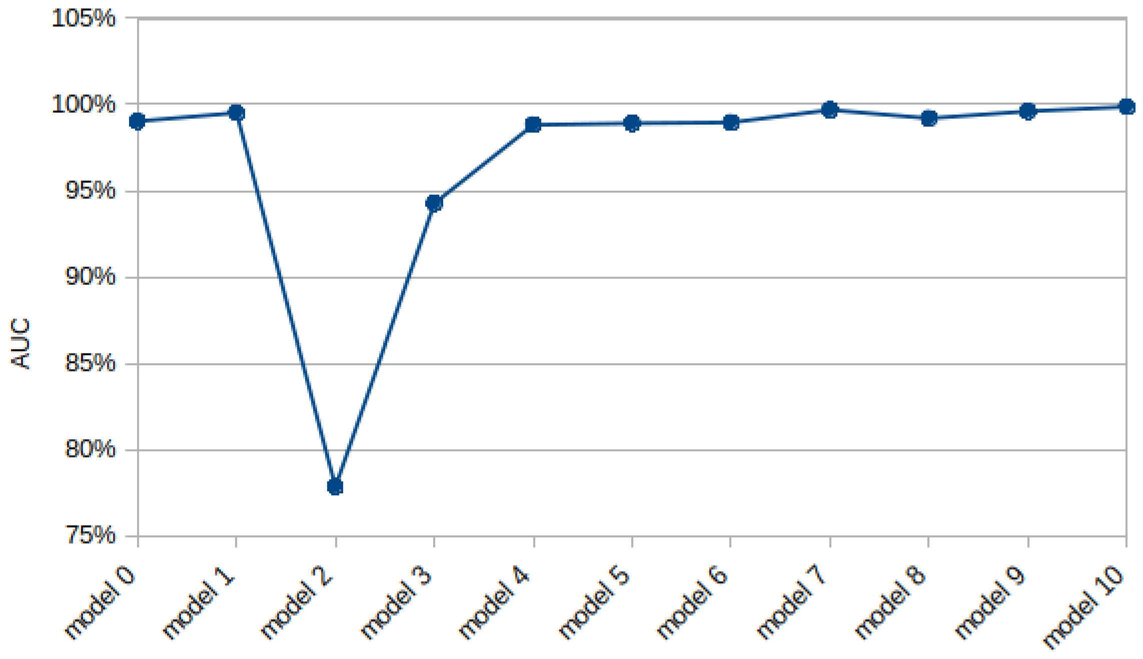

FIGURE 4.3: AUC of models

Therefore, model 10 gives the best results.

4.3 Results and Discussion:

4.3.1 Loss function with weights

The frequency of label was calculated for the entire data set. It was found that the distribution of labels is skewed.

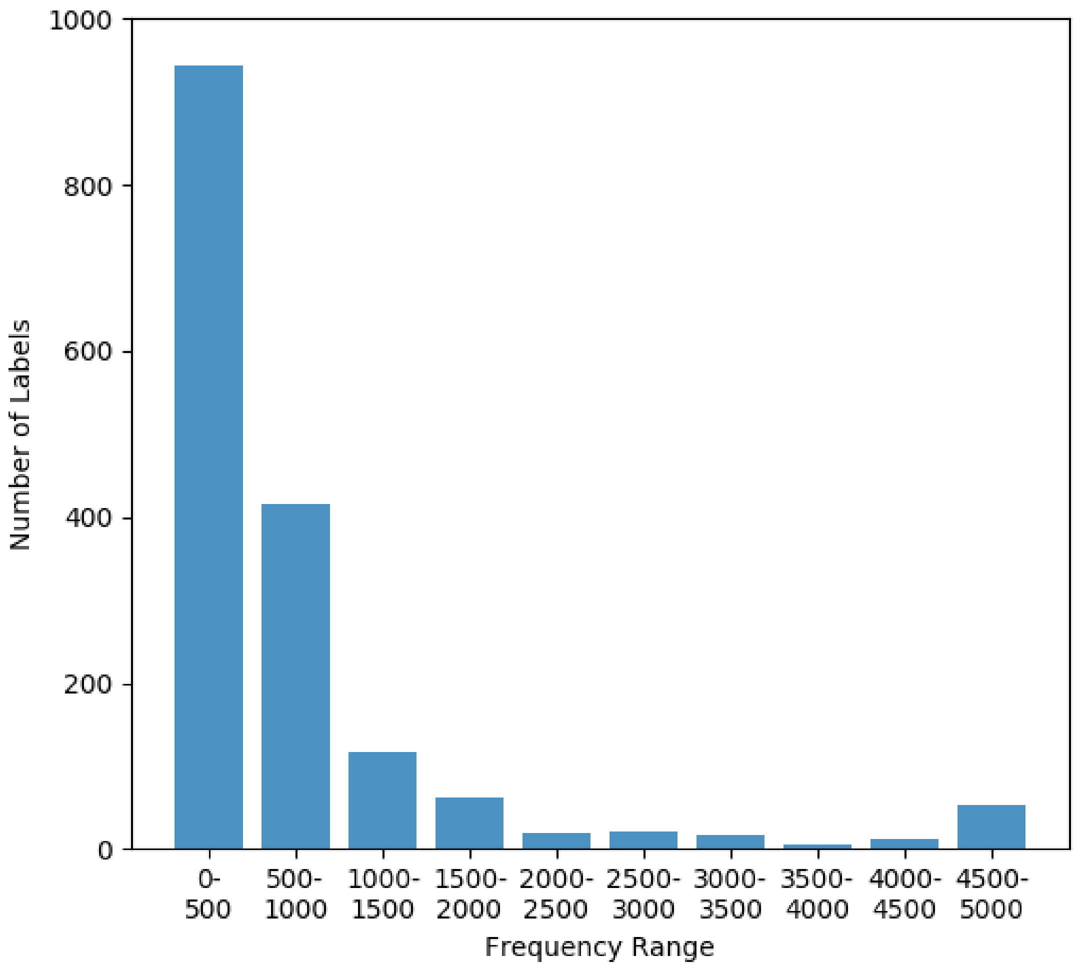

FIGURE 4.4: Label Frequency Distribution

For the labels with less frequency, learning is difficult. The solution to this problem was to do over-sampling of labels with lower frequency or under-sampling of labels with higher frequency or adding weights in the loss function. As the problem was multi-label classification, over-sampling a label occurring few times can also lead to over-sampling of a frequently occurring label. A similar case could be considered for under-sampling. Hence, introducing weights associated with individual labels in the loss function is a better approach. The loss function with weights is given by-

$$L(y, \hat{y}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_j \left[-y_{ij} \log \hat{y}_{ij} - (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \right] \tag{4.1}$$

The mean label frequency is 1100. If training was done without weights ($w_j = 1.0$) in the loss function, it was observed that *precision* and *recall* was poor for label frequencies smaller than mean frequency.

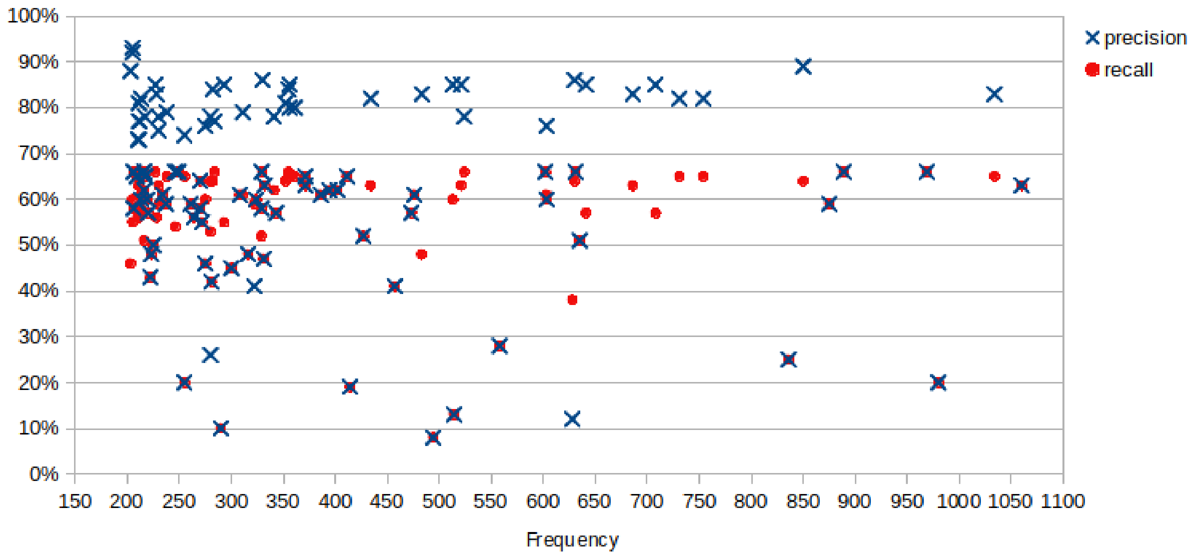

FIGURE 4.5: Precision and Recall vs Label Frequency

Hence the weights for labels with frequency less than the mean frequency need to be increased. Let s_j be the label frequency of the j^{th} label. The weight w_j is given by

$$w_j = \max\left\{1, \min\left\{\frac{mean\,(s)}{s_j}, 5\right\}\right\} \tag{4.2}$$

where, $mean\,(s)$ is *Mean Label Frequency*. Here, we limit the weights to be in the range of $[1, 5]$ for practical purposes.

4.3.2 Heuristics to improve Precision and Recall

The training, according to the model specified in [1], resulted in good overall precision and recall, but the precision and recall for individual labels were still quite poor. Calculating the subset accuracy according to the model proposed in [1] to be around 44% points to improving the individual precision and recall for the labels. As mentioned in the previous section, improving the individual precision and recall of the labels can be achieved through oversampling the labels with lower frequency, under-sampling of the labels with high frequency or modifying the weights of the loss function so as to penalize the error made in learning.

The heuristic approach applied to improve on the precision and recall for the individual labels relies largely upon the fact that modifying the weights of the loss function for each instance influences the learning process. The idea was to import the weights from a model performing better than the model at hand.

Following were the experiments performed, depicting the approach -

Consider the model(Model 10) with architecture and weights according to [1]. On training, it achieves subset accuracy of 44.46% with 225 labels with precision lesser than 67%, and 58 labels with recall lesser than 67%(a benchmark set for reference purposes). Another model(Model 11), was trained with no weights in the loss function, and achieved subset accuracy of 50.85%, with 94 labels with precision lesser than 67%, and 113 labels with recall lesser than 67%. Note that no change in architecture was performed during this process.

- In Phase-I of crossing the weights, the individual recall could be improved in Model 11, and individual precision in Model 10. Hence, the weights of Model 10 for the corresponding labels(labels whose recall for Model 11 is lesser than 67%) were assigned to the same labels in Model 11, represented in Model 12. Similarly, weights of Model 11 were assigned to the same labels in Model 10, for labels whose precision for Model 10 was lesser than 67%, represented in Model 13.
  Results for Model 12 -
  Subset Accuracy : 48.20%, Number of labels with Precision lesser than 67% : 106, Number of labels with Recall lesser than 67% : 107
  Results for Model 13 -
  Subset Accuracy : 49.44%, Number of labels with Precision lesser than 67% : 173, Number of labels with Recall lesser than 67% : 58.
- In Phase-II, the same process was performed between Model 11 and Model 13(because of higher subset accuracy), resulting in Model 14(improving recall in Model 11) and Model 15(improving precision in Model 13).
  Results for Model 14 -
  Subset Accuracy : 50.16%, Number of labels with Precision lesser than 67% : 112, Number of labels with Recall lesser than 67% : 64
  Results for Model 15 -
  Subset Accuracy : 49.31%.Number of labels with Precision lesser than 67% : 137, Number of labels with Recall lesser than 67% : 85
- In the final Phase, the Model 11 and Model 14 were considered. Result for final Model 16 -
  Subset Accuracy : 55.43%, Number of labels with Precision lesser than 67% : 81, Number of labels with Recall lesser than 67% : 87.

The heuristic approach to the improvement of the individual precision and recall is formalized as follows:

- Take 2 models, with the best subset accuracy available, say A and B.
- Calculate the number of labels for each with Precision lesser than 67% and Recall lesser than 67%.
- If either of A or B, has a better of both precision and recall, then stop.
- Else if say, A has greater precision and B has a greater recall, then -

- Sort the labels of A according to recall, and B according to precision.
- Substitute B's weights in A for labels with recall lesser than 67%, and substitute A's weights in B for labels with precision lesser than 67%.
- This results in two new models, with different weights, which can be trained further. Find their subset accuracy, and return to the starting step.

This can be continued till a satisfactory final model has been reached, with an improved subset accuracy.

	Precision	Recall	AUC	SA
Final Model	92.37%	90.07%	99.94%	55.43%

TABLE 4.12: Results after applying the above heuristic approach

Results of performing the heuristic approach has been visualized as follows, by the plots of Precision and Recall of individual label samples for the parent models and the child model.

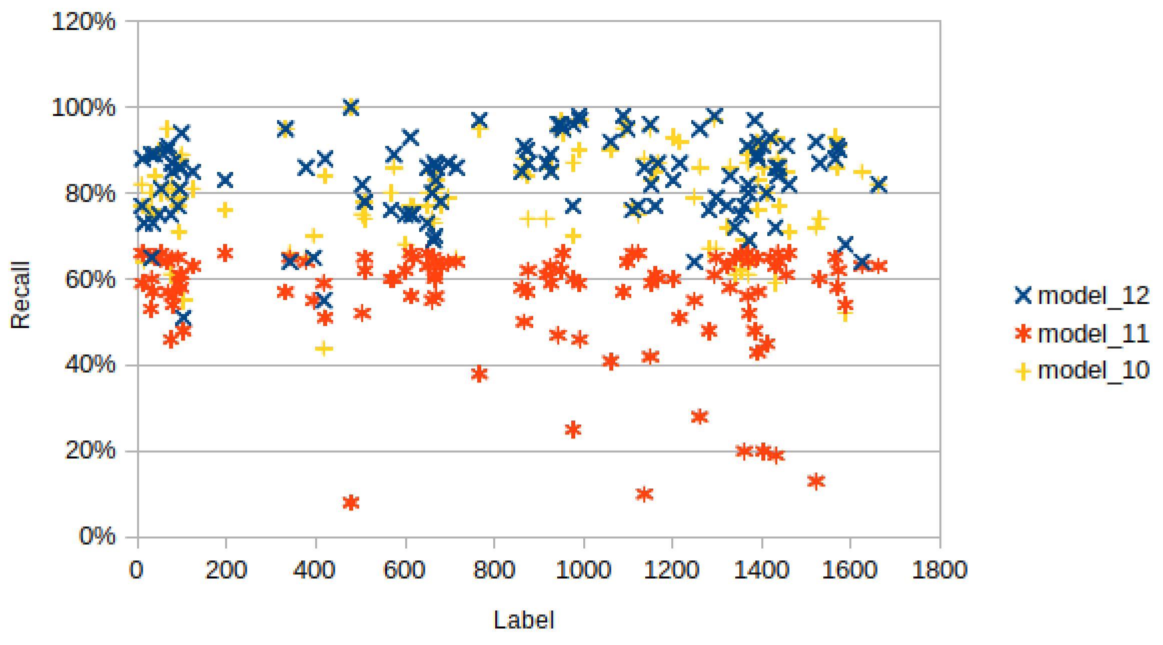

FIGURE 4.6: Plot of Recall for individual Labels for Models - 10,11 and 12. Model 12 obtained from substituting weights of Model 10 in Model 11.

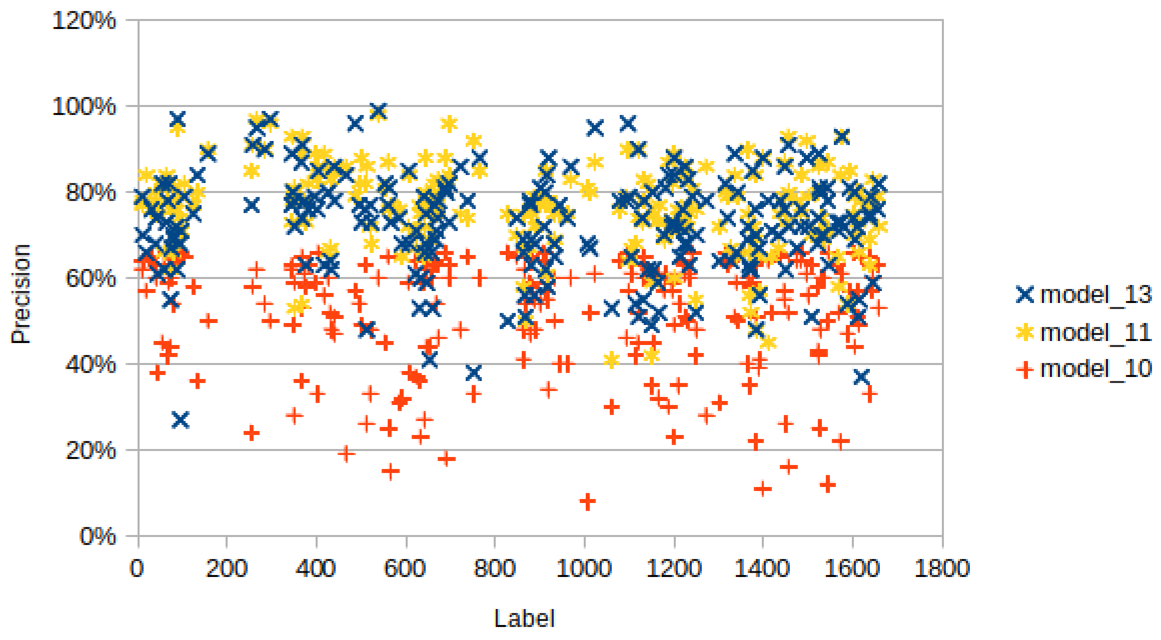

FIGURE 4.7: Plot of Precision for individual Labels for Models - 10,11 and 13. Model 13 obtained from substituting weights of Model 11 in Model 10.

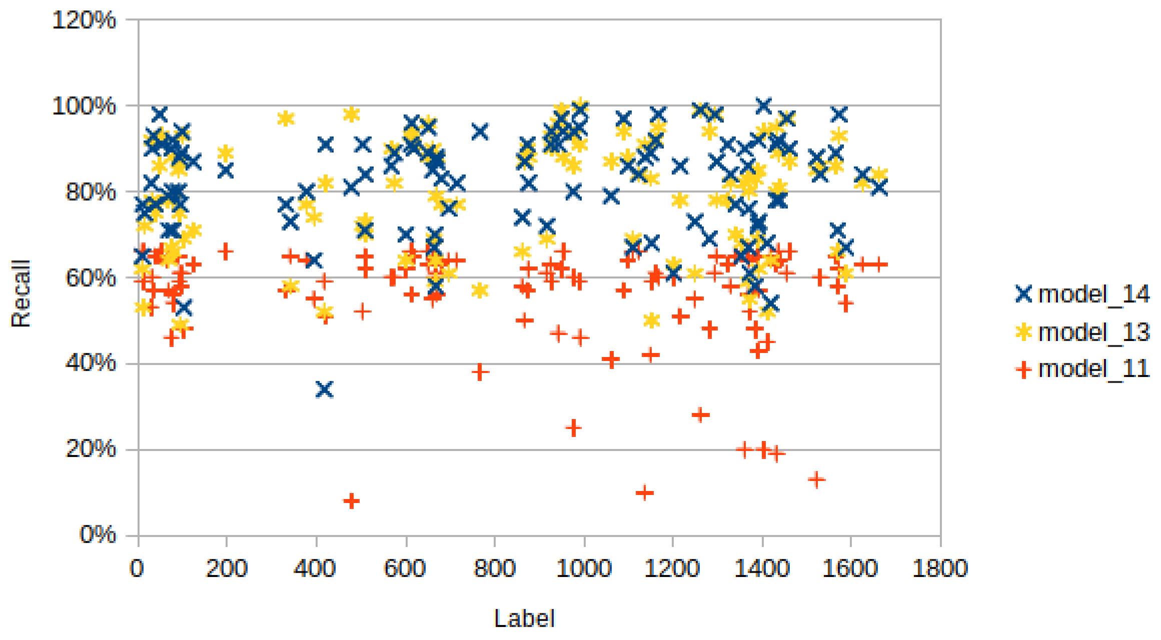

FIGURE 4.8: Plot of Recall for individual Labels for Models - 11,13 and 14. Model 14 obtained from substituting weights of Model 13 in Model 11.

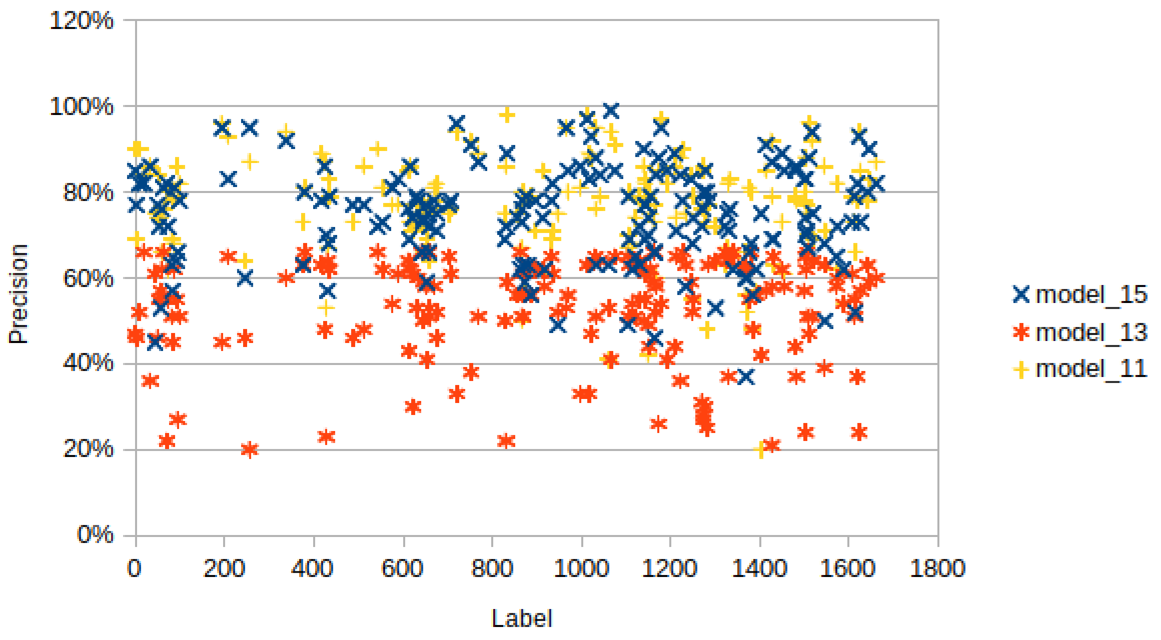

FIGURE 4.9: Plot of Precision for individual Labels for Models - 11,13 and 15. Model 15 obtained from substituting weights of Model 11 in Model 13.

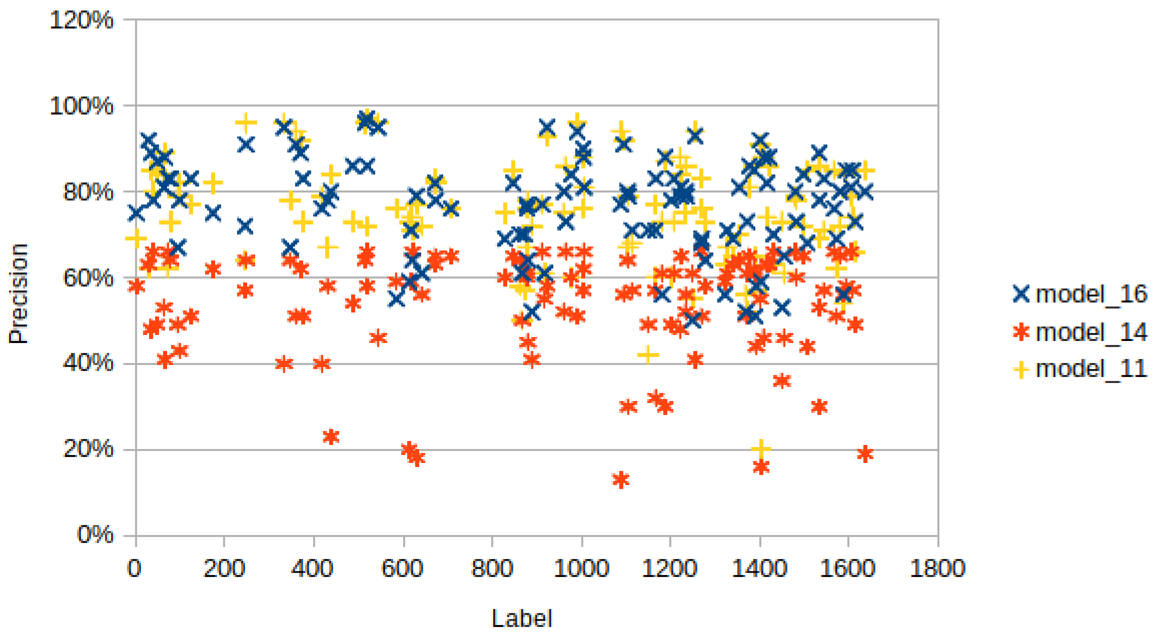

FIGURE 4.10: Plot of Precision for individual Labels for Models - 11,14 and 16. Model 16 obtained from substituting weights of Model 11 in Model 14.

As can be concluded from the above graphs, performing the substitution of weights in the models, leads to improvement in Precision and Recall for the individual label samples.

4.3.3 Summary of the results

Model	Precision	Recall	AUC	SA
0	75.39%	62.38%	99.03%	29.88%
1	77.09%	75.97%	99.52%	29.02%
2	7.52%	4.75%	77.84%	0.01%
3	36.07%	20.24%	94.27%	3.64%
4	72.28%	59.92%	98.82%	27.23%
5	75.10%	66.01%	98.91%	30.67%
6	73.97%	66.09%	98.96%	29.96%
7	81.04%	80.23%	99.69%	34.57%
8	77.47%	69.80%	99.20%	33.36%
9	81.42%	75.50%	99.61%	33.37%
10	88.30%	86.97%	99.87%	44.46%
11	90.73%	89.23%	99.92%	50.86%
12	90.52%	87.80%	99.92%	48.20%
13	89.46%	89.79%	99.93%	49.44%
14	90.00%	89.56%	99.93%	50.17%
15	89.72%	88.68%	99.92%	49.31%
16	92.37%	90.07%	99.94%	55.43%

TABLE 4.13: Summary of the results

Chapter 5

Conclusion and Future Work

The objectives of this project were to

- Process the data obtained from UniProt [18], and make it suitable to be applied to various neural network models.
- Propose variations to the model used by [1], in order to improve individual precision and recall of the label samples.
- Evaluate the performance of the variant models against existing methods, providing a heuristic approach.
- Showcase the statistical significance of our results.

In this project, we evaluated the SwissProt subset of the UniProt database[18] and tried to incorporate the skewness of the label distribution into the learning process. Previously mentioned methods were performing sequence alignment based similarity search or performing multi-label classification for very few classes. The work in [1], performed the multi-label classification on the same subset of UniProt dataset [18] and considered 983 Gene Ontology classes, where we increased the number of Gene Ontology classes in contention to 1665, meanwhile also improving on the individual precision and recall of the label samples, and hence, increasing the overall subset accuracy from 44% to 55.43%.

We performed various modifications to models proposed in previously mentioned research work by varying the number of neurons in the fully connected layers, changing the activation function, and dropout. Over this, we modified the weights in the loss function for individual labels by applying a heuristic approach to penalize the error in learning for the less frequent labels, which eventually led to an increase in the overall subset accuracy too. The results are discussed through various parameters and performance metrics such as Precision, Recall, Hamming Loss, F1-score and AUC.

	Precision	Recall	AUC	SA
Final Model	92.37%	90.07%	99.94%	55.43%

TABLE 5.1: Results

The future work in this direction could be to extend the heuristic approach into a formal evolutionary method, on improving the individual precision and recall of the label samples, while also modifying the model architecture to a hybrid sequence matching and ANN based structure.

Bibliography

[1] Balázs Szalkai and Vince Grolmusz. “Near Perfect Protein Multi-Label Classification with Deep Neural Networks”. In: *Methods* 132 (Apr. 2017). DOI: 10.1016/j.ymeth.2017.06.034.

[2] T. F. Smith and M. S. Waterman. “Identification of common molecular subsequences”. In: *Journal of Molecular Biology*. Vol. 147(1). Cargse, France., 1981, pp. 195–197.

[3] Gabor Ivan, Daniel Banky, and Vince Grolmusz. “Fast and Exact Sequence Alignment with the Smith-Waterman Algorithm: The SwissAlign Webserver”. In: *Gene Reports* 4 (Sept. 2013). DOI: 10.1016/j.genrep.2016.02.004.

[4] Stephen Altschul et al. “Basic Local Aligment Search Tool”. In: *Journal of molecular biology* 215 (Nov. 1990), pp. 403–10. DOI: 10.1016/S0022-2836(05)80360-2.

[5] Sean Eddy. “A new generation of homology search tools based on probabilistic inference”. In: *Genome informatics. International Conference on Genome Informatics* 23 (Oct. 2009), pp. 205–11. DOI: 10.1142/9781848165632_0019.

[6] Sean R. Eddy. “Accelerated Profile HMM Searches”. In: *PLOS Computational Biology* 7.10 (Oct. 2011), pp. 1–16. DOI: 10.1371/journal.pcbi.1002195. URL: https://doi.org/10.1371/journal.pcbi.1002195.

[7] Balázs Szalkai et al. “The Metagenomic Telescope”. In: *PLOS ONE* 9.7 (July 2014), pp. 1–9. DOI: 10.1371/journal.pone.0101605. URL: https://doi.org/10.1371/journal.pone.0101605.

[8] Kristoffer Illergård, David Ardell, and Arne Elofsson. “Structure is three to ten times more conserved than sequence-A study of structural response in protein cores”. In: *Proteins* 77 (Nov. 2009), pp. 499–508. DOI: 10.1002/prot.22458.

[9] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[10] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 346–361. ISBN: 978-3-319-10578-9.

[11] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *IEEE International Conference on Computer Vision (ICCV 2015)* 1502 (Feb. 2015). DOI: 10.1109/ICCV.2015.123.

[12] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: (Feb. 2015).

[13] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[14] Shun ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4 (1993), pp. 185 –196. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(93)90006-O. URL: http://www.sciencedirect.com/science/article/pii/092523129390006O.

[15] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[16] Yann Dauphin, Harm de Vries, and Yoshua Bengio. "Equilibrated adaptive learning rates for non-convex optimization". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 1504–1512. URL: http://papers.nips.cc/paper/5870-equilibrated-adaptive-learning-rates-for-non-convex-optimization.pdf.

[17] Yann Dauphin et al. "Equilibrated adaptive learning rates for non-convex optimization". In: *NIPS*. 2015.

[18] Amos Bairoch et al. "The Universal Protein Resource (UniProt) 2009". In: *Nucleic Acids Research* 37 (Jan. 2009). DOI: 10.1093/nar/gkn664.

[19] Edgardo A. Ferrán, Pascual Ferrara, and Bernard Pflugfelder. "Protein Classification Using Neural Networks". In: *Proceedings. International Conference on Intelligent Systems for Molecular Biology* 1 (1993), pp. 127–35.

[20] Cathy Wu et al. "Neural networks for molecular sequence classification". In: *Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology* 1 (Feb. 1993), pp. 429–37.

[21] Cathy Wu and S. Shivakumar. "Gene family identification network design". In: June 1998, pp. 103 –110. ISBN: 0-8186-8548-4. DOI: 10.1109/IJSIS.1998.685426.

[22] Claude Pasquier and Stavros Hamodrakas. "An hierarchical neural network system for the classification of transmembrane proteins". In: *Protein engineering* 12 (Sept. 1999), pp. 631–4. DOI: 10.1093/protein/12.8.631.

[23] Claude Pasquier, Vasilis Promponas, and Stavros Hamodrakas. "PRED-CLASS: Cascading Neural networks for generalized protein classification and genome wide applications". In: *Proteins* 44 (Aug. 2001), pp. 361–9. DOI: 10.1002/prot.1101.

[24] Jason T. L. Wang et al. "Application of Neural Networks to Biological Data Mining: A Case Study in Protein Sequence Classification". In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Boston, Massachusetts, USA: ACM, 2000, pp. 305–309. ISBN: 1-58113-233-6. DOI: 10.1145/347090.347157. URL: http://doi.acm.org/10.1145/347090.347157.

[25] Dianhui Wang et al. "Protein Sequences Classification Using Modular RBF Neural Networks". In: Dec. 2002, pp. 477–486. DOI: 10.1007/3-540-36187-1_42.

[26] Jinmiao Chen and Narendra Chaudhari. “Protein Family Classification Using Second-Order Recurrent Neural Networks”. In: *Genome Informatics* 14 (Jan. 2003), pp. 520–521.

[27] Wagner Rodrigo Weinert and Heitor Silvério Lopes. “Neural networks for protein classification”. In: *Applied Bioinformatics* 3.1 (2004), pp. 41–48. ISSN: 1175-5636. DOI: 10.2165/00822942-200403010-00006. URL: https://doi.org/10.2165/00822942-200403010-00006.

[28] Helen M. Berman et al. “The Protein Data Bank”. In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 235–242. ISSN: 0305-1048. DOI: 10.1093/nar/28.1.235. eprint: http://oup.prod.sis.lan/nar/article-pdf/28/1/235/9895144/280235.pdf. URL: https://doi.org/10.1093/nar/28.1.235.

[29] Konstantinos Blekas, Dimitrios I. Fotiadis, and Aristidis Likas. “Protein Sequence Classification Using Probabilistic Motifs and Neural Networks”. In: *Artificial Neural Networks and Neural Information Processing — ICANN/ICONIP 2003*. Ed. by Okyay Kaynak et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 702–709. ISBN: 978-3-540-44989-8.

[30] Konstantinos Blekas, Dimitrios Fotiadis, and Aristidis Likas. “Motif-Based Protein Sequence Classification Using Neural Networks”. In: *Journal of computational biology : a journal of computational molecular cell biology* 12 (Feb. 2005), pp. 64–82. DOI: 10.1089/cmb.2005.12.64.

[31] Jiuwen Cao and Lianglin Xiong. “Protein Sequence Classification with Improved Extreme Learning Machine Algorithms”. In: *BioMed research international* 2014 (Mar. 2014), p. 103054. DOI: 10.1155/2014/103054.

[32] Davide Chicco, Peter Sadowski, and Pierre Baldi. “Deep autoencoder neural networks for gene ontology annotation predictions”. In: *ACM BCB 2014 - 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics* (Sept. 2014), pp. 533–540. DOI: 10.1145/2649387.2649442.

[33] Nguyen Ngoc Giang et al. “DNA Sequence Classification by Convolutional Neural Network”. In: *Journal of Biomedical Science and Engineering* 09 (Jan. 2016), pp. 280–286. DOI: 10.4236/jbise.2016.95021.

[34] Xueliang Liu. “Deep Recurrent Neural Network for Protein Function Prediction from Sequence”. In: *ArXiv* abs/1701.08318 (2017).