TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A

PROJECT REPORT

ON

PREDICTION OF LYSINE SUCCINYLATION SITE USING PROTEIN LANGUAGE MODEL

SUBMITTED BY:

ANANDA THAKUR (PUL075BEI007)

PRADIPTI SIMKHADA(PUL075BEI021

SHAMBHAVI ADHIKARI(PUL075BEI034)

SUBIN MAHARJAN(PUL075BEI040)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

March, 2023

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **"Prediction of Lysine Succinylation Site using Protein Language Model"** submitted by **Ananda Thakur**, **Pradipti Simkhada**, **Shambhavi Adhikari**, **Subin Maharjan** in partial fulfillment of the requirements for the Bachelor's degree in Electronics Communication and Information Technology Engineering.

.....
Supervisor
Lokhnath Regmi
Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

.....
Internal examiner
.....
Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

.....
External examiner
.....
Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

Date of approval:

Copyright

Acknowledgement

We would like to express our sincere gratitude to **Asst. Prof Lokhnath Regmi**, our project supervisor, for his constant guidance, inspiring lectures, and precious encouragement. Without his invaluable supervision and suggestions, this project would have been a difficult journey for us. His useful suggestions for this whole work and cooperative behavior are sincerely acknowledged.

We would like to earnestly acknowledge the support, guidance and valuable time given by **Mr. Suresh Pokhrel** and **Asst. Prof Dr. Dhukka KC**.

We are also very grateful to **Department of Electronics and Computer Engineering, Pulchowk Campus** for providing us with such a splendid opportunity to do the project

We are thankful for and fortunate enough to get constant encouragement, from all respected teachers namely **Dr. Nanda Bikram Adhikari**, **Dr. Dibakar Raj Pantha**, **Dr. Surendra Shrestha**, **Prof. Dr. Ram Krishna Maharjan**, **Prof. Dr. Subarna Shakya**, **Assist. Prof. Nischal Acharya**, **Asst. Prof. Dr. Basanta Joshi** and all our friends, directly and indirectly, have lent their helping hand and suggestions in this venture.

Last but not least, we express our deep appreciation to our family members who have been a constant source of inspiration for us. We welcome any kind of suggestion or criticism, and we highly appreciate and acknowledge all feedback received.

Ananda Thakur (075BEI007)
Pradipti Simkhada (075BEI021)
Shambhavi Adhikari (075BEI034)
Subin Maharjan (075BEI040)

Abstract

Lysine succinylation is an important post-translational modification(PTM) that controls protein shape, function, and physiochemical properties and has an effect on metabolic processes, the incidence of many diseases, and their progression. Several experimental and computational approaches have been proposed. This method uses a protein Language Model (pLMs) to extract features from protein sequences and convolution neural network (CNN) with artificial neural networks to predict succinylation.

This method used two protein Language Models which are ProtBert and ProtT5. The protein sequences are fed to the model to develop a different set of protein embeddings. The embeddings from different pLMs were found to have negligible similarity. The embeddings are fed to 1D-CNN Neural networks and the outputs from the networks are stacked and ANN is trained on top of that. The stacking ensemble has improved the performance of our proposed architecture. On comparison using benchmarking dataset, our method was comparable with other state of the art models on nearly every metric.

Keywords: Post-translational modification, Lysine Succinylation, Protein Language Model, ProtBert, ProtT5, Protein Embeddings

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AUC	Area Under the Curve
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
BOW	Bag-of-Words
CNN	Convolutional Neural Network
COVID-19	Coronavirus Disease 2019
DNN	Deep Neural Network
GPUT4	Graphics Processing Unit Tesla T4
KNNC	k-Nearest Neighbor Classifier
LMs	Language Models
LSTM	Long Short-Term Memory
MCC	Matthews Correlation Coefficient
MHA	Multi-Head Attention
ML	Machine Learning
MLMSS	Meta LM Succinylation Site
NLP	Natural Language Processing
PCA	Principal Component Analysis
PR AUC	Precision-Recall Area Under the Curve
ProtBERT	Protein Bidirectional Encoder Representations from Transformers
ProtT5	Protein Transformer-5
PSSM	Position-Specific Scoring Matrix
ReLU	Rectified Linear Unit
RF	Random Forest
ROC	Receiver Operating Characteristic
SARSCoV-2	Severe Acute Respiratory Syndrome Coronavirus 2
SGD	Stochastic Gradient Descent
t-SNE	t-Distributed Stochastic Neighbor Embedding

Amino Acid Representation

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

Table 1: Animo Acid Representation

Note: 'X' is used as pseudo amino acid while padding

1 Introduction

1.1 Background

Post-translational modification(PTM) of proteins is the modification of individual amino-acid residues after mRNA is translated into proteins. There are many known PTMs and some are very rare in nature. Due to its importance in several bio-molecular pathways, PTM is an essential mechanism for regulating the function of a protein, which plays an irreplaceable role in biological processes and signal paths, and reversibly determines cell dynamics and plasticity. So, the identification of PTM is an important problem.

Lysine succinylation is a PTM where succinyl donor (- CO - CH2 - CH2 - CO -) is transferred to a lysine residue of a protein molecule. Succinylation controls the nervous system's protein metabolism and function, which is crucial in the development of neurological disorders[1] SARS-CoV-2 has been demonstrated to cause a rise in succinylation in the early stages of infection, indicating that succinylation inhibitors may be used to treat COVID-19 by reducing viral replication[2]. It controls protein shape, function, and physiochemical properties, and has an effect on metabolic processes, the incidence of many diseases, and their progression. Hence, the discovery of lysine succinylation sites in proteins aids in the comprehension of cell physiology, which in turn aids in biomedical research and the creation of new drugs.

Several methods have been developed for predicting lysine succinylation. The experimental methods are highly accurate but extremely time-consuming and expensive. Most of the computational methods used for predicting succinylation use either manual extraction of features from the protein sequence or word embeddings generated from methods like one hot encoding, a bag of words, or others. The problem is that they suffer from likelihood of bias, are unable to account for unknown factors, and dependency on data.

Recent developments have made language models (LMs) an important method for learning embeddings directly from big, unlabeled natural language datasets. The same techniques used by language models are being transported to proteins to develop protein language models (pLMs). The pLMs are trained on large sequence of proteins and are able to extract features from protein sequences which can be then used for predicting succinylation. pLMs enable transfer learning.

1.2 Problem Statement

Identification of lysine succinylation sites in protein is helpful in understanding cell physiology which supports biomedical research and drug development.

The prediction can be done by using various experimental techniques like high-performance liquid chromatography assays, mass spectrometry which are time-consuming and expensive. The computational methods developed so far suffers from dependency on the data, likelihood of bias, unable to account for unknown factors. Due to the limitations, the problem is to use protein language model to generate protein embedding and combine that with deep learning to predict lysine succinylation sites.

1.3 Objective

The major objectives that we are trying to achieve are listed below:

- To use protein language model and deep learning methods to identify Lysine succinylation sites in a protein sequence.
- To compare different protein language model to access their performance in predicting succinylation.
- To combine language models to observe if the performance of the system increases.
- To compare with other methods employing different techniques based on different accuracy metrics.

1.4 Scope

As discussed in the objective section, the primary scope of the project is to use protein language model and deep learning methods to identify lysine succinylation sites in a protein sequence. The project also aims to compare performance of different protein language model and combine them to see if the performance of the system increases and finally compare the results with other methods employing other techniques.

1.5 Limitations

There were several limitations found on our project. While testing on the benchmarking dataset with class imbalance, it was found that the performance of the model decreased. The model was

not able to generalize well on the negative sequences of benchmarking dataset. The size of the dataset was small which might have hindered the performance and with the increase in data, the performance can increase. t-SNE visualization showed that the boundary of separation between succinylation and non-succinylation sites for the model was slightly overlapping and could be better. It means the model was not able to classify succinylated sites and non-succinylated sites with good confidence. The method was quite computationally expensive as it required extensive use of high-performance GPU for feature extraction using the very large language models with billions of parameter.

1.6 Overview of the Report

Our report has been carefully divided into several chapters that are connected to one another logically. First, we began with the introduction of the project explaining the background, history and application. This follows with problem statement, objectives and scope of the project.

Following the introduction, we discussed past works done on the subject in the literature review. This section covers a variety of conventional, machine learning, and deep learning methods for predicting succinylation along with related works and references to relevant papers.

The report then moves on to theoretical background where the theories which are prerequisites for understand the report explained in detail with mathematical proofs, formulas, supporting illustrations and tables.

Following the theoretical background, we explain the methods specific to our work and all the techniques we applied in order to carry out the research in the Methodology section. The dataset, model architecture, different approaches, tools and technologies used are mentioned in the section.

After the methodology, we summarize our results and findings of the research with implementational details and necessary diagrams and plots.

Finally, we end the report with conclusion and discussion where we mention the limitations of the research and future enhancements that can be done.

2 Literature Review

Protein post-translational modification (PTM) is the modification of individual amino-acid residues after proteins are translated from mRNA. Many PTMs are known, and some are very rare in nature. In lysine succinylation, a succinyl donor (- CO - CH2 - CH2 - CO -) is transferred to a lysine residue of a protein molecule. Protein shape, function, and physiochemical properties are controlled by this PTM, and it has an effect on metabolic processes, the incidence of many diseases, and their progression. Hence, various experimental and computational methods have been proposed for the prediction of succinylation.

2.1 Traditional Approaches

PTMs detection with high accuracy has been achieved using experimental techniques [3]. Lysine succinylation sites are identified using experimental methods such as high-performance liquid chromatography assays, mass spectrometry, and liquid chromatography-mass spectrometry [4]. However, the traditional methods are time-consuming, expensive, and involve potentially hazardous chemical substances. Therefore, computational methods are being developed over time to overcome these limitations.

2.2 Machine Learning Approaches

Over the years, different PTM site prediction tools have been developed using machine learning approaches. iSuc-PseAAC[5], which uses peptide position-specific propensity into pseudo amino acid composition with support vector machine algorithm, iSuc-PseOpt[6], which uses sequence-coupling effects into the general pseudo amino acid composition and k-nearest neighbor cleaning (KNNC), pSuc-Lys[7], which uses sequence-coupled information into amino-acid composition, HybridSucc[8], which combines deep-learning and conventional machine-learning algorithm, and SuccinSite[9], a classifier based on three sequence encoding and random forest (RF)-based approach, have been proposed for succinylation sites prediction. Similarly, PSSM-Suc[10] uses evolutionary information of amino acids by using position specific scoring matrix (PSSM), SSEvol-Suc[11] combines PSSM with secondary structure, and GPSuc[12] uses RF scores via logistic regression. The manual extraction of features, dependency on the data, likelihood for bias, and absence of unknown features that cause succinylation have caused machine learning approaches to suffer from limitations, paving the way for deep learning approaches.

2.3 Deep Learning Approaches

To a large extent, the need for feature engineering is limited by deep learning, which overcomes the limitation of machine learning. Therefore, several deep learning approaches have been developed. Succinylation sites are predicted by the CNN-SuccSite architecture[13], which uses four feature encoding techniques as input. DeepSuccinylSite was developed by Thapa et al.[14] using methods from natural language processing (NLP) to generate word embeddings from the protein sequence along with one hot encoding, eliminating the need for manual feature extraction. LSTMCNNsucc was developed by Huang et al.[15] using word2vec to generate embeddings and a combination of long short-term memory (LSTM) and convolution neural network (CNN). Similarly, MDCAN-Lys[16] captures feature information by using a three-way network, each of which consists of dense a convolution block and convolution block attention module. For the prediction of succinylation sites, pSuc-FFSEA[17] uses feature fusion from bag of words (BOW), one hot encoding with others and a stacking ensemble algorithm.

Recent developments have made language models (LMs) a important method for learning embeddings directly from big, unlabeled natural language datasets. The same techniques used by language models are being transported to protein to develop protein language models (pLMs). The pLMs are trained on large sequence of proteins and are able to extract features from protein sequence which can be then used for other tasks. LMSuccSite[18] used protein language model for generating protein embeddings and combined that with one hot encoding using a stacking ensemble for the prediction of succinylation sites.

Our project is built on the work of LMSuccSite[18] which combined pLMs with supervised word embedding to predict Lysine Succinylation. We use just pLMs in this project and compare the performance of language models. We combined two state-of-the-art protein language model ProtT5 and ProtBert to predict Lysine Succinylation. The performance of our method is found to be comparable with other state-of-the-art methods.

3 Theoretical Background

3.1 Embeddings

An embedding can be used as input to a machine learning algorithm which is a mathematical representation of a categorical or discrete variable in a lower-dimensional space. Categorical variables, like words, tags, or product IDs, are variables with a fixed, constrained set of possible values. Each categorical variable is converted throughout the embedding process into a vector of continuous values, where each dimension of the vector represents a distinct quality or feature of the original variable. As a result, machine learning algorithms that need numerical input can use the categorical variable.

During training, the machine learning algorithm itself can learn the embedding, or it can be pretrained on a sizable dataset using methods like word2vec or GloVe for natural language processing.

3.1.1 One Hot encoding

One-hot encoding is a simple and widely-used encoding method. For example, a categorical variable having as categories female, male, other can be encoded respectively with 3-dimensional feature vectors: [1, 0, 0], [0, 1, 0], [0, 0, 1]. In the resulting vector space, each category is orthogonal and equidistant to the others, which agrees with classical intuitions about nominal categorical variables[19].

Thus, we can see a sparse matrix-like structure (or a list of four one-hot vectors) for the sentence "This is a cat" as is illustrated in Figure 1 Each row of the yellow block is a vector where there exists only a single entry of 1, indicating the presence of the very word in the vector. Hence, with this technique, we can think that input data is sparse and exists in a very high-dimensional space[20].

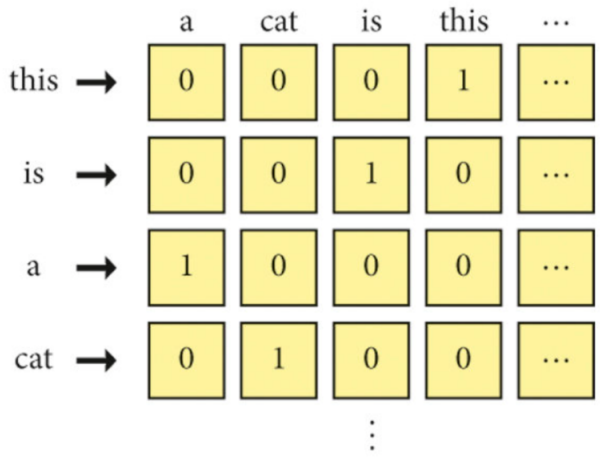


Figure 1: One hot encoding

3.1.2 Word embedding

On the other hand, word embedding takes context into account and gives words with similar meaning or influence in a sentence similar value for a specific feature. They are dense vectors of fixed size that represent words in a continuous vector space. Word embedding is a way to represent a word with fixed length vectors of continuous real numbers. It maps a word in a vocabulary to a latent vector space where words with similar contexts are in proximity. Through word embedding, a word is converted to a vector that summarises both the word's syntactic and semantic information[21]. This illustrates the example of word embedding where each row in blue color is a dense representation of the word in 4-D space.

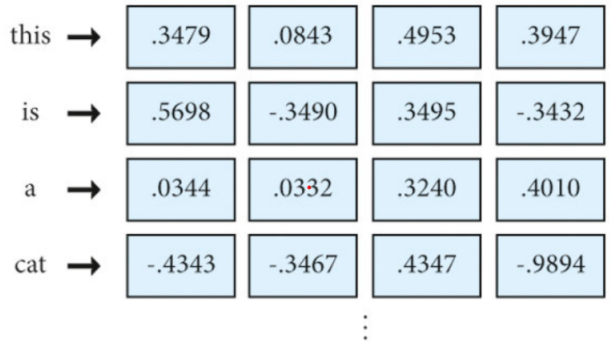


Figure 2: Word Embedding

3.2 Language Model

A language model in NLP is a probabilistic statistical model that determines the probability of a given sequence of words occurring in a sentence based on the previous words. It helps to predict which word is more likely to appear next in the sentence.

Google's BERT is a transformer-based language model which is pre-trained on enormous quantities of text data. Being bidirectional, it can process textual information in multiple directions and excel at comprehending context more efficiently than previous models.

3.2.1 Protein Language Model

The protein language model is trained using deep learning methods like recurrent neural networks or transformer models on massive datasets of protein sequences. The model is fed a series of amino acids during training and learns to anticipate the subsequent amino acid that is most likely given the context of the preceding amino acids. By repeatedly sampling from the learnt probability distribution over amino acids, the model can then produce new protein sequences.

The vast number of protein sequence makes difficult for a protein language model to compute the full complexity of protein sequence. To solve this, transfer learning is used where a pre-trained model is fine-tuned on a smaller dataset for a specific protein prediction task.

3.3 Feature Construction

ProtBert is based on Bert architecture. It is pretrained on a large collection of protein sequences in self-supervised fashion. Self-supervised means that the protein sequences were not labelled by human and it was trained on raw protein sequences only with an automatic process to generate inputs and label from those protein sequences. Transformer is the intuition behind language model architecture.

3.3.1 Transformer Architecture

The Transformer follows encoder decoder architecture using stacked self-attention and point-wise, fully connected layer[22]. Encoder maps input sequence to sequence of continuous representation. The first layer implements a multi-headed self-attention mechanism and second layer is a fully connected feed-forward network with Rectified Linear Unit (ReLU) activation in between.

The first layer of decoder receives previous output of decoder stack and implements multi-head self attention. The second layer receives input from previous layer as well as encoder and performs multi-head self-attention mechanism. The final layer is a fully connected feed-forward network. Encoder-Decoder Structure of Transformer Architecture

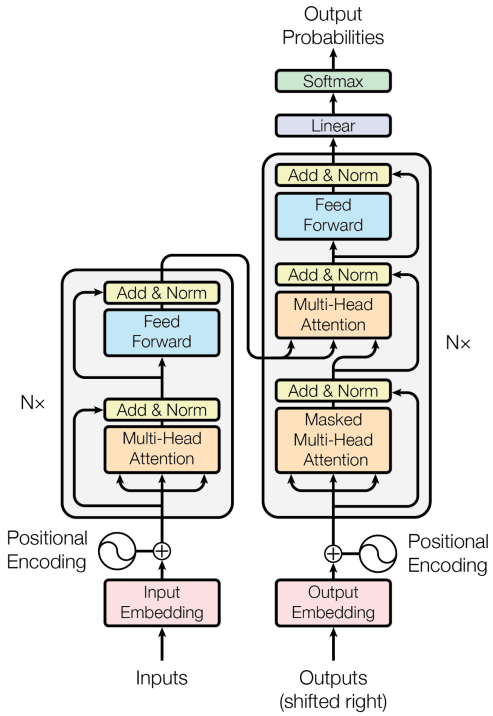


Figure 3: Encoder-Decoder Structure of Transformer Architecture

3.3.1.1 Input Embedding

Firstly it starts with feeding input to word embedding layer. A word embedding layer serves as a reference table that retrieves a pre-learned vector representation for each word. Neural networks operate on numerical values, so each word is associated with a continuous-valued vector representation. This layer contains an index for every word in the vocabulary, and for each index, a corresponding vector is assigned. The vectors initially contain random values, but during training, the model adjusts them to values that are optimized for the given task. Embedding layer as awhole takes input indices and converts them into word embedding.

3.3.1.2 Positional Encoding

Since transformer contains no recurrence and no convolution, in order for the transformer to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, it adds "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks[22].Position embeddings are useful in our architecture since they give our model a sense of which portion of the sequence in the input or output it is currently dealing with [23]. Transformers use a smart positional encoding scheme, where each position/index is mapped to a vector. Unlike LSTM which takes embedding sequentially in there designated order, transformer take all the embedding at once. Hence, the output of the positional encoding layer is a matrix, where each row of the matrix represents an encoded object of the sequence summed with its positional information.

3.3.1.3 Encoder

The process of the transformer involves an encoder that identifies characteristics of the input sentence, and a decoder that utilizes these characteristics to generate a corresponding output sentence. The transformer's encoder is comprised of several encoder blocks, which receive the input sentence and ultimately provide the decoder with its necessary input features. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network[24].

- Multi-head self-attention mechanism:

  An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors[24]. The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from

the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder[22]. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this[24]. The scaled dot product attention is calculated as,

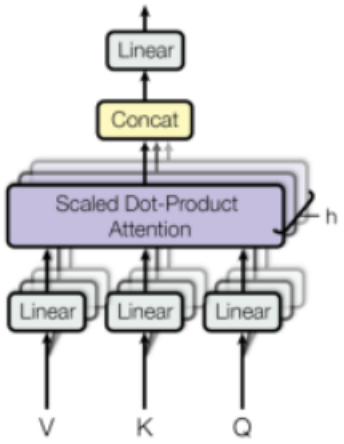


Figure 4: Scaled dot product attention

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- Position-wise fully connected feed-forward network:

  The decoder takes as input a sequence of tokens and generates a sequence of output tokens. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, it employs residual connections around each of the sub-layers, followed by layer normalization. It modifies the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

  In addition to attention sub-layers, each of the layers in the encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between[22].

$$FFN(x) = max(0, xW1 + b1)W2 + b2$$

3.3.1.4 Decoder

The decoder takes as input a sequence of tokens and generates a sequence of output tokens. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, it employs residual connections around each of the sub-layers, followed by layer normalization. It modifies the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

1. Masked multi-head attention:

   It receives inputs with masks so that the attention mechanism does not use information from the hidden (masked) positions. MHA possesses the ability to more efficiently model long-term dependencies. Moreover, masking can be employed to ensure that the MHA mechanism remains causal — an attribute critical for real-time processing.[25]

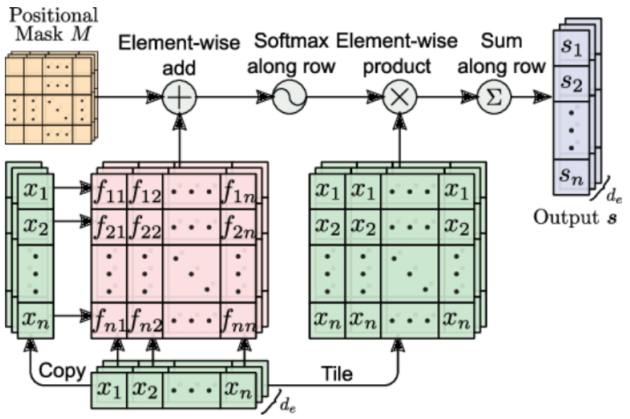


Figure 5: Masked self attention mechanism

3.3.2 Bert Architecture

Bidirectional Encoder Representations from Transformers (BERT) uses the Transformer architecture. It makes use of the encoder model of the transformer to generate a language model. It applies bidirectional training of transformer to language modelling[26]. It takes the entire sequence of words at once. It uses two strategies for training:

1. Masked Language Model : 15% of the words in each sequences are replaced with a [MASK] token. The model then tries to find the original value of the mask.
2. Next Sentence Prediction : The model receives sentences as input and tries to predict if the next sentence is related to previous or not.

3.3.2.1 ProtBert Model

It was trained using both UniRef100 and BFD-100 datasets[27]. It is trained for 300k steps on sequences with maximum length of 512 and then for another 100k steps on sequences on length 2048. So, the model can extract useful features from shorter sequences and is efficient on longer sequences.

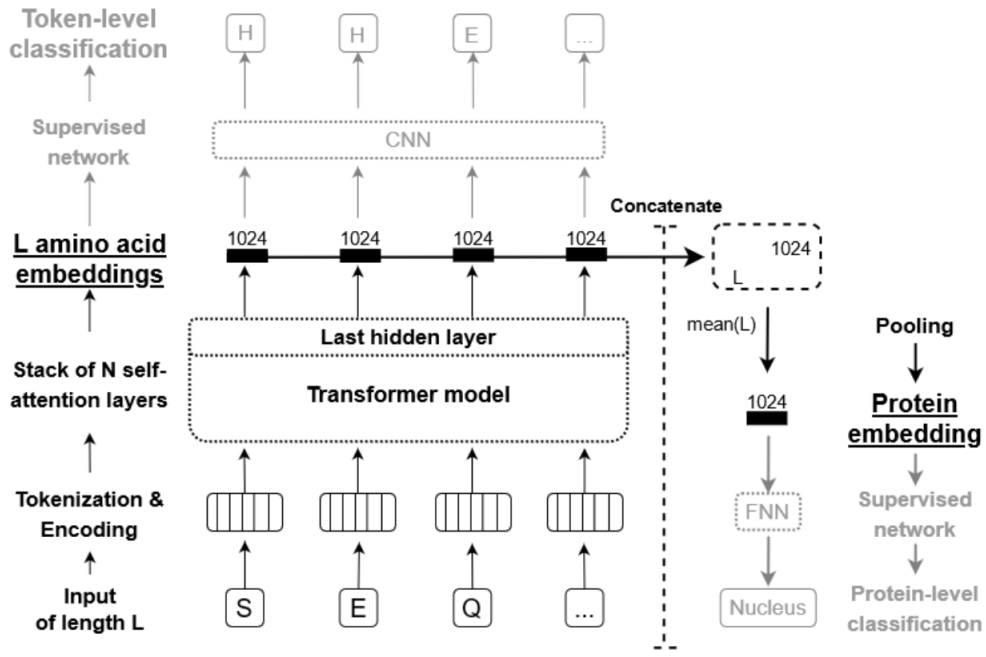


Figure 6: Feature Extraction using ProtBert Model

Protein sequence is tokenized and positional encoding is added. The resulting vectors are sent

through ProtBert model to create context-aware embeddings for amino acid. The embeddings are contatenated and pooled to obtain fixed size embeddings irrespective of the input length.

The resulting protein-level embedding is utilised with Convolution Neural Network (CNN) and other architectures to identify lysine succinylation sites.

3.3.3 Prot-T5 Model

ProtT5, a protein language model is designed to analyze and predict protein sequences and structures. ProtT5 is a version of the T5 language model, a highly scalable transformer-based model developed using enormous amounts of textual data. It has been trained on a large corpus of protein sequences and structures, and it is capable of a wide range of tasks relating to proteins, including predicting protein structures, functions, and interactions. One of ProtT5's distinguishing qualities is its capacity for transfer learning, which enables it to be fine-tuned on smaller datasets for particular protein-related tasks.

The protein sequences are uppercased and tokenized with a vocabulary size of 21 using a single space. The protein sequences were trimmed and padded up to 512 tokens during the preprocessing stage.

15% of the amino acids in ProtT5 are masked, and in 90% of cases, the masked amino acids are replaced with the [MASK] token. The masked amino acids are replaced by a random amino acid from the one they replace in 10% of the cases.

ProtT5 model is trained utilizing sequence length 512 in 2k batch size, and about 3B parameters over a single TPU Pod V2-256 for a total of 991.5 thousand steps[27].

3.4 ANN

An ANN is made up of layers of interconnected neurons, that analyze input data and produce predictions as output. Each neuron in an ANN takes input signals from other neurons or from the input data, transforms the input signals into output signals using a mathematical function, and then sends the output signals to the next layer of neurons. In order to reduce the discrepancy between the model's anticipated outputs and the actual outputs for a set of training instances, the model learns to modify the weights of the connections between neurons during training. The training data are continually fed through the network in this process, and the weights are changed based on the error between predicted and actual output[28].

3.4.1 Activation Functions

Activation adds non-linearity to the output of a neural network model. Without an activation function, a neural network is simply a linear regression. The activation function in ANN determines whether or not to stimulate a neuron by generating a weighted sum and then adding bias to it. The obtained values are mapped between 0 and 1 or -1 and 1, etc. (depending upon the function).

The mathematical equation for calculating the output of a neural network is:

$$Y = Activation(\sum(weight * input) + bias)$$

The two main categories of activation functions are:

1. Linear Activation Function

   The function is a line or linear, as you can see. Thus, no range will be used to limit the output of the functions. The complexity or other parameters of the typical data that is input to the neural networks are unaffected.For example:

   - Linear Function

     $$f(x) = x$$

     with range of $(-\infty, \infty)$

   - Binary Step Function

     $$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

2. Non-Linear Activation Function

   The non-linear activation function transforms the input in a non-linear way, enabling it to learn and approximate complex functions. It facilitates the model's generalization or adaptation to a variety of data and facilitates output differentiation.

   - Sigmoid Activation Function

     The Sigmoid Function curve has an S-shaped appearance. The sigmoid activation function maps its input into a range between 0 and 1.

     $$\sigma(x) = \frac{1}{1+e^{-x}}$$

   - Tanh Activation Function

     The tanh function maps its input into a range between -1 and 1. Tanh is sigmoidal as well (s-shaped).

     $$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

   - ReLU(Rectified Linear Unit) Activation Function

     The ReLU (Rectified Linear Unit) activation function is a mathematical function that returns its input if it is positive, and 0 otherwise.

     $$\text{ReLU}(x) = \max(0,x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

   - Leaky ReLU Activation Function

     The Leaky ReLU Activation Function (LReLU) is very similar to the ReLU Activation Function. Instead of sending negative values to zero, a very small slope parameter (α) is used which incorporates some information from negative values, which cause the model to effectively fit or train from the data by incorporating the values from negative data.

     $$\text{LeakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

3.4.2 Regularization

Regularization reduces generalization error[29].Regularization is a crucial technique in Artificial Neural Networks (ANN) that helps prevent overfitting and improve the generalization of the model. Overfitting occurs when the model becomes too complex and learns the noise in the training data

instead of the underlying pattern. Regularization techniques add a penalty term to the loss function that encourages the model to have smaller weights and biases, thus reducing the complexity of the model[30].

3.4.2.1 L2(Ridge) Regularization

L2 regularization, also known as weight decay or Ridge regularization, is used to prevent overfitting and improve generalization by adds a penalty term proportional to the square of the weights to the loss function, which encourages the model to have smaller weights and biases[31].

$$L2regularization = \lambda \|\|\|w\|\|\|^2$$

where λ is the regularization parameter and $\|\|\|w\|\|\|^2$ is the square of the L2 norm of the weight vector. The L2 norm of the weight vector is defined as:

$$\|\|\|w\|\|\|^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

where $w_1, w_2, \ldots, w_n$ are the weights in the weight vector. The L2 regularization penalty term is added to the standard loss function of the model, and the overall loss function becomes:

$$Lossfunction = Standardlossfunction \; + \; L2regularization$$

The value of the regularization parameter, λ, determines the strength of the regularization. A higher value of λ results in smaller weights and biases and more regularization. The regularization parameter λ is usually determined using cross-validation techniques[32]. The L2 regularization is particularly useful when the data has many correlated features, as it encourages the model to have small weights across all the features.

3.4.2.2 Dropout Regularization

Dropout is a regularization technique used in deep learning to prevent overfitting and improve generalization. It works by randomly dropping out (setting to zero) a proportion of the neurons in the model during training, which forces the remaining neurons to learn more robust and less co-independent features, as it can no longer rely on any single neuron to always be present. This technique is useful when there is a large number of neurons in the network [29].

Let x be the input to a layer and y be the output. During training, we randomly set a fraction p of the neurons to zero, and scale the remaining neurons by $1/(1-p)$. The output y is then given by:

$$y = \frac{1}{1-p} \cdot r \cdot x$$

where r is a binary mask vector, where each element is set to 0 with probability p, and 1 with probability $(1-p)$ [31]. During inference, all the neurons are used, but their outputs are scaled by $(1-p)$ to account for the fact that they were active during training.

3.5 CNN

3.5.1 Convolution

Convolution combines two functions to create a third function. In image processing, convolution involves sliding a filter over input image and calculating the inner product between the filter and overlapping region of the input.

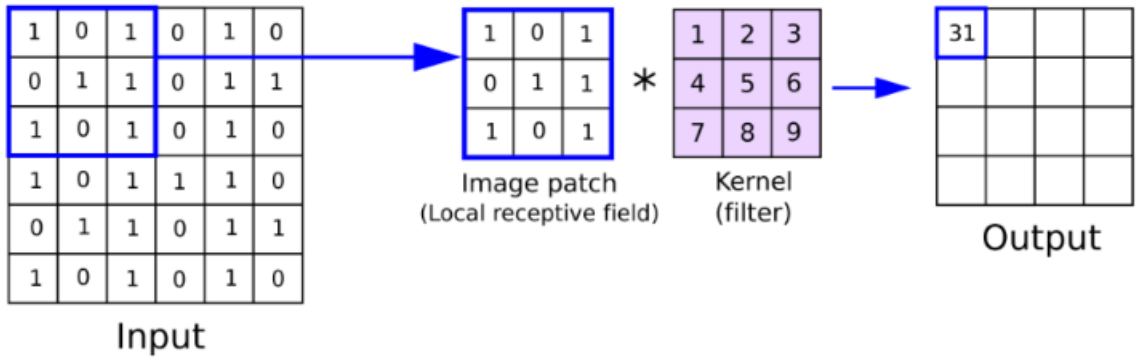


Figure 7: Convolution

The mathematical operation follows as:

$$1 \times 1 + 2 \times 0 + 1 \times 3 + 0 \times 4 + 1 \times 5 + 1 \times 6 + 1 \times 7 + 0 \times 8 + 1 \times 9 = 31$$

The filter slides to the next image window and repeats until the entire image is covered.

A convolutional neural network (CNN) is a form of neural network that automatically learns hierarchical representations of data through the use of convolutional layers. The input data for a CNN is passed via a number of convolutional layers, nonlinear activation functions, and pooling layers. Each convolutional layer generates a set of feature maps and applies a set of filters to the input.

3.5.2 1D CNN

A 1D CNN (Convolutional Neural Network) is a type of neural network that applies convolutional filters over a one-dimensional input signal to learn and extract relevant features for tasks such as time series analysis and natural language processing.

3.5.3 Filter

A filter is a small matrix of weights as seen in fig 7 that is used to perform convolution on an input signal or image to extract features. The filter is generally smaller than the input and is slid over the input to obtain new output.

3.5.4 Strides

In convolution, stride refers to the step size at which the filter is moved over the input during the convolution operation. A larger stride value means that the filter moves more quickly over the input, producing a smaller output signal or image. Conversely, a smaller stride value means that the filter moves more slowly over the input, producing a larger output.

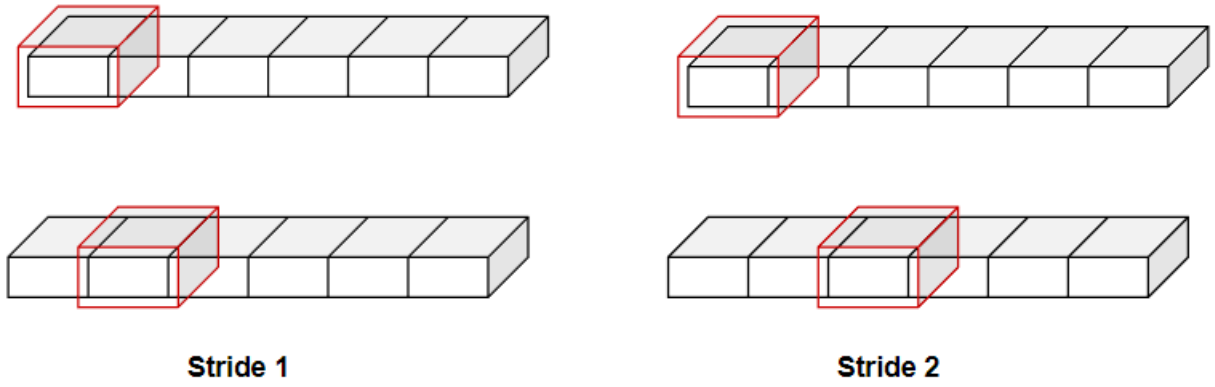


Figure 8: Strides

3.5.5 Padding

Padding is done to preserve the dimension of the input data when performing convolution. The output of a filter on a input is of smaller dimension that the input. Padding involves adding extra rows and columns of zeros around the input to preserve the dimension after convolution.

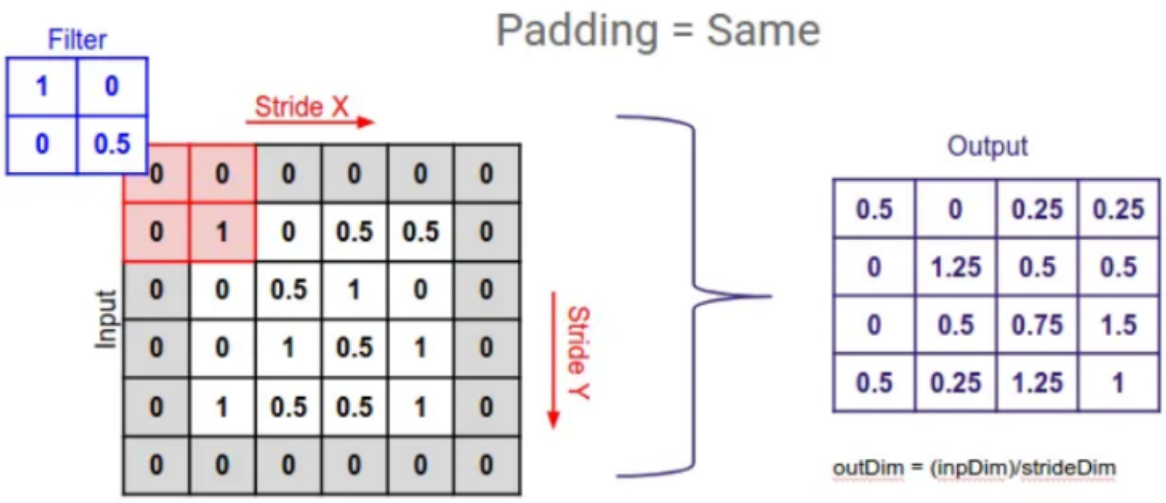


Figure 9: Padding

3.5.6 Pooling (Max-Pooling)

Pooling is used in CNN to reduce the size of feature map while preserving important information. Max-pooling is a type of pooling that selects the maximum value within a certain window of the feature map, discarding the rest of the values.

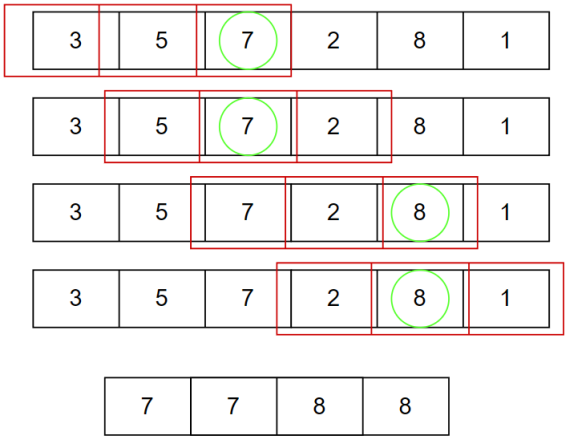


Figure 10: Max Pooling

3.6 Transfer Learning

Transfer learning is a machine learning technique where a model learned on one job is repurposed or adapted for a new task. Transfer learning avoids the need for large amount of training data and helps achieve higher accuracy on new task by reducing computational time and resources[33].

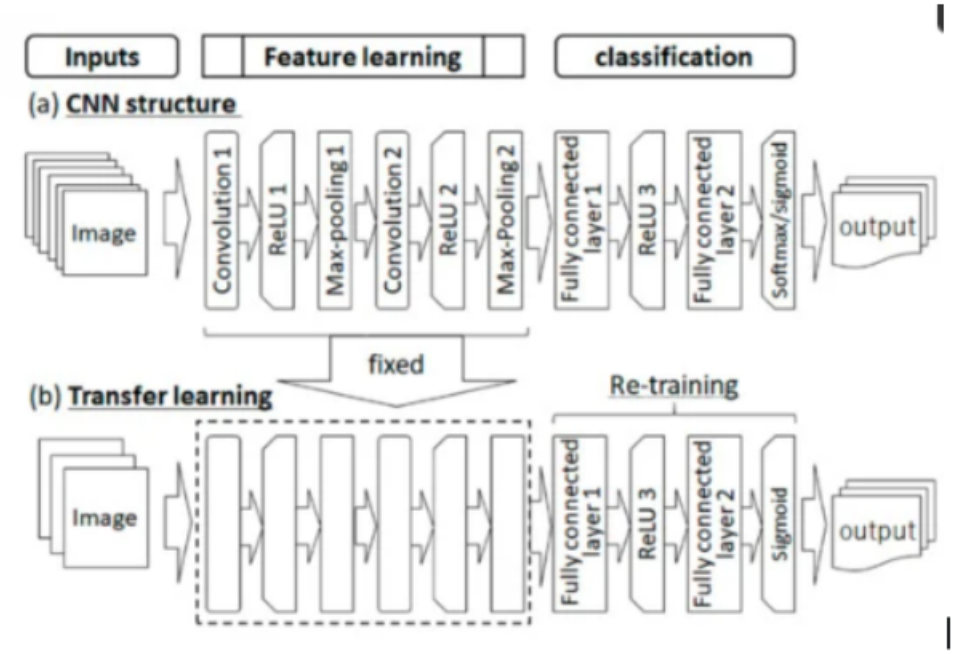


Figure 11: Transfer Learning

The transfer learning extracts the features as common knowledge from Domain A and then applies common knowledge to predict in model B[34].

3.7 Ensemble Method

Ensemble methods combines the predictions of multiple individual models to improve the overall performance and accuracy of the model. When individuals models cannot make accurate prediction or the problem is too complex, ensemble methods are used.

3.7.1 Bagging

Bootstrap aggregation (bagging) involves training an ensemble on bootstrapped data sets. A bootstrapped set is created by selecting from the original training data set with replacement[35], a bootstrap set may contain a given example zero, one, or multiple times. Ensemble members can

also have limits on the features (e.g., nodes of a decision tree), to encourage exploring of diverse features.

3.7.2 Boosting

Boosting involves training successful models by emphasizing training data misclassified by previously learned models. In some cases, boosting has yielded better accuracy than bagging, but tends to over-fit more method employs decision trees, which significantly decrease variance [35]. This decrease in variance leads to an improvement in accuracy by reducing the occurrence of overfitting, which is a common problem in many predictive models.

3.7.3 Stacking

Stacking combines the outputs of a set of base learners and lets another algorithm, referred to as the meta-learner, make the final predictions typically yields performance better than any single one of the trained models[36]. It has been successfully used on both supervised learning tasks (regression, classification and distance learning) and unsupervised learning (density estimation). The stacking process typically involves dividing the training data into multiple subsets, with each subset being used to train a different base model. The meta-model learns how to combine the predictions from the base models to produce a final prediction.

4 Methodology

4.1 Data-sets

4.1.1 Data Collection

This dataset, which was originally sourced from the UniProtKB/Swiss-Prot database and the NCBI protein sequence database, consists of experimentally validated succinylation sites that were supplied by Hasan et al.[9]. First, a CD-hit technique[37] was used to retrieve these sequences and remove redundancy from them using a cut-off similarity of 0.3 to any other proteins in the dataset. As a result, 2322 protein sequences produced 5009 succinylated sites. The sequences were then randomly divided into two sets: a training set that contained 2192 protein sequences and a testing set that contained 124 proteins. There are 4755 and 254 succinylation sites, respectively, in the training and test set sequences. There are 50,565 non-succinylation sites and to deal with imbalance class, 4755 random samples are chosen by performing undersampling. Finally we used independent test set of 254 succinylation sites and 2977 non-succinylated sites.

4.1.2 Data Preprocessing

The collected data had some missing values and a few redundant data. The protein "P15559" position of lysine residue is greater than length of sequence, which indicates that given data is corrupted. So, we filtered those data. The dataset was supposed to be of 4750 succinylated and 4750 non-succinylated site but after analysing the dataset, non-succinylated site had 221 duplicates and 1 corrupted protein in succinylated site. Thus after filtering the dataset we were left with 4529 non-succinylated site and 4749 succinylated site. Among 254 test set of each succinylated and non-succinylated site, 12 duplicates were found in succinylated site. Therefore, the total number of data used were 254 for succinylated and 242 for non-succinylated site.

Performing further analysis, we found the max length of protein to be 7450 and min length of 46. The mean length of protein was 496, 921 with a standard deviation of 550,1221 for succinylated and non-succinylated site respectively.

Each succinylated site and the non-succinylated site were represented by a sequence. In this paper, we employed a window size of $(2n+1)$, where the candidate site (lysine residue) is always at the central position $(n+1)$, and the upstream positions were labeled as 1, 2, and so on, and the downstream positions were labeled as $(n+1)$, $(n+2)$, and so on. Padding of "X" was done to make up the difference if the number of residues immediately upstream or immediately downstream of

the candidate site was less than n. For example, the original sequence of the succinylated protein "P27348" is "MEKTELIQKAKLAEQAERYDD...". The first lysine(K) is the candidate site and for the window size of 10, the resultant sequence is "XXXMEKTELIQ". All the sequence samples contain only 20 standard amino acids and a pseudo amino acid "X" making a total of 21 amino acids.

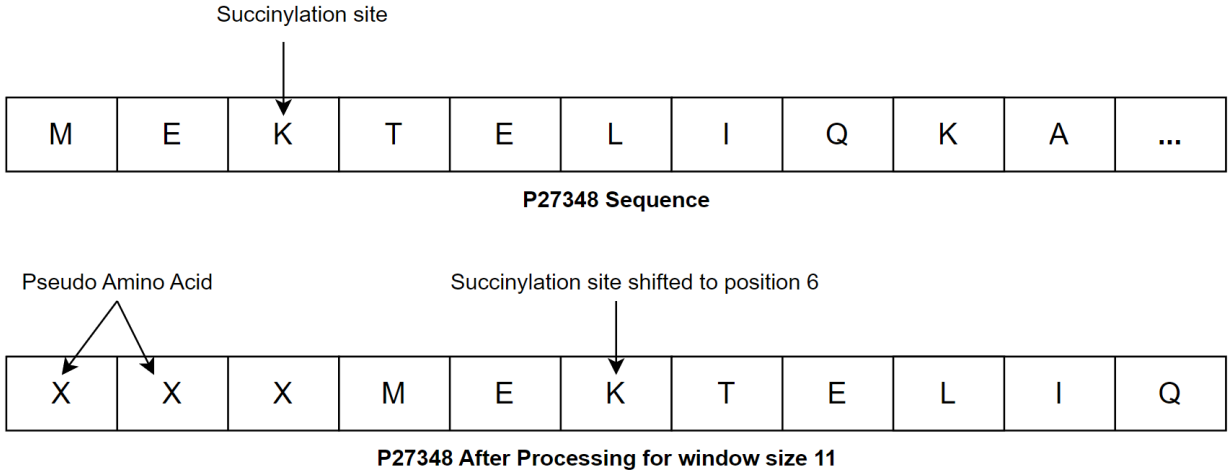


Figure 12: Data preprocessing

4.1.3 Data Distribution

After data preprocessing, we obtained a total dataset of 9278 for training and 496 for testing. We used 254 succinylated and 2973 non-succinylated site as our independent test set to evaluate our model with existing methods.

Dataset type	Succinylated	Non Succinylated
Training data	4750	50565
Training data (after balancing)	4750	4750
Training data (after balancing +preprocessing)	4749	4529
Benchmark independent test data	254	2977

Table 2: Dataset Description of Training and Benchmark Dataset

4.2 Model Architecture

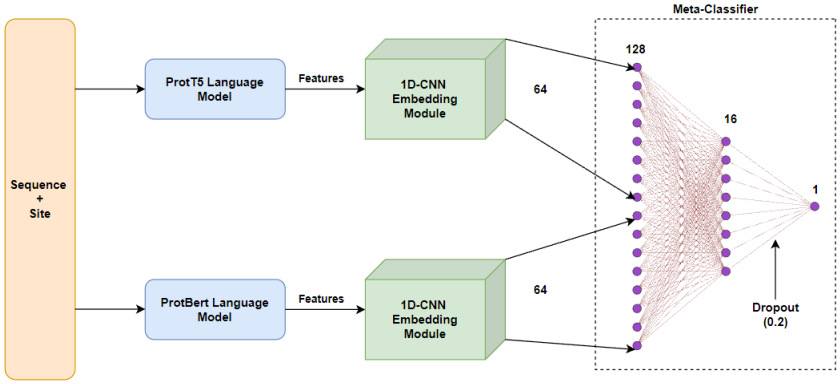


Figure 13: MetaLMSuccinylSite (MLMSS)

In this paper, we proposed an ensemble deep learning model which captures essential features from the unsupervised protein language model (ProtBERT, ProtT5) that is pre-trained on a large dataset of proteins. The features from the language model were subjected to separate 1D-CNN-based architecture. Initially, we used the ProtBERT model and then move on to the ProtT5 model. Eventually, two 1D-based-CNN, one for the ProtT5 model and the other for the protBert model, produced features for a meta-classifier based on ANN.

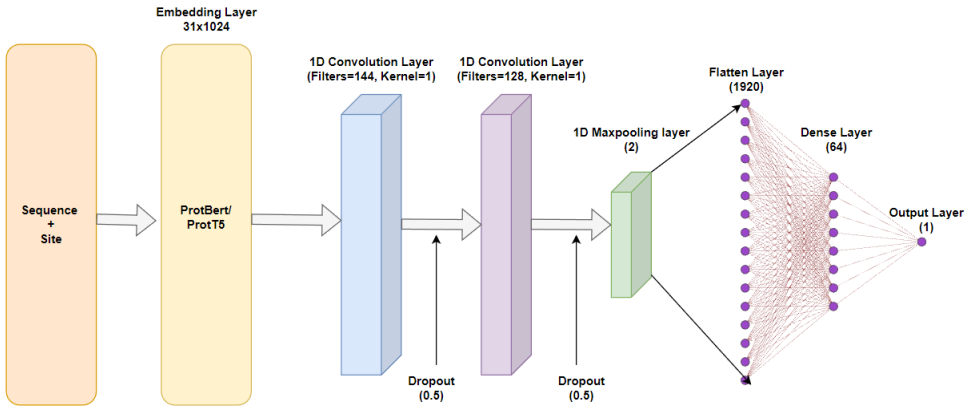


Figure 14: LMSuccinylSite(LMSS)

In figure 14, the model takes the input sequence as an input which is then passed through a pre-trained sequence-based protein model and generates an output of embedding size 1024 x 1 vector for each amino acid of the sequence. In our case, we have chosen the ProtBERT and T5 models. We took the n embeddings from the left of the protein position, the protein site embedding, and n embeddings from the right of the protein position and formed a window size of (2 x n+1). These

embeddings are then passed through 1D CNN architecture which consists of a convolution layer for feature extraction, max pooling for down-sampling, and a fully connected layer.

The input protein sequence is first passed through a pre-trained sequence-based protein model, which maps the amino acids represented by the alphabet in the sequence to a high-dimensional vector space, where each vector captures the semantic meaning of amino acids in the protein sequence.

To further capture the features between the neighboring amino acids, we used 1D CNN architecture. We applied a convolutional layer with multiple filters to the sequence of embeddings. The filters slide over the sequence, extracting valuable features for classification. Also, the max-pooling layer is used to reduce the dimensionality of the feature map obtained from the convolutional layer, which helps to prevent overfitting and makes the model more efficient. An activation function such as ReLu and LeakyReLu are applied to the output of the convolutional layer to introduce non-linearity.

Finally, the output of the max-pooling layer is flattened and fed into a fully connected layer which produces a probability distribution over the possible classes. The final layer of the model applies a sigmoid activation function to the output of the fully connected layer. The class with the highest probability is then chosen as the predicted class.

4.2.1 ProtBERT 1D CNN

In this approach, we used 4529 non-succinylated sites and 4749 succinylated sites and calculated the embeddings from the ProtBERT model, these embeddings are then broken down into window sizes of 21, 31, 41, and fed to the 1D CNN model.

We noted that taking a window size of 31, gave us a better result compared to other window sizes. But, since we were using the ProtBERT model, a pre-trained model that was trained on subset of Uniref50 contrast to protT5, which was trained on large dataset of Uniref50, and PDB-100, protT5 seemed a good choice as it is thorough, non-redundant, and more likely to include isofunctional clusters.

4.2.2 ProtT5 1D CNN

Compared to ProtBert, ProtT5-XL-Half-Uniref50-enc is an improved version in many aspects. It is based on the T5 transformer architecture, which can handle lengthy sequences and capturing complex relationships between different sequence. ProtBert, on the other hand, is built on the BERT architecture, which was initially intended for tasks related to natural language processing

and might not be as suitable for protein sequence analysis. ProtT5-XL-Half-Uniref50-enc is able to generalize to new sequences because it was trained on a combination of the Uniref50 and PDB-100 databases, whereas ProtBert was trained on a smaller fraction of Uniref50.

We used ProtT5-XL-Half-Uniref50-enc to extract embeddings from the protein sequences. The ProtT5-XL-Half-Uniref50-enc pre-trained model that we utilized differs from the original ProtT5 (ProtT5-XL-UniRef50) model solely in the encoder component because it incorporates the original ProtT5-XL-UniRef50 model's encoder utilizing half-precision (float16).

This approach is similar to the above ProtBERT 1D CNN approach as only the pre-trained model was different. As predicted, this approach slightly increased the performance compared to ProtBERT 1D CNN.

4.2.3 ProtBERT +T5 Metaclassifier

While testing our model with ProtBERT and T5 embeddings, we noticed that the embeddings generated by ProtBERT and T5 embeddings were nowhere close to each other. To confirm this, we also checked cosine similarity and found it to be only 0.05. The embeddings from different models might be extracting features that are not correlated.

$$\text{Cosine similarity} = \cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

In this approach, we took the embeddings from ProtBERT as well as from protT5 and passed them to two independent 1D-CNN models. The output of the 1D-CNN model produced a dense layer of 64. These dense layers of size 64 from both the CNN models were concatenated to obtain a resulting dense layer of 128 which were further reduced to 16, and 1 as shown in fig 13. We took the window sizes 21, 31, and 41 and repeated this process.

We observed significant improvement in the performance of the model when the input window size was 31. Overall, this approach performed better compared to any other approaches mentioned above.

4.3 Model Loss Function

Since, from the given protein sequence, we want to predict whether the given protein site is a succinylated site or a non-succinylated site, it is a binary classification problem. Ideally, we would want a probability of 1.0 for the succinylated site and 0.0 for the non-succinylated site. To evaluate our prediction, we need a good loss function. For binary classification, Binary Cross- Entropy

(Log Loss) performs better than any other loss function because it is intended to penalize models more severely for inaccurate predictions that are made closer to the actual label. In other words, if the model predicts a probability close to one but the true label is negative, the penalty will be significantly greater than if the model predicted a chance of 0.6. This makes it easier to check sure the model is accurate and doesn't make confident inaccurate predictions.

The binary Binary Cross- Entropy Loss Function is:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N}[\log(p(y_i)) + (1-y_i)\cdot\log(1-p(y_i))]$$

Where y is the label and p(y) is the predicted probability for the label.

4.4 Model Optimizer

Various optimizer algorithms can be used to train binary classification models. Among them, we used Stochastic Gradient Descent (SGD) and Adam, as they were popular among binary classifications.

4.4.1 Stochastic Gradient Descent(SGD)

SGD is an iterative algorithm that optimizes the loss function. It updates the model parameters by calculating the gradient of the loss function with respect to the parameters and adjusting them in the opposite direction of the gradient.

$$\theta_{i+1} = \theta_i - \alpha\cdot\nabla_\theta J(\theta; x_j, y_j)$$

4.4.2 Adam

It is a variation of SGD that computes adaptive learning rates for each parameter. It is efficient and effective for many deep learning tasks, including binary classification.

$$\theta_{i+1} = \theta_i - \hat{m_t}(\alpha/\sqrt{\hat{v_t}+\varepsilon})$$

where,

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t}, \quad \hat{v_t} = \frac{v_t}{1-\beta_2^t},$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\delta L}{\delta w_t}\right)^2,$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left(\frac{\delta L}{\delta w_t}\right),$$

ε, a small positive constant to avoid division by zero, and β_1 and β_2, decay rates of an average of gradients.

4.5 Model Optimization

Many hyperparameters in the model need to be tuned. So how do we find the right value of hyperparameters? For example, how do we find the right number of filters, the right value for dropout, and the right number of the dense neuron? One might opt to do it manually, testing and noting every loss, accuracy, and making a grid table. But it is a tedious process, so we used Keras Tuner to help speed up the process.

- **KerasTuner**

  Keras Tuner is an open-source library for hyperparameter tuning, which allows to define a space of hyperparameters, and then search over that space to find the optimal combination of hyperparameters for the model.

  By configuring the search space for the number of filters, dropout, activation layers, optimizers, the best optimal parameter and configuration is then determined.

4.6 Methods used to reduce Overfitting

4.6.1 Regularization

$$\hat{H_p}(q) = H_p(q) + \lambda \sum_{j=1}^{p} |\beta_j| \qquad \text{(Lasso Regularization)}$$

$$\hat{H_p}(q) = H_p(q) + \lambda \sum_{j=1}^{p} \beta_j^2 \qquad \text{(Ridge Regularization)}$$

The bold term in the above loss function represents the regularization term. When lambda equals zero, it adds no penalty; when lambda is very high, it adds too much weight, which might lead to under-fitting. So, choosing the right value is essential to reduce overfitting.

Setting up the model with both L1 and L2 regularization, we found L2 regularization performed best as it reduced the overfitting as well as improved the performance of our model. On the other hand, L1 regularization led to underfitting. Training on more epochs did not have a significant impact on the performance.

Since capturing relevant features and the dimensions of the embeddings are important for the classification task, L2 regularization was the best choice.

4.6.2 Dropout

We applied dropout after the convolutional layers and before the fully connected layers. The overall size of the activations was kept constant by randomly dropping a fixed proportion of neurons during training and scaling up the remaining neurons by a factor equal to the inverse of the dropout rate. A certain percentage of the dropout was added after the dense layer to ensure that model does not rely too heavily on any one feature or set of features, which made our model robust to variations in the input.

4.6.3 Early Stopping

Early Stopping stops the training process before the model begins to overfit the training data, preventing overfitting. The goal of early stopping is to evaluate the model's performance on a different validation set throughout training and halt the process when the validation set begins to deteriorate.

In the Keras callback method, we set the restore_best_weights to true with a 'patience' set to five. If the validation set's performance did not improve even after waiting for five epochs after the performance drop, the training process was terminated and the best weights were restored.

4.6.4 Batch Normalization

Batch Normalization (BatchNorm) enables faster and more stable training of deep neural networks (DNNs). In order for batch normalization to work, each layer of the network's activations must first be normalized to have a zero mean and unit variance before being scaled and shifted using previously learnt parameters[38].

The batch statistics (mean and variance), which are obtained over each mini-batch during training, are then used for normalization. To maintain consistency between training and inference, the batch statistics are often replaced with population statistics (calculated across the whole training set).

In this paper, we implemented BatchNorm to reduce overfiting by lowering internal covariate shift and enhancing gradient flow.

4.7 Stratified Ten Fold Cross-Validation

To evaluate the model, we need to split our dataset into training set and test set. But there is a problem when we change our random state while randomly splitting our datasets. We obtain different levels of accuracy, and our model's performance varies, making it difficult to evaluate. But, by utilizing stratified sampling, we can get around this issue.

- **Random Sampling**

  In random sampling, each data in the dataset has an equal chance of being selected without any bias or stratification.

- **Stratified Sampling**

  To ensure that the sample is representative of the population, stratified sampling divides the dataset into homogenous classes depending on some pertinent features. A random sample is then chosen from each class in proportion to its size.

For example, assume we have 1000 dataset, 800 are positive and 200 are negative. While splitting the dataset into training and test set in 8:2 ratio, we might get all negative data as test set and all positive data as training set. But stratified sampling ensures that represents the entire dataset in equal proportion. It selects 80% of 800 from positive class and 80% 200 from negative class as training set and remaining as test set.

In 10-fold cross-validation, dataset is divided into 10 fold through random sampling. Among 10 fold, one fold is selected as validation set and remaining set as training set as the model is trained. This process is repeated 10 times with next fold that has not been used as validation set.

Stratified 10-fold cross-validation is same as 10-fold cross-validation but it does stratified sampling instead of random sampling. This type of train test split results in good accuracy.

We employed stratified 10-fold cross-validation in our model, and measured the average accuracy, f1-score, MCC score and other performance metrics as this split represent the portion of the overall dataset.

4.8 Experiment Versioning

Version control and experiment tracking are integrated in ML experiment versioning. Experiment versioning is a technique used to track each iteration of a machine learning experiment, together with the data, code, and results that go with it.

Kaggle supports experiment versioning via kernels and datasets. Jupyter notebooks that can be used to develop and run machine learning models are known as kernels. Each time a kernel is saved, a new version is created, allowing to keep track of the code and revert to an earlier version. Datasets also include a version history for tracking data changes over time.

Utilizing experiment versioning, it has been simple to maintain code and contrast our various strategies. It was simpler to spot mistakes and make the required corrections by comparing various code versions. We completed our work by incrementally improving the code, monitoring experiments, and comparing the changes between various versions of the data and code.

4.9 Tools and Technologies Used

4.9.1 Numpy,Pandas and Matplotlib

NumPy is a Python library for numerical computations with multi-dimensional arrays and matrices.

Pandas is a Python library for data manipulation and analysis, built on top of NumPy. It provides data structures for efficiently storing and manipulating large, heterogeneous datasets, as well as tools for data cleaning, merging, reshaping, and visualization.

Matplotlib is a Python library for creating static, animated, and interactive visualizations in Python. It provides a wide range of tools for creating 2D and 3D plots, histograms, bar charts, scatter plots, and more. Matplotlib is widely used in the scientific and data visualization communities, and is highly customizable and extensible.

4.9.2 Pytorch

PyTorch is an open-source Python machine learning framework that provides a dynamic computational graph and a tensor computing library. It is widely used in deep learning research and applications, and supports various neural network architectures such as convolutional neural networks, recurrent neural networks, and transformers. PyTorch allows for easy debugging, model visualization, and deployment, and provides interfaces for integrating with other popular Python libraries such as NumPy and Matplotlib.

4.9.3 Tensorflow

TensorFlow is an open-source Python library for machine learning and artificial intelligence. It was developed by Google and provides a range of tools for building and training various types of neural

networks, including convolutional neural networks, recurrent neural networks, and transformers. TensorFlow is highly scalable and efficient, and allows for distributed training across multiple CPUs and GPUs. It also provides a range of high-level APIs and tools for model deployment, including TensorFlow Serving and TensorFlow Lite for mobile and embedded devices. TensorFlow has a large community of developers and users, and is widely used in industry and academia for a variety of applications, including computer vision, natural language processing, and robotics.

4.9.4 Keras

Keras is an open-source Python deep learning library that provides a high-level API for building and training neural networks. It is designed to be easy to use and user-friendly, allowing developers to quickly prototype and experiment with different neural network architectures. Keras supports various neural network types, including convolutional neural networks, recurrent neural networks, and transformers. It also provides a range of tools for data preprocessing, model evaluation, and visualization. Keras can be run on top of various backend engines, including TensorFlow, Microsoft Cognitive Toolkit, and Theano. It is widely used in industry and academia for a variety of applications, including image recognition, natural language processing, and time-series forecasting.

4.9.5 GPU(T4)

The T4 is a graphics processing unit (GPU) developed by NVIDIA, designed for use in machine learning and artificial intelligence workloads. It is based on NVIDIA's Turing architecture and provides a range of features optimized for deep learning, including tensor cores for fast matrix operations and support for mixed-precision training to reduce memory usage and increase throughput. The T4 GPU is particularly well-suited for inference workloads, where trained models are used to make predictions on new data, due to its high performance and low power consumption. It is commonly used in cloud computing environments, where it can be used to accelerate deep learning workloads through services such as Google Cloud AI Platform and Amazon SageMaker.

4.9.6 Sklearn

The scikit-learn (also known as sklearn) is an open-source Python library for machine learning and data mining. It provides a range of tools for building and evaluating various types of machine learning models, including classification, regression, clustering, and dimensionality reduction. The library includes a wide range of algorithms and methods, such as decision trees, support vector machines, k-nearest neighbors, random forests, and more. In addition, scikit-learn provides tools

for data preprocessing, feature selection, and model evaluation, such as cross-validation and grid search. The library is well-documented and widely used in industry and academia for a variety of applications, including image recognition, natural language processing, and fraud detection.

5 Experimental Results

The hyper-parameters used in Meta Classifier are

1. Learning rate (α) = 0.001
2. Epsilon (ε) = 10^{-8}
3. Epochs = 20
4. Adam Optimizer (Exponential Decay Rates)
    - $\beta 1 = 0.9$
    - $\beta 2 = 0.999$

Window Size	Filter, Kernel Size (1)	Filter, Kernel Size (2)	Dropout (1)	Dropout (2)	Dense (1), (2)	Avg Train Acc	Avg Test Acc	Test MCC
21	64,1	64,1	0.5	0.5	32,8	0.85	0.73	0.47
21	128,1	128,1	0.4	0.3	64,8	0.85	0.75	0.50
21	144,1	128,1	0.5	0.5	128,16	0.88	0.78	0.54
31	64,1	64,1	0.6	0.3	32,8	0.85	0.74	0.50
31	128,1	128,1	0.5	0.5	64,8	0.88	0.77	0.53
31	144,1	128,1	0.5	0.5	128,16	0.90	0.79	0.55
41	64,1	64,1	0.6	0.6	32,8	0.82	0.73	0.49
41	128,1	128,1	0.5	0.5	64,8	0.83	0.74	0.50
41	144,1	128,1	0.5	0.5	128,16	0.88	0.77	0.53

Table 3: Experimental Results of Meta Classifier using ProtT5+ProtBert

6 Result and Analysis

6.1 Implementation Details

For the implementation of our proposed network we use TensorFlow as our deep learning framework. We use the Adam optimizer to train the model with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$ along with a learning rate of 0.001. Each dataset was trained for 20 epochs. We did every experiment and training described here using Kaggle with either of the Nvidia T4 GPUs.

6.2 Evaluation Metrics

For comparing with previous works and evaluating our results, we used the following metrics:

1. Accuracy: Accuracy measures the total number of predictions a model gets right. Accuracy is not a good metric when the classes are imbalanced.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Recall/ Sensitivity/TPR: Recall calculates the percentage of actual positives a model correctly identified. Sensitivity is the metric that evaluates a model's ability to predict true positives of each available category.

$$Recall = \frac{TP}{TP + FN}$$

3. Precision: Precision measures how precise a model is in predicting positive labels. It answers the questions that out of how many times a model predicted positive, how often was it correct?

$$Precision = \frac{TP}{TP + FP}$$

4. F1-score: F1-score is the harmonic mean of precision and recall

$$F1\text{-}score = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

5. Matthews Correlation Coefficient(MCC): MCC is a statistical metric that produces a high score only if the prediction obtained good results in all four categories of confusion matrix (true positives, false negatives, true negatives and false positives)

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

6. Specificity: Specificity is the metric that evaluates a model's ability to predict true negatives of each available category.

$$Specificity = \frac{TN}{TN + FP}$$

7. AUC-ROC:AUC-ROC curve is an efficienct performance measurement for the classification of problems at various threshold levels. AUC symbolizes the degree of separability, whereas ROC is a probability curve. It reveals how well the model can differentiate across classes. The model performs well if it has higher AUC. So, if the AUC is higher, the model is more accurate at differentiating between succinylated and non-succinylated sites.

   The ROC curve is plotted with TPR against FPR, where

$$TPR = \frac{TP}{TP + FN} \quad \text{and} \quad FPR = \frac{FP}{TN + FP}$$

8. PR AUC: Area Under the Precision Recall Curve(PR AUC) combines precision and recall in a single visualization. The higher on y-axis the curve the better the model performance.

9. t-SNE: t-Distributed Stochastic Neighbor Embedding (t-SNE)[39] is a technique to visualize high-dimensional data by giving each datapoint a location in two or three-dimensional map.

6.3 t-SNE Visualization

To reduce the dimensionality of high-dimensional datasets, t-SNE (t-Distributed Stochastic Neighbor Embedding)[39] is often used. t-SNE attempts to move high-dimensional data into a low-dimensional space while retaining as much of the data's original structure as feasible.

The underlying theory of t-SNE is based on the idea that points that are close together in a high-dimensional space should be represented by points close to each other in a low-dimensional space, and points that are far apart in a high-dimensional space should be represented by points far away from each other in a low-dimensioanl space.

PCA(Principal Component Analysis) can also be used to visualize high-dimensional data by identifying the directions in which the data varies the most and by projecting the data onto a lower-dimenstional space defined by these components[5]. PCA can, however, omit significant aspects of the data since it is not effective at capturing nonlinear relationships between variables. PCA can also be susceptible to outliers and miss complex correlations between variables.

On the other hand, T-SNE is a nonlinear method that works well for visualizing and examining complex data that has nonlinear interactions between the variables. T-SNE functions by maintaining the local structure of the data, which means that close-by high-dimensional points are mapped

to close-by low-dimensional points regardless of their separation in the original space. This can show the data in greater depth and provide a more accurate view by capturing more subtle correlations between variables.

The t-SNE (t-Distributed Stochastic Neighbor Embedding) equation is given by:

$$p_{i|j} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq l} \exp(-||x_k - x_l||^2/2\sigma_k^2)}$$

where, $p_{i|j}$ is the conditional probability of point j given point i, x_i is the i-th high-dimensional input data point,and σ_i is the variance of the Gaussian distribution used to define the similarity between point i and other points in the high-dimensional space.

The t-SNE (t-Distributed Stochastic Neighbor Embedding) objective function is given by:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

where, $KL(P||Q)$ is the Kullback-Leibler divergence between the joint probabilities P and Q, p_{ij} is the conditional probability of point j given point i, and q_{ij} is the corresponding probability in the low-dimensional space.

6.4 Ablation Study

We conducted various ablation studies to evaluate the effectiveness of our proposed architecture.

6.4.1 Experiment with different Architecture

In this experiment, we started with ProtBert pre-trained embedding followed by ProtT5 and then by Meta classifier. We measured the performance of all three models by varying the window size using stratified tenfold cross-validation.

Method	Window size	Parameters	Accuracy	MCC
ProtBert 1D-CNN	21	249K	0.72±0.01	0.45±0.02
	31	**290K**	**0.75±0.01**	**0.49±0.01**
	41	331K	0.73±0.01	0.47±0.02
ProtT5 1D-CNN	21	249K	0.77±0.01	0.50±0.02
	31	**290K**	**0.78±0.01**	**0.52±0.01**
	41	331K	0.77±0.01	0.51±0.02
MetaClassifier using ProtT5	21	498K	0.78±0.01	0.54±0.02
+ ProtBert	**31**	**580K**	**0.79±0.01**	**0.55±0.01**
	41	662K	0.77±0.01	0.53±0.02

Table 4: Evaluating Result on UniProt Dataset with Different Architecture and Window Size
Best Result In Bold

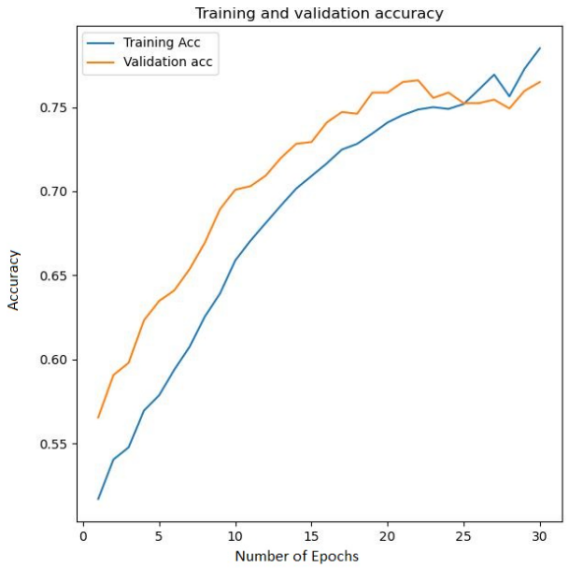
We observed that every model performed best on window size 31. Among the three different architectures, ProtBert 1D-CNN performed poorly while using the stacking ensemble method on the output of each ProtBERT, and ProtT5 generated by 1D-CNN has shown to be the best model in every metric.

Since ProtT5 was trained on a bigger and more varied collection of protein sequences than ProtBERT, it performed better than protBert. Additionally, ProtT5 is built on the T5 transformer architecture, which is more adaptable and flexible than the BERT transformer used in ProtBERT, making ProtT5 easily customizable for a variety of protein sequence analysis tasks.

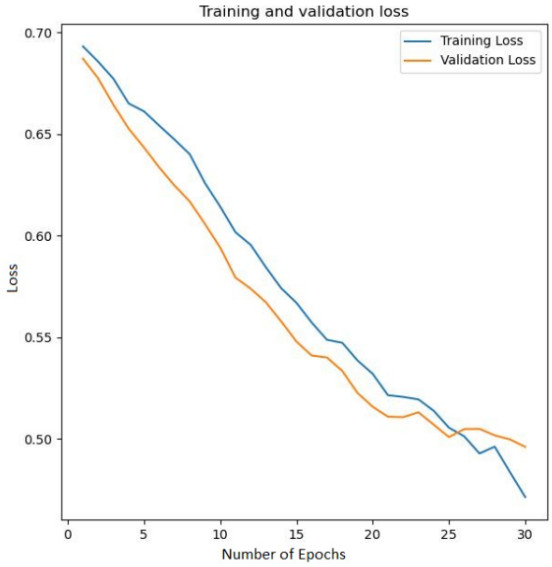
6.4.2 Plots

We also compared the performance of these various models using accuracy, loss, and ROC curve for the stratified tenfold cross-validation. The acc, loss, ROC, and Pr-Auc curves of the ProtBert-based model, PortT5-based model, and meta-model of ProtT5-based model and ProtBert-based model are shown in figures below.

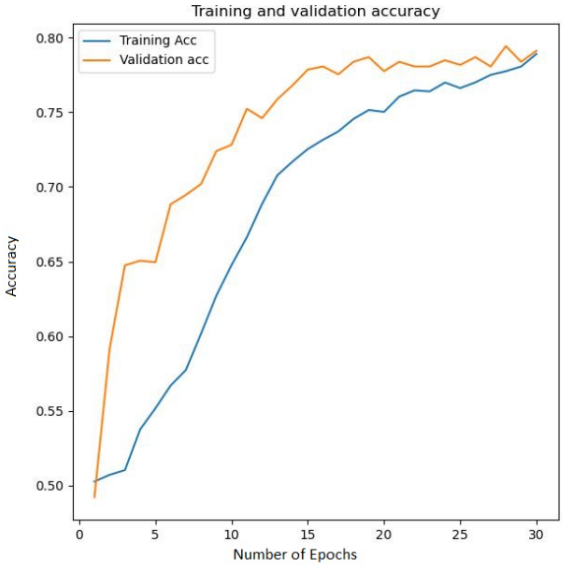
6.4.2.1 Training and Validation Accuracy/Loss plot



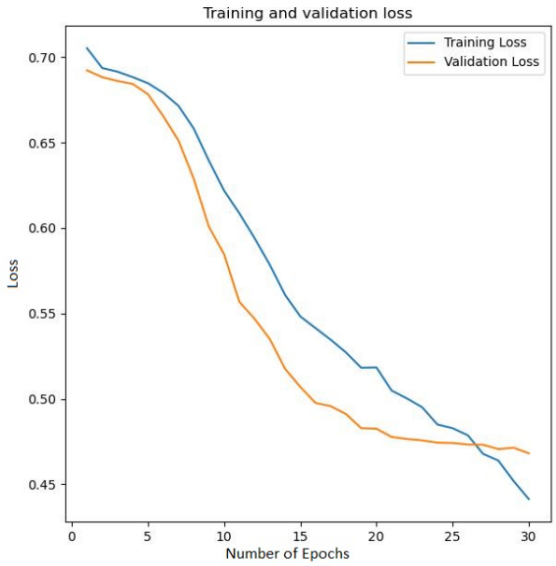
(a) ProtBERT Accuracy



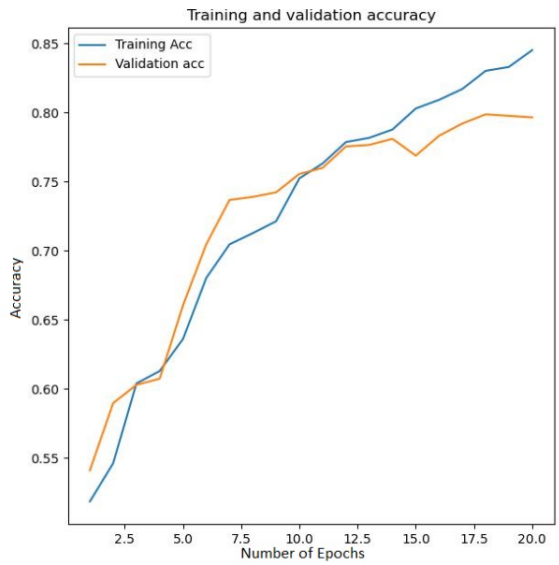
(b) ProtBERT Loss



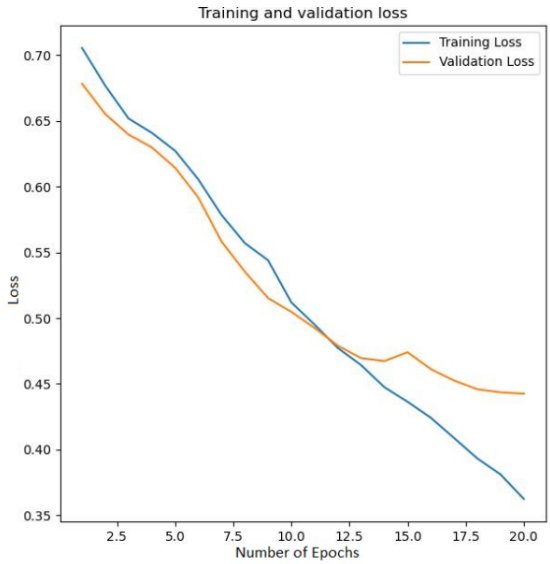
(c) ProtT5 Accuracy



(d) ProtT5 Loss



(e) Meta Classifier Accuracy

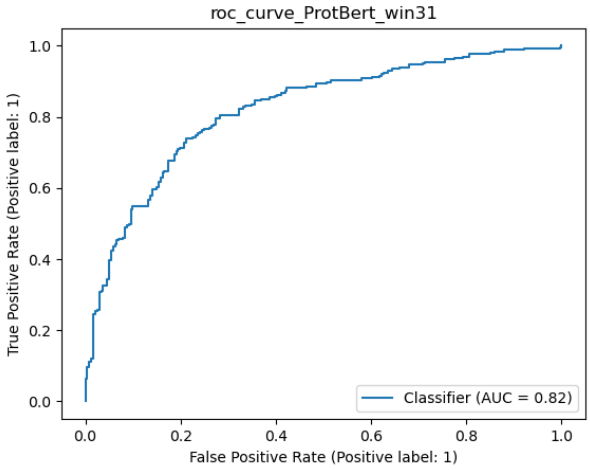


(f) Meta Classifier Loss

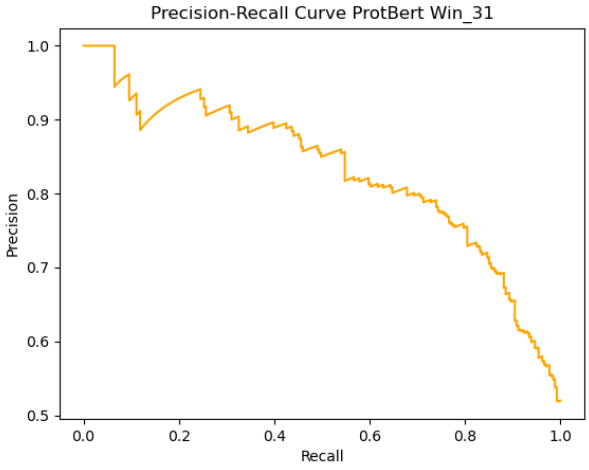
Figure 15: Training and validation accuracy/loss for different architectures

Fig 15 shows the training and accuracy curve for our best models in different architectures in window size 31.To address the problem of overfit we check for additional 5 epochs to determine improvement in performance and then restore the best weights if no improvement was spotted. Validation and training accuracy increased with the increase in number of epochs, which indicates that our model was performing better over each epoch. Further more analysing the loss graph, validation and training loss decreases over epoch, which shows that our model is fitting well with both training data and validation data.

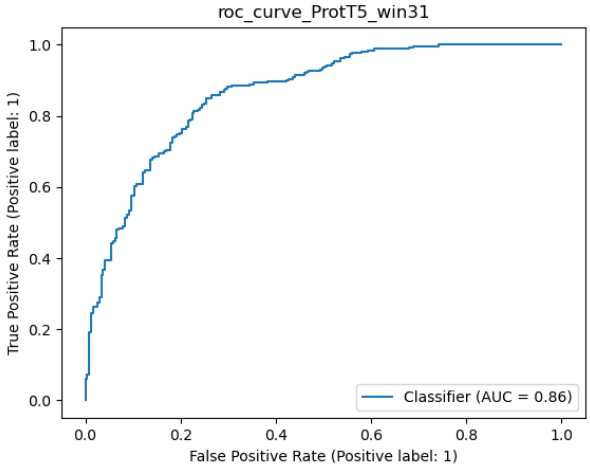
6.4.2.2 ROC/Precision curve plot



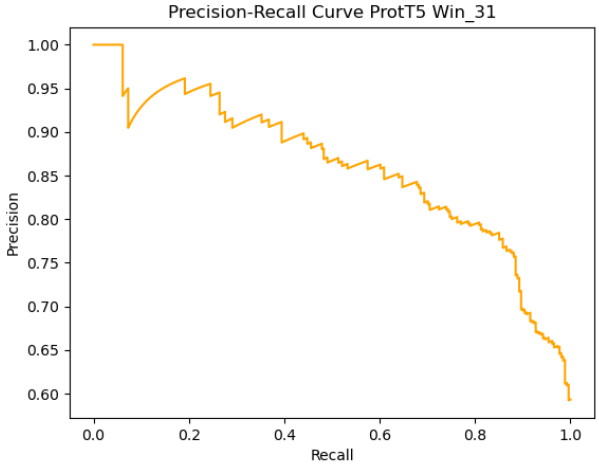
(a) 1D-CNN ProtBERT



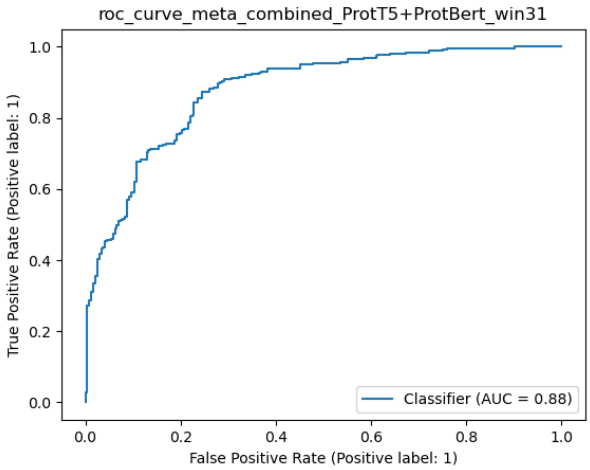
(b) 1D-CNN ProtBERT



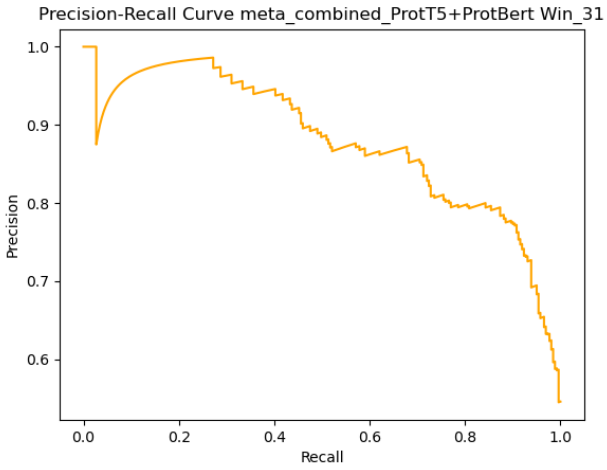
(c) 1D-CNN ProtT5



(d) 1D-CNN ProtT5



(e) Meta Classifier



(f) Meta Classifier

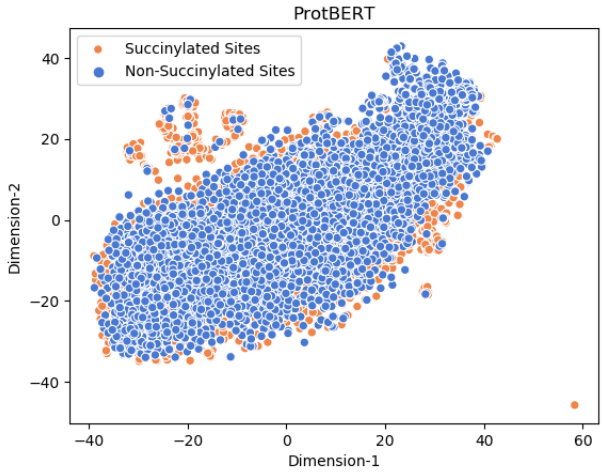
Figure 16: ROC and PR curves for different architectures

Fig 16 shows the AUC-ROC curve and Pr-AUC curve for our model. Our meta classifier model has the highest area under the curve which indicates that it is able to classify most of our prediction correctly than the other two model.

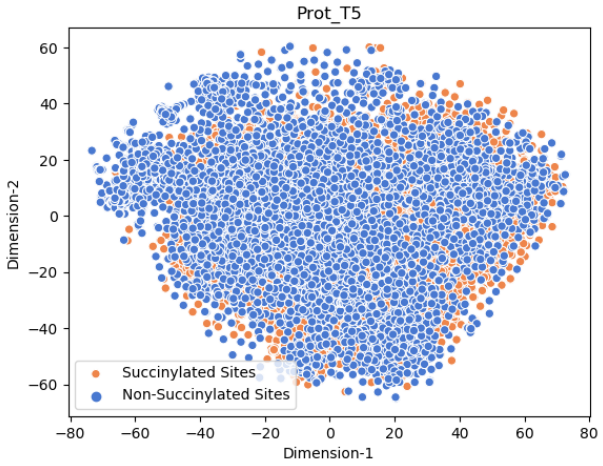
ROC curves is plotted on the different threshold value. When we look closely to our AUC-ROC and Pr-AUC of our meta classifier, the optimial values of threshold should lie between 0.3-0.4. By taking threshold at different settings, we verified the optimal threshold to be 0.38 for best F1-score.

6.5 Further Analysis

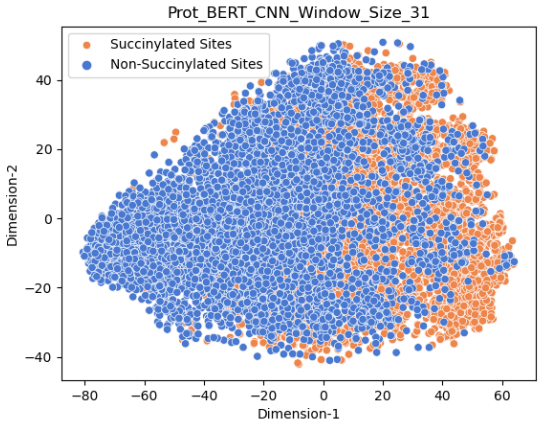
We created t-SNE plots in R^2 space for the original data and features discovered through supervised word embedding, ProtBert-based 1D-CNN, ProtT5-based 1D-CNN, and the MetaLMsuccinylationSite(MLMSS) model to gain understanding into the foundation for model's enhanced performance.The features extracted from the embedded vector space, in contrast to the original data, reveal the development of distinct clusters of succinylated (orange data points) and non-succinylated (blue data points) sites. The boundary of separation was more distinct in the MetaLMsuccinylationSite model as the ensemble model exhibit more prominent features.



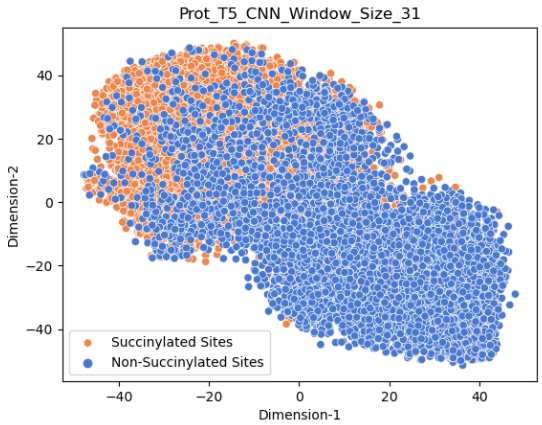
(a) ProtBert Embedding



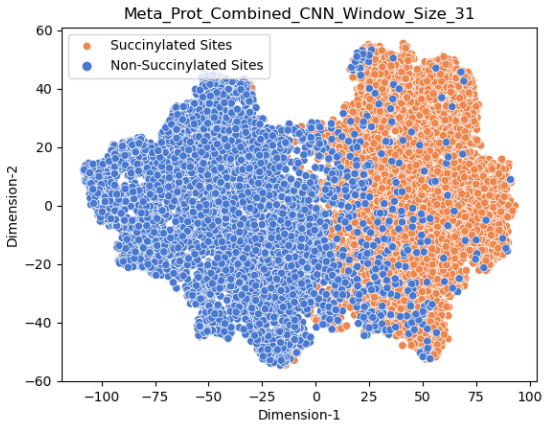
(b) ProtT5 Embedding



(c) After training ProtBert embedding with 1D-CNN



(d) After training ProtT5 embedding with 1D-CNN



(e) After training the meta-classifier

Figure 17: t-SNE to visualize high dimensional embedding where blue dots represent the non-succinylated sites and orange dots represent the succinylated sites.

Fig 17 shows the t-SNE plot for our different models. Fig 17a and Fig 17b represents the clustering of our succinylated(orange) and non-succinlyated(blue) site of ProtBert and ProtT5 respectively. From the figure 17a and 17b, we observe that the classification is not possible only through the use of ProtBert and ProtT5 alone. But after applying 1-D CNN to ProtBert and ProtT5 model fig 17c and 17d, we begin to observe the separation of the clustering. Furthermore, when using our meta classifier (combination of ProtBert + ProtT5) as seen in Fig 17e separation between clustering and bounder became more clearer and visible.

6.6 Comparison of our model with other predictors

To compare our model with other existing models, we tested our model with the independent test set described in Table 4 and compare performance metrics such as accuracy, MCC, sensitivity, specificity and g-mean (shown in Table 5).It should be noted that all of the evaluated models used the identical training and test sets.

Tool	ACC	MCC	Sn	Sp	g-mean
iSuc-PseAAC[5]	0.83	0.01	0.12	0.89	0.33
iSuc-PseOpt[6]	0.72	0.04	0.30	0.76	0.48
pSuc-Lys[7]	0.78	0.04	0.22	0.83	0.43
SuccineSite[9]	0.84	0.19	0.37	0.88	0.57
SuccineSite2.0[40]	0.85	0.26	0.40	**0.88**	0.63
GPSuc[12]	**0.85**	0.30	0.50	0.88	0.66
PSuccE[41]	0.85	0.20	0.38	0.89	0.58
DeepSuccinylSite[14]	0.70	0.27	**0.79**	0.69	0.74
LMSuccSite[18]	0.79	**0.36**	**0.79**	0.79	**0.79**
Our model	0.76	0.32	0.78	0.75	0.76

Table 5: Performance comparison of Our Model with other Predictors.

As observed in Table 5, our model (MLMSS) performs well on benchmark datasets and is comparable in nearly all metrices except MCC with the state of the art method.

7 Conclusion

In this project, to address the problem of predicting lysine succinylation in protein sequences, we have designed a model using protein language model and deep learning and compared with other methods. We conducted various ablation studies to analyse the effectiveness of different language model and different deep learning approaches. It was found that 1D-based CNN outperforms other approaches. Stacking the features of different language model helped us to increase the overall performance. On comparing our result on independent benchmarking dataset with other state-of-the-art methods, our approach was comparable in almost all the evaluation metrics. The performance of individual language model was also compared and it was found that ProtBert performed quite poorly in comparison with ProtT5. Even with small dataset, language model-based approach enabled transfer learning from large dataset which allowed for useful feature extraction from our

small dataset. The stacking ensemble improved our result which proved a single language model is not able to extract all the features and that it is better to combine multiple models. Although our model was outperformed by combination of supervised word embedding and language model, with time and increasing research on language model, language-model based approach is expected to outperform all other approaches. Also the features extracted from language model in combination with very simple architectures perform very well.

7.1 Future Enhancement

We have mentioned several limitations of the project. Most of the limitations can be overcome with the use of a larger dataset which is simply not possible in this case. Our work included working with just two protein language models. But there are several other models that might give better representation of sequence for succinylation prediction task. Several protein language models have been proposed for extracting features from a very large pool of protein sequences.Evolutionary Scale Modeling (ESM) model have been performing well on other PTM tasks which might result in better performance in Lysine PTM as well. So, relative performance analysis of various LM and their combination for prediction of lysine succinylation sites can be beneficial.

References

[1] Yun Yang and Gary E Gibson. Succinylation links metabolism to protein functions. *Neurochemical research*, 44:2346–2359, 2019.

[2] Quan Liu, Heming Wang, He Zhang, Liyan Sui, Letian Li, Wang Xu, Shouwen Du, Pengfei Hao, Yuhang Jiang, Jing Chen, et al. The global succinylation of sars-cov-2–infected host cells reveals drug targets. *Proceedings of the National Academy of Sciences*, 119(30):e2123065119, 2022.

[3] Md Mehedi Hasan and Mst Shamima Khatun. Prediction of protein post-translational modification sites: an overview. *Ann Proteom Bioinform*, 2:049–57, 2018.

[4] Christina Lind, Robert Gerdes, Ylva Hamnell, Ina Schuppe-Koistinen, Helena Brockenhuus von Löwenhielm, Arne Holmgren, and Ian A Cotgreave. Identification of s-glutathionylated cellular proteins during oxidative stress and constitutive metabolism by affinity purification and proteomic analysis. *Archives of biochemistry and biophysics*, 406(2):229–240, 2002.

[5] Yan Xu, Ya-Xin Ding, Jun Ding, Ya-Hui Lei, Ling-Yun Wu, and Nai-Yang Deng. isuc-pseaac: predicting lysine succinylation in proteins by incorporating peptide position-specific propensity. *Scientific reports*, 5(1):1–6, 2015.

[6] Jianhua Jia, Zi Liu, Xuan Xiao, Bingxiang Liu, and Kuo-Chen Chou. isuc-pseopt: identifying lysine succinylation sites in proteins by incorporating sequence-coupling effects into pseudo components and optimizing imbalanced training dataset. *Analytical biochemistry*, 497:48–56, 2016.

[7] Jianhua Jia, Zi Liu, Xuan Xiao, Bingxiang Liu, and Kuo-Chen Chou. psuc-lys: predict lysine succinylation sites in proteins with pseaac and ensemble random forest approach. *Journal of theoretical biology*, 394:223–230, 2016.

[8] Wanshan Ning, Haodong Xu, Peiran Jiang, Han Cheng, Wankun Deng, Yaping Guo, and Yu Xue. Hybridsucc: a hybrid-learning architecture for general and species-specific succinylation site prediction. *Genomics, proteomics & bioinformatics*, 18(2):194–207, 2020.

[9] Md Mehedi Hasan, Shiping Yang, Yuan Zhou, and Md Nurul Haque Mollah. Succinsite: a computational tool for the prediction of protein succinylation sites by exploiting the amino acid patterns and properties. *Molecular BioSystems*, 12(3):786–795, 2016.

[10] Abdollah Dehzangi, Yosvany López, Sunil Pranit Lal, Ghazaleh Taherzadeh, Jacob Michaelson, Abdul Sattar, Tatsuhiko Tsunoda, and Alok Sharma. Pssm-suc: Accurately predicting succinylation using position specific scoring matrix into bigram for feature extraction. *Journal of theoretical biology*, 425:97–102, 2017.

[11] Abdollah Dehzangi, Yosvany López, Sunil Pranit Lal, Ghazaleh Taherzadeh, Abdul Sattar, Tatsuhiko Tsunoda, and Alok Sharma. Improving succinylation prediction accuracy by incorporating the secondary structure via helix, strand and coil, and evolutionary information from profile bigrams. *PloS one*, 13(2):e0191900, 2018.

[12] Md Mehedi Hasan and Hiroyuki Kurata. Gpsuc: Global prediction of generic and species-specific succinylation sites by aggregating multiple sequence features. *PloS one*, 13(10):e0200283, 2018.

[13] Kai-Yao Huang, Justin Bo-Kai Hsu, and Tzong-Yi Lee. Characterization and identification of lysine succinylation sites based on deep learning method. *Scientific reports*, 9(1):1–15, 2019.

[14] Niraj Thapa, Meenal Chaudhari, Sean McManus, Kaushik Roy, Robert H Newman, Hiroto Saigo, and Dukka B Kc. Deepsuccinylsite: a deep learning based approach for protein succinylation site prediction. *BMC bioinformatics*, 21(3):1–10, 2020.

[15] Guohua Huang, Qingfeng Shen, Guiyang Zhang, Pan Wang, and Zu-Guo Yu. Lstmcnnsucc: a bidirectional lstm and cnn-based deep learning method for predicting lysine succinylation sites. *BioMed Research International*, 2021, 2021.

[16] Huiqing Wang, Hong Zhao, Zhiliang Yan, Jian Zhao, and Jiale Han. Mdcan-lys: A model for predicting succinylation sites based on multilane dense convolutional attention network. *Biomolecules*, 11(6):872, 2021.

[17] Jianhua Jia, Genqiang Wu, and Wangren Qiu. psuc-ffsea: predicting lysine succinylation sites in proteins based on feature fusion and stacking ensemble algorithm. *Frontiers in Cell and Developmental Biology*, 10, 2022.

[18] Suresh Pokharel, Pawel Pratyush, Michael Heinzinger, Robert H Newman, and Dukka B Kc. Improving protein succinylation sites prediction using embeddings from protein language model. *Scientific Reports*, 12(1):16933, 2022.

[19] Patricio Cerda, Gaël Varoquaux, and Balázs Kégl. Similarity encoding for learning with dirty categorical variables. *Machine Learning*, 107(8-10):1477–1494, 2018.

[20] Muhammad Yaseen Khan, Abdul Qayoom, Muhammad Suffian Nizami, Muhammad Shoaib Siddiqui, Shaukat Wasi, and Syed Muhammad Khaliq-ur-Rahman Raazi. Automated prediction of good dictionary examples (gdex): a comprehensive experiment with distant supervision, machine learning, and word embedding-based deep learning techniques. *Complexity*, 2021:1–18, 2021.

[21] Congcong Wang, Paul Nulty, and David Lillis. A comparative study on word embeddings in deep learning for text classification. In *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, pages 37–46, 2020.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[23] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[24] Alexander M Rush. The annotated transformer. In *Proceedings of workshop for NLP open source software (NLP-OSS)*, pages 52–60, 2018.

[25] Aaron Nicolson and Kuldip K Paliwal. Masked multi-head self-attention for causal speech enhancement. *Speech Communication*, 125:80–96, 2020.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[27] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghassan Rihawi, Yutong Wang, Landon Jones, Tyson Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. *arXiv preprint arXiv:2007.06225*, 2020.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Regularization for deep learning. *Deep learning*, pages 216–261, 2016.

[30] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[32] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[33] Yuan-Pin Lin and Tzyy-Ping Jung. Improving eeg-based emotion classification using conditional transfer learning. *Frontiers in human neuroscience*, 11:334, 2017.

[34] Nobuaki Kimura, Ikuo Yoshinaga, Kenji Sekijima, Issaku Azechi, and Daichi Baba. Convolutional neural network coupled with a transfer-learning approach for time-series flood predictions. *Water*, 12(1):96, 2019.

[35] Reem Salman, Ayman Alzaatreh, Hana Sulieman, and Shaimaa Faisal. A bootstrap framework for aggregating within and between feature selection methods. *Entropy*, 23(2):200, 2021.

[36] Cheng Ju, Aurélien Bibaut, and Mark van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of Applied Statistics*, 45(15):2800–2818, 2018.

[37] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[38] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[39] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[40] Md Mehedi Hasan, Mst Shamima Khatun, Md Nurul Haque Mollah, Cao Yong, and Dianjing Guo. A systematic identification of species-specific protein succinylation sites using joint element features information. *International journal of nanomedicine*, 12:6303, 2017.

[41] Qiao Ning, Xiaosa Zhao, Lingling Bao, Zhiqiang Ma, and Xiaowei Zhao. Detecting succinylation sites from protein sequences using ensemble support vector machine. *BMC bioinformatics*, 19(1):1–9, 2018.