Deep Learning for Taxonomy Prediction

Shreyas Ramesh

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Madhav Marathe, Chair
Andrew Warren, Co-chair
Anil Vullikanti

April 25, 2019
Blacksburg, Virginia

Keywords: taxonomy prediction, convolutional neural networks, hierarchical prediction,
cnn, taxonomic binning

Deep Learning for Taxonomy Prediction

Shreyas Ramesh

(ABSTRACT)

The last decade has seen great advances in Next-Generation Sequencing technologies, and, as a result, there has been a rise in the number of genomes sequenced each year. In 2017, there were as many as 10,000 new organisms sequenced and added into the RefSeq Database (Nasko et al. [20]). Taxonomy prediction is a science involving the hierarchical classification of DNA fragments up to the rank species. In this research, we introduce **P**redicting **Link**ed **O**rganisms, **Plinko**, for short. Plinko is a fully-functioning, state-of-the-art predictive system that accurately captures DNA - Taxonomy relationships where other state-of-the-art algorithms falter. Plinko leverages multi-view convolutional neural networks and the predefined taxonomy tree structure to improve multi-level taxonomy prediction. In the Plinko strategy, each network takes advantage of different word usage patterns corresponding to different levels of evolutionary divergence. Plinko has the advantages of relatively low storage, GPGPU parallel training and inference, making the solution portable, and scalable with anticipated genome database growth. To the best of our knowledge, Plinko is the first to use multi-view convolutional neural networks as the core algorithm in a compositional, alignment-free approach to taxonomy prediction.

Deep Learning for Taxonomy Prediction

Shreyas Ramesh

(GENERAL AUDIENCE ABSTRACT)

Taxonomy prediction is a science involving the hierarchical classification of DNA fragments up to the rank species. Given species diversity on Earth, taxonomy prediction gets challenging with (i) increasing number of species (labels) to classify and (ii) decreasing input (DNA) size. In this research, we introduce **P**redicting **Link**ed **O**rganisms, **Plinko**, for short. Plinko is a fully-functioning, state-of-the-art predictive system that accurately captures DNA - Taxonomy relationships where other state-of-the-art algorithms falter. Three major challenges in taxonomy prediction are (i) large dataset sizes (order of 10^9 sequences) (ii) large label spaces (order of 10^3 labels) and (iii) low resolution inputs (100 base pairs or less). Plinko leverages multi-view convolutional neural networks and the pre-defined taxonomy tree structure to improve multi-level taxonomy prediction for hard to classify sequences under the three conditions stated above. Plinko has the advantage of relatively low storage footprint, making the solution portable, and scalable with anticipated genome database growth. To the best of our knowledge, Plinko is the first to use multi-view convolutional neural networks as the core algorithm in a compositional, alignment-free approach to taxonomy prediction.

Acknowledgments

This study is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the Army Research Office (ARO) under cooperative Agreement Number W911NF-17-2-0105A. The author would also like to thank Dr. Jacob Porter for his help in motivating the experiments, generating data sets that were of crucial importance to conduct the experiments, outlining the chapters within this report and supporting the development of the Plinko software.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN Convolutional Neural Networks

DNA Deoxyribonucleic Acid

GPGPU General Purpose Graphics Processing Unit

MVCNN Multi-View Convolutional Neural Networks

NGS Next-Generation Sequencing

Chapter 1

Introduction

1.1 Overview of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 provides the computational background required for this thesis. Chapter 3 talks about the input remapping techniques, results and takeaways. Chapter 4 describes the Plinko n-ary tree architecture and experiments that showcase important properties of Plinko. Chapter 5 provides insight into Plinko's inference characteristics and concludes this thesis.

1.2 Motivation

State-of-the-art taxonomy prediction algorithms today use either compositional or alignment based methods to identify the source organism from input DNA fragments. Exact match, pseudo alignment algorithms such as kraken2 (Wood and Salzberg [27]) are compositional methods that build an indexed sequence data structure to differentiate between the many taxonomic divisions. With the Earth's large unsampled genetic reservoir, programmatic, look-up based taxonomy prediction faces the problem of having a varying and ever-increasing number of genomes to potentially classify. Algorithms such as kraken2 are especially susceptible to this problem since they need to store all variations of a sequence in order to reproduce its taxonomic assignment. Recently, Opal (Luo et al. [16]), overcame

the challenge of storing exact k-mers, by defining a family of hash functions that sample uniformly, with even coverage, a predefined number of nucleotides from each k-mer. With Opal, it was now possible to incorporate longer k-mers that were insensitive to small sequence variations, and at the same time sensitive to long range patterns within DNA fragments.

In terms of storage requirements, kraken2 has to store both the index and taxonomy structure in memory to perform inference. Opal also has to store it's hash table (mapping k-mers in 4^k dimensions to k-mer counts), and one model for each label, in memory. Currently, these memory requirements restrict usage of Opal and kraken2 to high performance, high memory computer systems. Plinko's models on the other hand, have the advantage being almost the same size, regardless of the number of training samples, number of labels or taxonomy rank. Plinko's level-wise or model-wise load settings gives the user flexibility to trade off between inference speed and memory usage. This enables Plinko to be used on both the most powerful computing environments and on commodity computers.

1.3 Context of the Study

Machine learning models are naturally well suited to solve taxonomy prediction as they can learn to probabilistically distinguish between the many types of sequence variations. Given enough data, deep models can automatically capture important features (Setiono and Liu [25]). It is our hope that deep neural networks will help us to build robust taxonomy prediction systems, and also help show that there are features beyond sequence similarity and k-mer counts that help us determine the identity of microbes.

We focus our efforts in training Convolutional Neural Networks (CNNs) (LeCun et al. [14]), which are deep directed graphs constituting multiple layers of processing kernels that each learn the spatial features from inputs by composing multiple linear and non-linear transformations. Layering functions this way enables CNN models to refine features representations with each successive layer. In the past, CNNs have been successfully deployed in disciplines such as computer vision (Krizhevsky et al. [13]) and natural language processing (Kim [12]), where there is inherent spatial structure. In this research we investigate whether CNNs can be just as successful in the field of taxonomy prediction with DNA fragments as input.

1.4 Objectives and Contributions

We present a new deep learning design that leverages the taxonomy tree structure: An n-ary decision tree with multi-view CNNs at each node to predict taxonomy from DNA fragments. Given that our training DNA fragments are short (100 base pairs), we show that input remapping techniques can extract additional information encoded within DNA. We investigate nucleotide k-mer embeddings, leverage properties of Open Reading Frames inherent within the DNA and make use of Hilbert space-filling curves to map sequential k-mer indices into 2-dimensional co-ordinates, while maintaining k-mer spatial locality.

In our experiments, we compare Plinko with kraken2, a traditional non-learning based compositional, aligment-free classification tool, and Opal, an SVM based compositional, aligment-free classifier that recently achieved state-of-the-art precision and sensitivity in DNA read classification. It is important to note that we do not compare Plinko to alignment based methods such as Kaiju (Menzel et al. [18]) or Centrifuge (Kim et al. [11]). Our intention is to showcase the unique properties that compositional approaches based on neu-

ral networks can bring to taxonomy classification. Initial experiments show that Plinko is portable due to it's model size, and at the same time offers the highest classification performance (in terms of average F-measure) when predicting the taxonomy of novel organisms from coding DNA fragments.

Chapter 2

Computational Background

2.1 Introduction

Deep neural networks are modern extensions of artificial neural networks that are themselves loosely inspired from intelligent biological systems (Belatreche [4]) consisting of neurons, synapses and activation signals. The idea behind neural networks is: layer wise interconnection of information processing units (neurons) can be leveraged to approximate functional relationships between inputs and labels.

Early research on neural networks struggled for multiple reasons. (i) Models trained on small data sets were highly prone to overfitting. This meant that neural networks were not reliable on novel, unseen data. In the few cases where large input data sets were available, there were other issues. (ii) Apart from the high computational costs for training, the models easily found themselves (iii) stuck at local minima, and (iv) were highly prone to the vanishing gradient problem, due to which the back-propagation algorithm failed to propagate error from output layer to the earlier layers. The overall effect was that adding more layers to the network was practically useless, and the neurons were never updated from their previous states.

Advances in neural networks came in many forms. In 2006, a seminal paper (Bengio et al. [5]) proposed a greedy layer-wise unsupervised training strategy and confirmed that unsupervised pre-training mostly helps the optimization by initializing weights in a region near a good local minimum. This work helped alleviate difficulties when using deep models on smaller labeled data sets, and (re)ignited hope for deep neural network research. The ImageNet competition (Russakovsky et al. [24]) also motivated a large step forward by helping researchers to come up with novel deep learning architectures and more importantly, promoted the usage of GPGPUs for faster parallel training and inference. Researchers participating in the competition also made important algorithmic contributions in regularization to reduce overfitting of neural network models. With help from both the academic community and the industry, deep neural networks have now been used for feature learning (Dosovitskiy et al. [9]), image recognition and image segmentation (Goodfellow et al. [10]), end-to-end speech recognition (Amodei et al. [1]), machine translation (Costa-jussà [7]) and bioinformatics (Lee et al. [15]). It goes without saying that for deep learning research, GPGPUs and the cloud have been major enablers, as they allowed experiments to run within a reasonable amount of time and cost. Lastly, the availability of data through the internet allowed us to come up with solutions to problems that could be (i) formulated as an intelligence problem, and (ii) could be extracted from data alone.

In this section, we give a brief introduction to four important components of deep neural networks that are used for text input classification today. We begin this chapter with the simplest component of all, the artificial neuron. Building on the artificial neuron, we describe the artificial neural network and word embeddings. We end this chapter with a description of convolutional neural networks.

2.2 Artificial Neuron

Mathematically, the neuron is a function ($\mathbb{R}^n \rightarrow \mathbb{R}$) that maps inputs to a real or binary output. Computationally, the neuron accepts a group of weighted inputs, applies an activation function, and returns an output. Given a vector of inputs $x = (1, x_1, x_2, ...x_n)$, the neuron's output is calculated as $y = f(z) = f(w^T x)$, where, $z = w^T x$ is the weighted sum of the inputs, w is a weight vector, and f is the activation function.

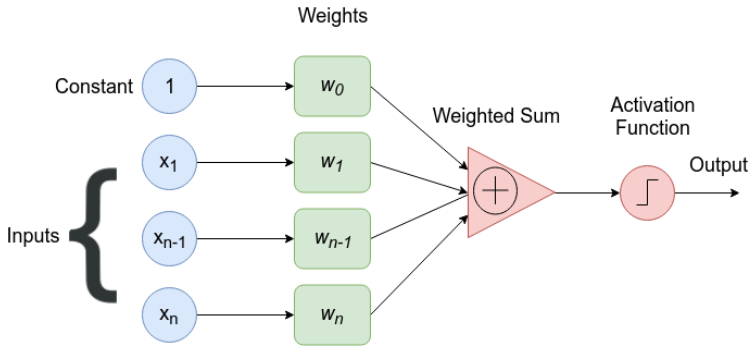

Figure 2.1: A simple artificial neuron with n inputs, a bias term, n synapse weights and a step function as the activation function.

Since neurons are arranged in layers, inputs to a neuron could either be a set of features from training data or the outputs from a previous layer's neurons. Weights, typically real, floating values are applied to inputs as they travel along edges (called synapses) to reach the neuron. The neuron applies activation functions to the weighted sum of the inputs and the process repeats for neurons in the next layer.

A neuron may choose to apply one of many activation functions. For example:

1. Step function: $f(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$

2. Linear function: $y = f(z) = z$

3. Rectified linear function (ReLU): $y = max(0, z)$

4. Sigmoid function: $\frac{1}{1+e^{-z}}$

Step activation functions make binary decisions at $z = 0$ and are not differentiable. Linear activation functions act as linear classifiers where the weights represent a hyper plane in the input space. A simple modification to the linear activation function is the Rectified linear function, which guarantees non-negative output. Activation functions such as the sigmoid capture non-linear relationships between input and output, and are also differentiable. In modern practice, step functions are replaced by sigmoid activations, because they are non-linear and differentiable. Additionally, the outputs of sigmoid activation functions can be viewed as probability distributions because (i) The sigmoid function f is non-decreasing and right-continuous, (ii) $\lim_{x\to-\infty} f(x) = 0$, and (iii) $\lim_{x\to\infty} f(x) = 1$

In Figure 2.1, the value 1 represents a bias term, which is an additional constant attached to neurons and added to the weighted input before an activation function is applied. These bias constants help the neuron represent patterns that do not necessarily pass through the origin. Non-linearity of activation functions and bias constants enable neurons to learn complex functions.

The process of learning for a neuron involves iteratively changing it's weights on it's incoming synapses to accommodate both previously observed training instances as well as the current training instance. Each neuron's synapse weight initially starts with a random value (usually drawn from a standard Gaussian distribution), and is iteratively updated with every new training instance. However, this update operation only occurs when there is an error

between the predicted output and the target output for the current training sample.

A single neuron is limited by the number of functions that it can fit. For instance, a neuron with a binary output cannot fit a simple XOR function. To fit more complex functions, layers of interconnected neurons are constructed. These layers of interconnected neurons are known as artificial neural networks.

2.3 Artificial Neural Networks

Artificial neural networks are machine learning algorithms that use the artificial neuron as a building block. In these networks, neurons are stacked together within layers, and each of these layers are interconnected in the direction from input towards the outputs. As shown in Figure 2.2 the three types of layers in a neural network are the input layer, hidden layer and output layer. Each layer transforms the weighted output of the previous layer using an activation function to produce a new output. This type of architecture of a neural network is called a feed-forward neural network (Bebis and Georgiopoulos [3]).

Let us now look at the propagation of inputs within these networks. Consider the i^{th} neuron in layer l of the network to be represented by N_i^l, and the synapse weight connecting neuron N_j^l in layer l to neuron N_i^{l+1} in layer $l+1$ represented by W_{ij}^l. Succinctly, we can represent all the synapse weights connecting neurons in layer l to neurons in layer $l+1$ by the weight matrix W^l. Similarly, the output of a neuron N_i^l is represented by y_i^l, and all outputs from neurons in layer l are succinctly represented by y^l.

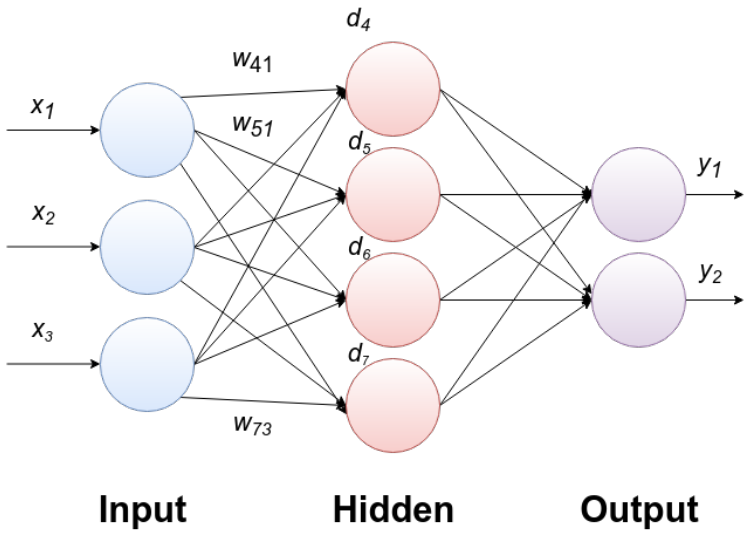

Figure 2.2: Feedforward artificial neural network with an input layer, a hidden layer and an output layer. Although every neuron in one layer is connected to every neuron in the next layer, this need not always be the case, as we will see in convolutional neural networks.

As shown in Figure 2.2, the output of the input layer is given by $y^1 = x = (x_1, x_2, x_3)$, the output of layer 2, which is a hidden layer is given by $y^2 = A_1(W^1 y^1)$, and, in general, $y^{l+1} = A_l(W^l y^l)$. Here A_i is an activation function such as the ReLU or Sigmoid applied at layer i.

Given an input, forward propagation is used to compute a prediction. This prediction becomes useful to the end-user only when the network is trained to accurately make predictions on novel, unseen inputs. Training neural networks requires an algorithm that intelligently adapts the synapse weights between neurons in every layer. This adaptation is performed through an efficient algorithm called back-propagation (Cun [8]).

The goals of back-propagation are straightforward: adjust every synapse weight in the network in proportion to how much it contributes to the overall error when predicting one or more inputs. This adjustment of weights is done using the delta-rule, where, any synapse

weight is updated in a direction that minimizes the total error E of the network. Eventually the back-propagation algorithm produces a set of synapse weights that capture input-output relationships and the network makes accurate predictions during the next forward pass.

$$\text{Delta-Rule: } W_{ij} = W_{ij} - \alpha \frac{\partial E}{\partial W_{ij}}$$

Here, α is a free parameter called the learning rate. The delta-rule requires the partial derivative of the network error with respect to each synapse weight to be useful. This partial derivative is computed using the famous chain rule.

Chain Rule: Given a forward propagation function $f(x) = A(B(C(x)))$, where A, B and C are activation functions at consecutive layers in the network, the derivative of f w.r.t x is given by

$$f'(x) = f'(A).A'(B).B'(C).C'(x)$$

With help from the chain rule, the delta-rule is able to identify (i) exactly how much each synapse weight contributes to the overall error in the network, and (ii) the direction to update each weight such that the total error in the network is reduced.

The back-propagation algorithm is also responsible to propagate output error contributions of each network branch to the inner layers. If these errors are not propagated, it leads to the vanishing gradient problem, where, internal synapse weights are never updated because they never received their error signals. Although many solutions have been proposed to alleviate the vanishing gradient problem, pre-training, LeakyReLU activations and layer-wise training seem to help in practice.

2.4 Word Embeddings

Alignment-free, compositional algorithms such as kraken2 and Opal assign taxonomic classes to DNA fragments by breaking up each DNA fragments into long k-mers. Although they have the potential to provide faster solutions compared to alginment-based methods, they may still not be able to accurately classify novel DNA fragments. Additionally, k-mers in their natural form face a drawback when used in machine learning models: the dimensionality of one-hot vector representation of k-mers grow exponentially in the length of k.

To combat the curse of dimensionality, while also trying to retain semantic meaning of each k-mer, we employ vector space models. One such vector space model is word embeddings. Word embedding software typically make use of the word2vec algorithm (Mikolov et al. [19]) to learn k-mer embeddings. In word2vec, a shallow neural network is trained to predict whether two k-mers occur within a certain distance of each other. As a result, the neural network learns to map closely related k-mers to similar regions in a d dimensional embedding space.

Formally, given the alphabet of nucleotides $\lambda = \{A, T, G, C\}$, a DNA fragment of length L is a sequence $x = x_1, x_2...x_L \in \mathrm{A}^L$. For any $1 \leq a \leq b \leq L$ we represent by $x_{[a,b]} = x_a x_{a+1}...x_b$ the sub string of the DNA fragment from position a to position b. A k-mer is then defined to be any sub string where $b - a = k$, and there are exactly $L - k + 1$ k-mers within every DNA sequence of length L. Given $d \in \mathbb{N}^*$, an embedding of a k-mer to $\mathbb{R}^d$ is a mapping $\phi : \lambda^k \to \mathbb{R}^d$ to represent each k-mer by a fixed length vector $\phi(x) \in \mathbb{R}^d$.

Let N represent the total number of (k - 1 nucleotide) overlapping k-mers from a DNA fragment of length L.

$$N = L - k + 1$$

Given that $d \in \mathbb{N}^*$, we can represent all N k-mers as a set of real-valued, fixed dimensionality vectors. When these vectors are arranged in increasing order of the k-mer indices, they form a $d \times N$ sentence matrix which retains semantic and structural integrity.

Let us now take a look at a simple example sentence matrix computed from a DNA fragment.

Assume for this example that k is 3, the embedding dimension d is 100 and DNA fragment is CTAGGAT. The number of unique k-mers will be given by

$$len(dna\ sequence) - k + 1 = 7 - 3 + 1 = 5$$

The unique k-mers in this example are {CTA, TAG, AGG, GGA, GAT}, and the sentence matrix has dimension 100×5 where the i^{th} row represents the i^{th} dimension of each of the five k-mers from the DNA fragment.

In Plinko, we make use of the DNA2vec software (Ng [21]) which internally calls the gensim word2vec algorithm (Řehůřek and Sojka [23]). Using DNA2vec and the NCBI non-redundant database (Coordinators [6]), we compute the distributed vector representation of every k-mer. The DNA2vec software has the added advantage that it allows computation of the

vector representation of a range of values of k in the same training run.

Table 2.1: Results from training the word2vec algorithm on the NR data set using DNA2vec

Property	***Value***
Number of k-mer examples	174,194,966,114
Number of base-pairs	1,045,166,912,015
Effective k-mers processed / sec	187,666
k-mer range	[3 - 8]
Dimension d of each k-mer	100
Number of unique k-mers	349,504
Wall time	400 hours (16.6 days)
CPU time	2,182 hours (90.9 days)

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are an important type of Artificial Neural Network that learn to extract spatial features from their inputs. Architecturally, CNNs are deep directed graphs consisting of multiple processing layers that each learn the spatial structure of their inputs by composing multiple linear and non-linear transformations. This layering technique enables CNNs to learn increasingly higher level features that lead up to the prediction of associated labels. CNNs have become successful in disciplines such as computer vision and natural language processing, where inherent spatial structure can be extracted.

CNNs derive their name from the type of hidden layers they are made from. Instead of using a simple stack of fully-connected neurons, the hidden layers in CNNs typically use a mix

of convolutional layers, pooling layers, fully-connected layers, and normalization layers. An important distinction between regular ANNs and CNNs is that in regular ANNs, activation functions are directly applied to every hidden neuron's output, however in CNNs, convolution and pooling functions are first applied before activation functions.

Conceptually, a convolution layer accepts an input, applies a convolution kernel over it by multiplying the input signal with kernel(s) to obtain a new modified signal. Mathematically, a convolution between two functions f and g is defined by:

$$(f * g)(i) = \sum_{j=1}^{m} g(j).f(i - j + \frac{m}{2})$$

This is equivalent to the dot product between the two functions f and g.

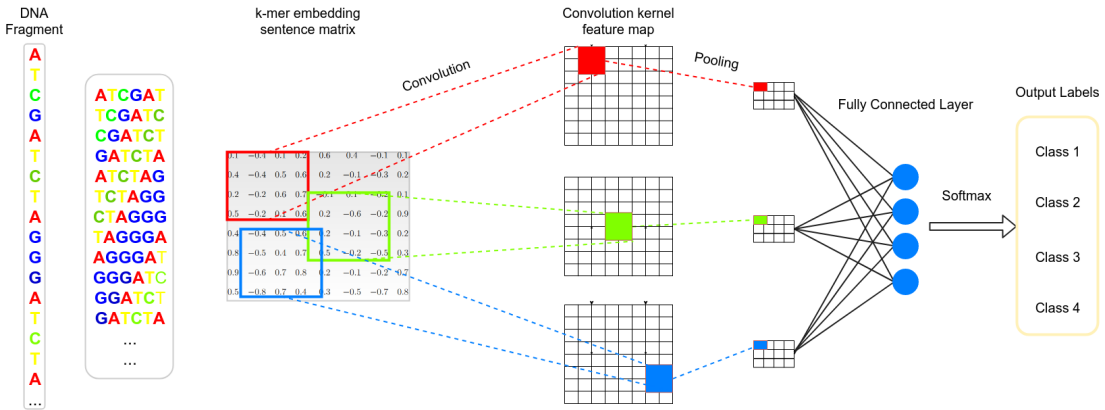

Figure 2.3: A simple CNN architecture that takes a sentence matrix as input, applies convolution and pooling functions in the hidden layers and makes a final label prediction. The sentence matrix is computed by the column-wise concatenation of k-mer word embedding vectors

The convolution and pooling layers in CNNs enforce three important properties to exploit spatially local structures within sentence matrices:

1. **Sparse Weights** - The kernel size is small compared to that of the sentence matrix. This helps them detect specific patterns co-occurring in a subset of k-mer dimensions.

2. **Parameter Sharing** - The features learned by a convolutional kernel are shared by more than one activation function in the model. Sparse weights along with parameter sharing help reduce the memory footprint of CNN models.

3. **Pooling** - Pooling ensures that neural network models are approximately invariant to small translations (synonymous mutations) of their input.

Chapter 3

Input Composition Techniques

3.1 Introduction

Traditionally in bioinformatics, composition based machine learning approaches have been trained on either (i) the raw DNA sequence, (ii) their one-hot form or (iii) k-mer profiles (frequency of k-mer occurrence). Recently, Opal made use of k-mer profiles of the inputs and trained a state-of-the-art SVM to perform taxonomic binning. These representations however, impose large restrictions on the model′s ability to extract meaningful low-level feature correlations between k-mers.

Although DNA is primarily viewed as a text sequence, DNA is a molecule, which may imply presence of spatial structure and shape that may be biologically meaningful. A first step towards designing good prediction systems is thus to find good representations for these input DNA sequences.

We hypothesize that deep neural networks trained with inputs that are designed to account for the double strandedness of DNA as well as long-term interactions may improve performance in taxonomy identification. In this chapter, we investigate four novel approaches to account for spatial properties of DNA sequences. They are: (i) k-mer embeddings and their

use in representing 1D DNA fragments as 2D sentence matrices. (ii) Multi-view Convolutional Neural Networks (MVCNN) that integrate properties of the double helix as well as the associated amino acid reading frame sequence(s). (iii) Hilbert space filling curves that map a 1D DNA fragment to 2D co-ordinates while maintaining spatial locality. (iv) The combination of six frame translations of nucleotides into amino acid sequences.

3.2 Plinko Convolutional Neural Network

In all of our experiments, unless specified otherwise, we use the same CNN architecture, while only varying the input layer.

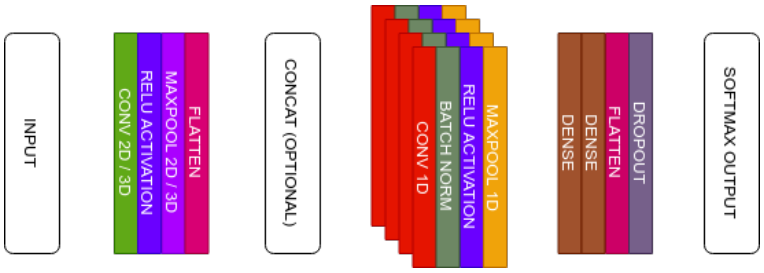

Figure 3.1: The Plinko CNN Architecture.

The first layer in a Plinko CNN is the Input layer. In our experiments, inputs may take the shape of either 2-dimensional or 3-dimensional tensors. The last axis of input tensors is the k-mer embedding vector of dimensionality $d = 100$. There may also be multiple inputs of varying dimensionality that all feed into the first computational block in the network. Networks with multiple inputs are hereby referred to as multi-view CNNs and are discussed in detail later in this chapter.

In the Plinko CNN architecture, we have a need to minimize the network complexity as

much as possible for three reasons. (i) Since we train and deploy one network at each node in the taxonomy, we will end up training on the order of 10^3 different networks. In order to minimize training time and model size, we need to keep the model as shallow as possible. (ii) One of the key metrics in taxonomy prediction is inference time. To keep inference time as low as possible, we need to minimize the complexity of the network. (iii) Specialized networks that classify fewer labels may be better suited to find clear hyper planes between output labels.

To achieve a lean architecture, we train five computational blocks in every network. The computational block immediately following the Input layer is either a Convolution 2D or Convolution 3D block, depending on the dimensionality of the input. This block deploys 256 parallel 2D (or 3D) convolution kernels over the input, passes the outputs through the ReLU activation function, and obtains the maximum activation from each kernel.

The top 256 activations are merged together in the concatenation layer, and passed through a series of four Convolution 1D computation blocks. Within each block, between 64 and 128 kernels operations are applied, vectors normalized and pooled with a stride of 2. The series of four Convolution 1D blocks aim to extract long-term spatial interactions between features that were previously extracted from the first Convolution 2D (or 3D) block.

A pair of fully-connected hidden layers follow the last Convolution 1D block, and are used to extract critical high-level patterns. The neurons in the last few layers are crucial in making the final link between extracted patterns and the output label. A dropout regularizer with probability 0.3 is then applied after the second dense layer to make the network robust against simple (synonymous) mutations in the input.

Finally, a softmax layer is applied to the outputs of the fully-connected layers to obtain y normalized probability scores, where y represents the number of possible classes the input can belong to.

3.3 Methods

To test Plinko's performance on different experiments, a data set of real DNA fragments were generated to classify the superkingdom rank of each fragment. The four superkingdom ranks were Bacteria, Archaea, Eukaryota, and Virus.

3.3.1 Data Generation

A data set containing length 100 coding DNA fragments were generated with the program Radogest using the Quality Sorting Leaf selection strategy with no more than 10 genomes per species. The Quality Sorting Leaf (QSL) strategy works by first sorting genomes by quality at each species node in the taxonomy tree. Genome quality is determined by whether it is a reference or representative genome and by the length of contigs. The strategy then selects up to a fixed number of these genomes at the species level. These genomes are propagated up the taxonomic tree so that each taxonomy includes every genome that was selected at the species level for a species included in that taxonomy. Coding DNA is DNA that codes for a gene. These coding DNA fragments were generated by randomly sampling from genomes downloaded from the National Center for Biotechnology Information (https://www.ncbi.nlm.nih.gov/genome). A training data set of 10^7, 100 length DNA fragments

Table 3.1: The number of coding domain genomes used by superkingdom

Superkingdom	Amount
Archaea	406
Bacteria	18721
Eukaryota	1829
Virus	1822

were drawn where approximately 2.5×10^6 fragments were sampled from each superkingdom. A test set and a validation set each consisting of 1.8×10^6 length 100 DNA fragments were sampled in the same manner as the training set. Table 3.1 refers to the number of genomes by superkingdom that were used to generate the data.

The training data set was used to train Plinko, and the validation data set was used to determine how many epochs Plinko was trained. The epoch with the highest validation accuracy was used to select the Plinko model. The test data set was used to report accuracy.

3.3.2 2D Sentence Matrices Using k-mer Embeddings

To generate a k-mer embedding to represent DNA and Amino Acid strings we made use of the DNA2vec software and the NR Nucleotide data set from RefSeq to compute the distributed vector representation for all k-mers, where k ranges from 3 to 8. We stop at 8 since the number of k-mers grows at the rate of 4^k. Additionally, since we fine tune the embedding vectors in every taxonomy prediction neural network, the time required to (re)train k-mer vectors beyond $k = 8$ becomes infeasible. Recently, (Menegaux and Vert [17]) showed that increasing dimensionality beyond 100 does not improve average species-level precision and recall. Thus, in our experiments, we fix the dimension d of all embedding vectors to be 100.

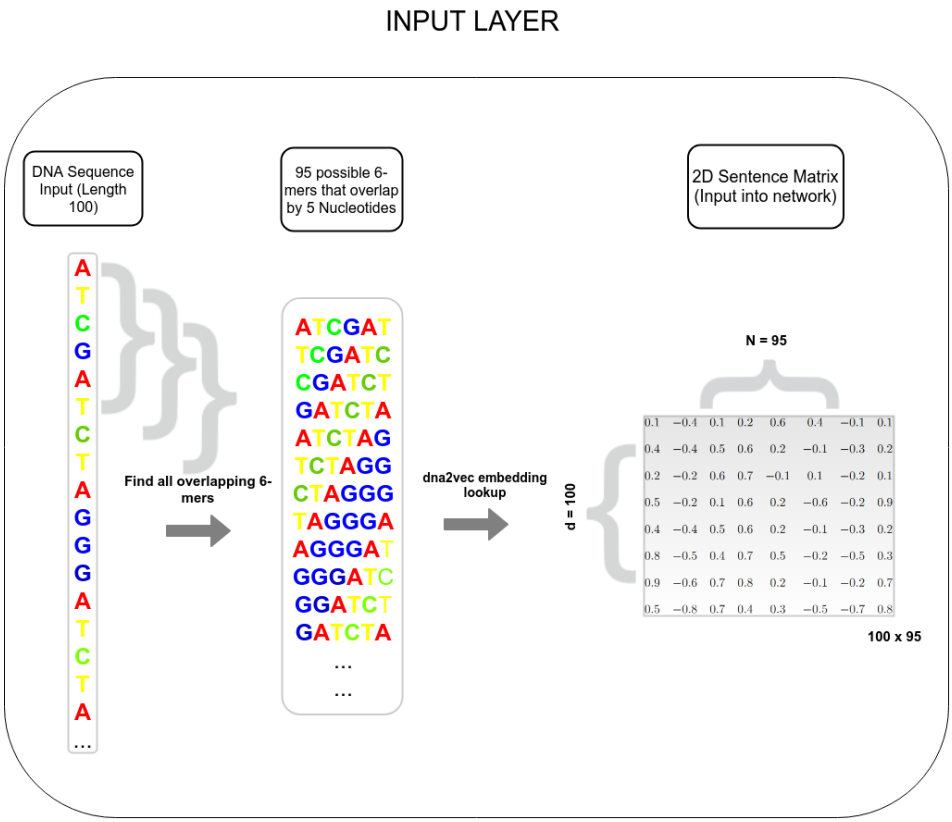

Figure 3.2: Example of how a DNA sequence of length 100 is converted to a 100×95 sentence matrix, where each row represents the embedding vector of a 6-mer from the original DNA sequence.

In Figure 3.2 we see three stages of input processing that ends in the creation of the 2-dimensional sentence matrix. First, the nucleotide sequence of length L is passed into a function that breaks down the sequence into a set of $L - k + 1$ different k-mers. These k-mers are used as indexes into an embedding lookup table built with DNA2vec. The embedding vectors for each k-mer are stacked together to create an $100 \times (L - k + 1)$ sentence matrix. This embedding layer feeds into the Convolution 2D block in the Plinko CNN.

3.3.3 Reading Frames: Moving from Nucleotides to Proteins

In molecular biology, reading frames are a way of partitioning the sequence of nucleotides in a nucleic acid molecule into a set of consecutive, non-overlapping triplets. When each triplet equates to an amino acid or stop signal during translation, they are called codons (https://en.wikipedia.org/wiki/Reading_frame). A single strand of a nucleic acid molecule has a phosphoryl end, called the 5 -end, and a hydroxyl or 3 -end. These define the 5' to 3' direction. There are three reading frames that can be read in this 5' to 3' direction, each beginning from a different nucleotide in a triplet. In a double stranded nucleic acid, an additional three reading frames may be read from the other, complementary strand in the 3' to 5' direction along this strand.

In general, at the most, one reading frame in a given section of a nucleic acid (say a 100-mer), is biologically relevant. This reading frame is called the open reading frame (ORF). An ORF is a reading frame that has the potential to be transcribed into RNA and translated into protein. In the coding data set, all 100-mers are segments of genes that code for proteins. Each triplet corresponds to an amino acid dictated by a translation table. For our experiments, we make use of translation table (11) that is typically used for Bacterial, Archaeal and Plant Plastid Code.

3.3.4 Multi-View Convolutional Neural Networks

The term multi-view originates from a concept in computer vision that involves the representation of 3D shapes in an image recognition setting. The authors in (Su et al. [26]) question whether 3D shapes should be represented with descriptors operating on their native

3D formats, such as voxel grid or polygon mesh, or if they can be effectively represented with multiple 2D view-based descriptors. Surprisingly, the authors found that when using standard CNNs trained to recognize the shapes' with views rendered independently of each other, a 3D shape can be recognized even from a single 2D view at an accuracy far greater than using 3D descriptors.

In the context of taxonomy identification, this is useful because we would like to make use of multiple derived views from the original DNA fragment to make a prediction. We have access to five views of a nucleotide sequence. (i) The one-hot encoding of the nucleotide sequence. (ii) The one-hot encoding of the nucleotide reverse complement sequence. (iii) The k-mer embedding sentence matrix of the nucleotide sequence. (iv) The k-mer embedding sentence matrix of the nucleotide reverse complement sequence. (v) The six amino acid translations that any DNA fragment can get translated into. The concatenation of all six translations forms a sequence that can be represented using a k-mer embedding sentence matrix where the embedding are indexed on amino acid k-mers. This is explained in detail in the subsection on amino acid concatenation.

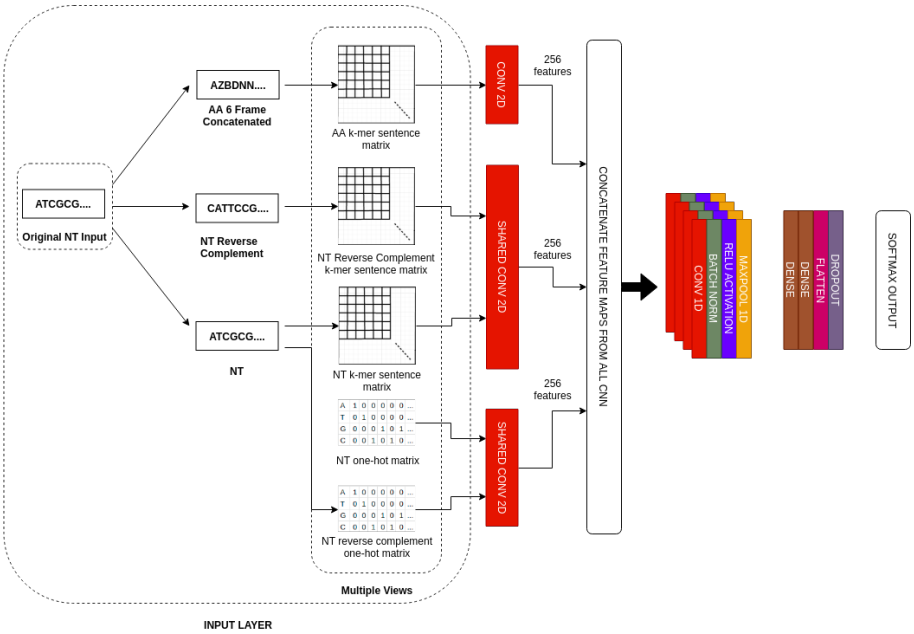

Figure 3.3: Multi-view Input to Plinko CNN

In Figure 3.3 above, the original nucleotide (NT) input sequence is transformed into three sequences. First, all six reading frames of the original NT sequence are concatenated together using translation table 11 to form an Amino Acid (AA) sequence that is 6 times the length of the longest Amino Acid translation. Second, the NT sequence is reversed, all nucleotides are complemented (A becomes T, G becomes C and vice versa). Third, the original sequence is passed unchanged.

Similar to the k-mer nucleotide embeddings, another amino acid embeddings look-up table is generated using DNA2vec for all possible amino acid 3-mers. In our experiments, we use only 3-mers of dimensionality 100.

In the second stage, each of three transformed sequences are converted into a representation that is easily understood by the Plinko CNN. The 6-frame concatenated amino acid

sequence is converted into a sentence matrix using all 1-NT overlapping 3-mer embeddings of dimensionality 100. The nucleotide and nucleotide reverse complement sequences are converted into their one-hot forms. The same nucleotide and nucleotide reverse complement sequence are converted to sentence matrices using all 1-NT overlapping 6-mer embeddings of dimensionality 100. We use 6-mer embeddings for nucleotides as they showed best trade off characteristics between performance and the number of model parameters.

Note on Implementation: In order to maximize usage of all inputs as features in the model, we constructed intermediate shared 2D convolution layers in between the input layer and the Plinko CNN. With Keras, it is easy to create shared layers with the functional API. In Figure 3.3, the Shared 2D CNN layer is instantiated using the functional API and is called with multiple inputs. The only constraint on calling an instantiated layer is that the input tensor must be of the expected size. Note that by calling an instantiated set of layer(s) we aren't just reusing the architecture, we are also reusing the weights.

The training time, inference time and size of each Plinko CNN model are dictated by the number of parameters in the model. In order to reduce the number of parameters while also leveraging the different views of the nucleotide sequence input, we make use of shared 2D CNN layers. In Figure 3.3, there are two shared CNN layers. One used by the nucleotide sentence matrices (NT and NTREV) and another used by the one hot matrices (OH and OHREV).

Each of the 2D CNN layers output a vector with 256 real values. The three vectors are concatenated to form a single 768 dimension vector that feeds into the Plinko CNN.

Table 3.2: Plinko with and without Shared Convolutional layers following multi-view inputs

Architecture Type	*Parameters*	*Size*	*Training Time (10 Epochs)*	*Prediction Time*
Plinko with Shared 2D CNN Layers	2,607,688	30 MB	29 Hours, 35 Mins	1,666 seq/sec
Plinko without Shared 2D CNN Layers	3,273,644	38 MB	37 Hours, 21 Mins	1,345 seq/sec

3.3.5 Hilbert Curves

Recently in bioinformatics, Hilbert space filling curves have become popular input representation techniques. The first bioinformatics paper making use of this technique with CNNs was in 2018, where the authors predicted key determinants of chromatin structure.

Hilbert curves have been effective in prediction tasks where distal relationships within DNA sequences are necessary to make accurate predictions. The authors in (Anjum et al. [2]) showed that for Enhancer region prediction, CNNs with Hilbert curve transformed DNA sequences as inputs had the highest accuracy compared to similar CNNs and SVMs with regular DNA sequence as input.

The main idea behind mapping a 1-dimensional DNA sequence to a 2-dimensional grid is to intelligently place each nucleotide of the sequence to a cell in the 2D grid so that proximal elements of the sequence will stay in close proximity to one another, while the distance between distal elements is reduced.

Hilbert curves have two main properties (i) **Neighborhood Preserving** and (ii) **Clustered Yields**. The neighborhood preserving property of Hilbert curves ensures that two nucleotides which are close together within the 1-dimensional DNA sequence are also close within the 2-dimensional grid. The clustered yields property ensures that when a rectangular subsection of the 2-dimensional grid is cut out, the subsection is guaranteed to contain

maximally connected 1-dimensional sub sequences.

The combination of (i) and (ii) motivate the use of convolutional kernels on Hilbert curves derived from DNA sequence: rectangular tensors encoding the convolution operations contain a minimum amount disconnected pieces of DNA sequences to train on. In Section 3.4.4 we test the ability of Hilbert curves to discriminate between classes of the Superkingdom Coding Sequences data set.

By construction, Hilbert curves yield a square grid of size $2^n \times 2^n$, where n is the order of the curve. Hilbert curves of orders 1 and 2 are shown in Figure 3.4.

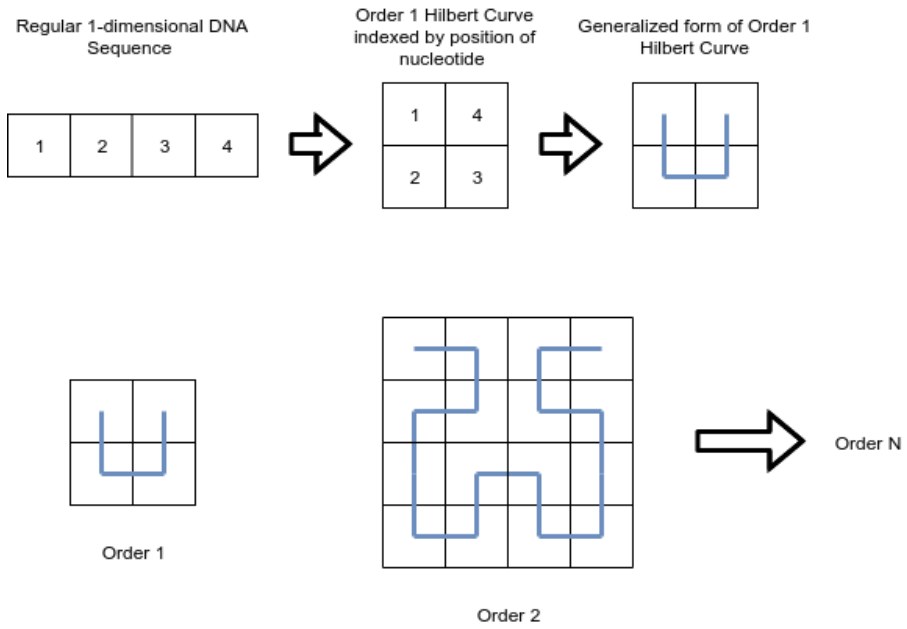

Figure 3.4: Process in transforming a simple 4 NT DNA sequence into an order 1 Hilbert curve. Figure also shows how higher order Hilbert curves are constructed.

3.4 Results

3.4.1 2D Sentence Matrices Using k-mer Embeddings

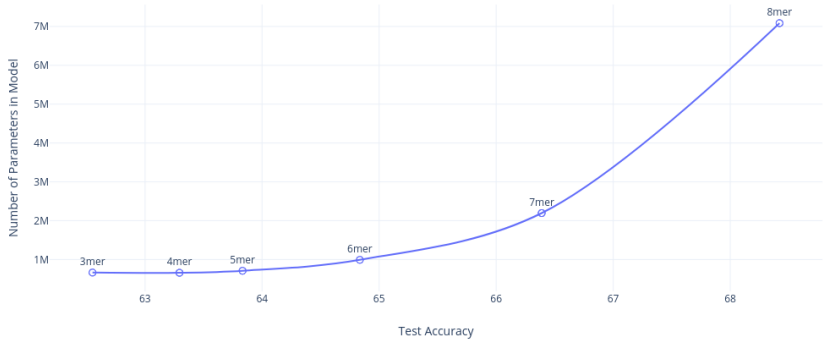

Figure 3.5: Plot showing test accuracy of Plinko CNN trained with k-mer embedding inputs where k ranges from 3 to 8.

Figure 3.6: Plot showing training and test accuracy curves for Plinko CNN trained with k-mer embedding inputs where k ranges from 3 to 8.

When we ran experiments measuring the performance of Plinko networks with 2D sentence matrix inputs set to trainable versus those set to non-trainable, we consistently noted degraded performance across 5 taxonomy prediction tasks for the non-trainable case. This indicates that the embedded representation of k-mers vary with the taxonomy prediction

labels and that it is important to fine-tune the vector representations of k-mers for each taxonomy prediction task. The DNA2vec k-mer embeddings thus act as an unsupervised pre-training step for all taxonomy classification CNNs used downstream.

As seen in the Figure 3.5 and 3.6, the embedding layer accounts for a large portion of the number of parameters in the Plinko model. The number of parameters in a model serves as a good proxy for memory requirements, training time and prediction time of the neural networks. We see that any value of k beyond 5 or 6 leads to extremely large number of model parameters which adversely affects training time, model size and inference time. Additionally, when looking at the training and test accuracy curves, we note that the model starts to overfit beyond $k = 6$.

In the experiments that follow, we use a value of $k = 6$ with dimension of each 6-mer as $d = 100$. We also ensure that the embedding vectors in the CNNs are trainable.

3.4.2 Reading Frames: Moving from Nucleotides to Proteins

To understand the benefits of using only the open reading frame as opposed to using all reading frames in the task of taxonomy identification, we trained two Plinko architectures that had the amino acid input view. The first network Plinko (AA) is a single view network, while the second network Plinko (NT, NT REV, AA) is a multi-view network.

In both architectures, we experimented with three types of amino acid inputs. (i) Correct Reading Frame only (ORF), (ii) Concatenation of the 5 incorrect reading frames and (iii) Concatenation of all 6 reading frames. The results are shown in Figure 3.7 below.

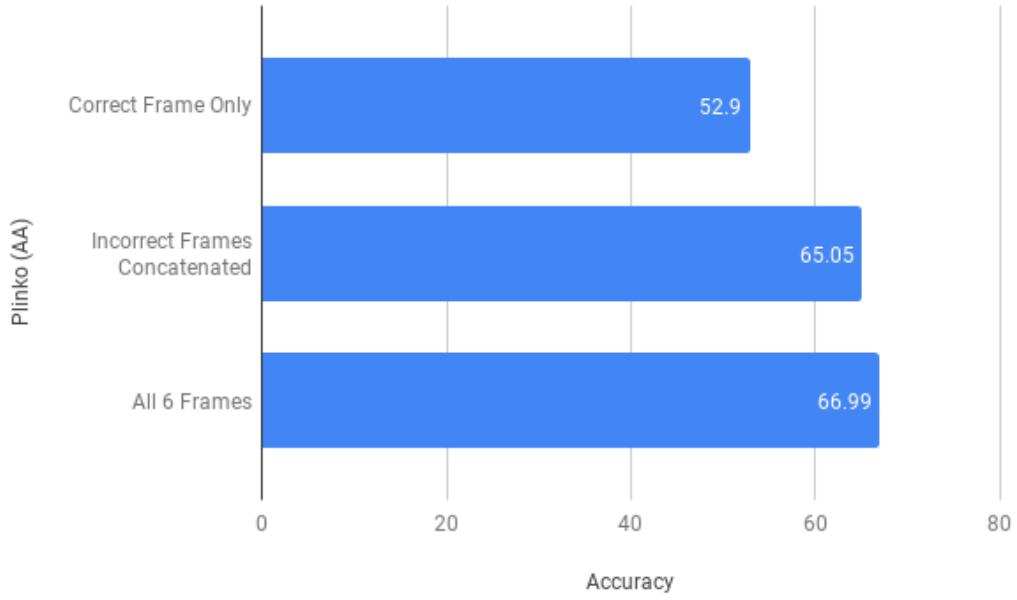

Figure 3.7: Plinko (AA) architecture trained with three types of AA sequences (i) Correct Reading Frame (ORF), (ii) Concatenation of 5 Incorrect Reading Frames (iii) Concatenation of all 6 Reading Frames

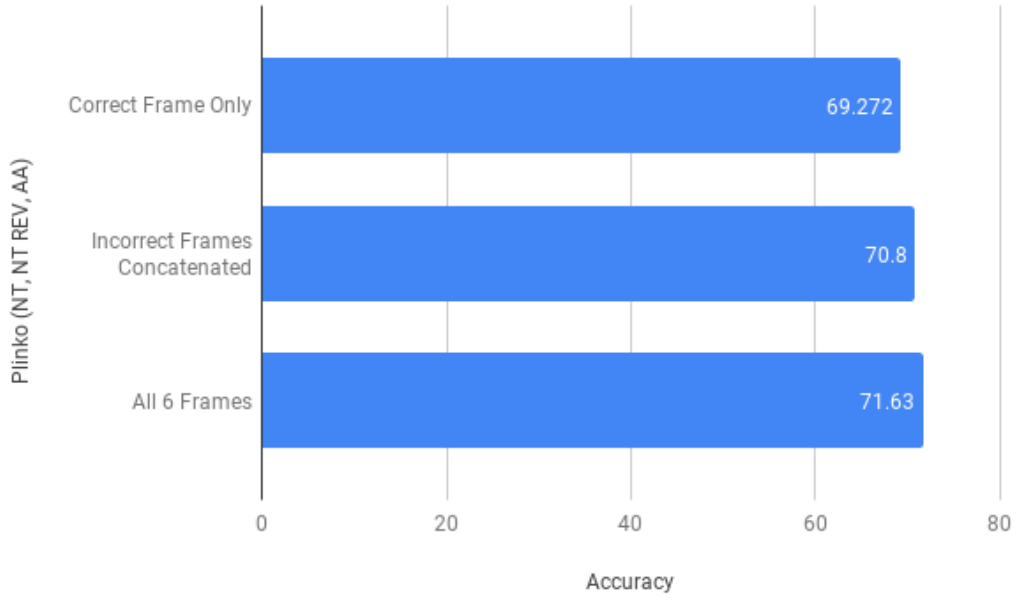

Figure 3.8: Plinko (NT, NT REV, AA) architecture trained with three types of AA sequences (i) Correct Reading Frame (ORF), (ii) Concatenation of 5 Incorrect Reading Frames (iii) Concatenation of all 6 Reading Frames

As seen in Figures 3.7 and 3.8 above, in the case where no other view of the input query sequence is given to the Plinko CNN, the additional information coming from the 5 incorrect reading frames makes a big difference in the classification accuracy. However, when Plinko is supplemented with more views of the input query sequence (NT and NTREV), the accuracy of the network remains roughly the same.

This helps us come to a conclusion that when constructing Plinko networks with an Amino Acid view, it is beneficial to use all reading frames for taxonomy identification. Thus, all of our networks used in the experiments make use of the concatenation of all 6 reading frames as AA input.

3.4.3 Multi-View Convolutional Neural Networks

As we see in Figure 3.9, rather than using a single view Plinko CNN like Plinko (NT) or Plinko (AA), the use of multiple views to the CNN helps in the task of differentiating between the four superkingdoms.

Figure 3.9: Plot showing the test accuracy of each of the 6 multi-view Plinko CNN models. For the data set, we see that NT, NT Reverse Complement and AA 6 frame concatenated provides the best results.

Other subtle, but important findings were that adding one-hot views to the input tends to inhibit test performance, while adding amino acid views to the input tends to consistently improve test performance on protein coding data.

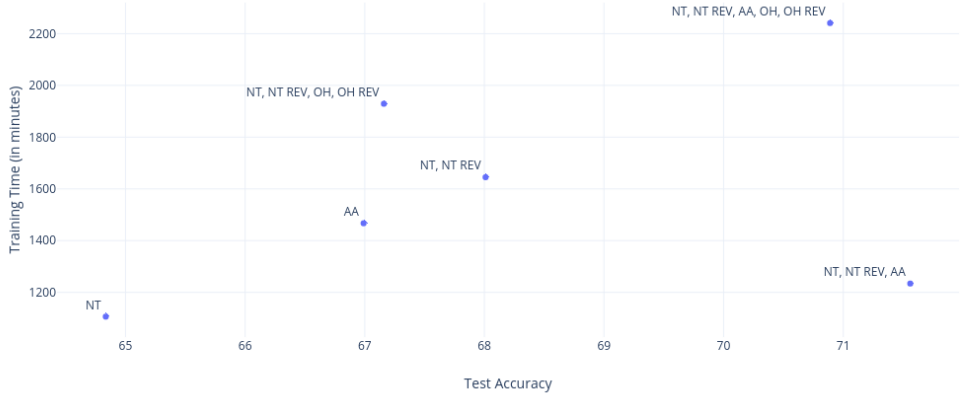

Figure 3.10: Comparing Training time and test accuracy shows the advantage of using Plinko (NT, NT REV, AA) as input over other input types.

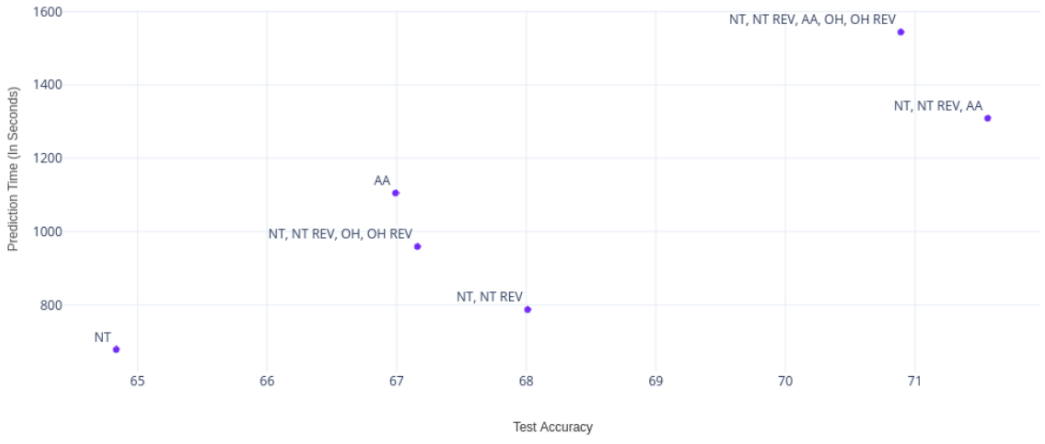

Figure 3.11: Comparing Prediction time and test accuracy shows the advantage of using Plinko (NT, NT REV, AA) as input over other input types.

As we can see from the Figure 3.10 and 3.11, Plinko (NT, NT REV, AA) is the best model in terms of test accuracy and training time. The model also has reasonable prediction time. It must be noted that although Plinko (NT, NT REV, AA) has the best trade off characteristics for the protein coding data set, we have noticed that for other data sets, Plinko (NT, NT REV, AA, OH, OH REV) performs just as well. To avoid the unforeseeable situations where one hot representations can add significant value to model performance, we use the Plinko (NT, NT REV, AA, OH, OH REV) architecture in the rest of this report.

3.4.4 Hilbert Curves

In our experiments, instead of replacing each index in the Hilbert grid by a nucleotide from the 1-dimensional DNA sequence, we replace each index in the grid by the relative position of every 6-mer from the 1-dimensional sequence. This way, we can represent the 1-dimensional sequence of 6-mers on the Hilbert grid.

To take additional advantage of the embedded vector representation of 6-mers, every indexed 6-mer in the Hilbert grid is replaced by their corresponding embedding vector. Thus, through a Hilbert curve mapping modified for k-mers, the 1-dimensional DNA sequence is transformed into a 3-dimensional tensor where the first two dimensions represent the Hilbert grid coordinates of the 6-mers and the last dimension represents the embedding vectors.

The results from the experiments using Hilbert curve transformed inputs on different Plinko architectures are presented in Figure 3.12.

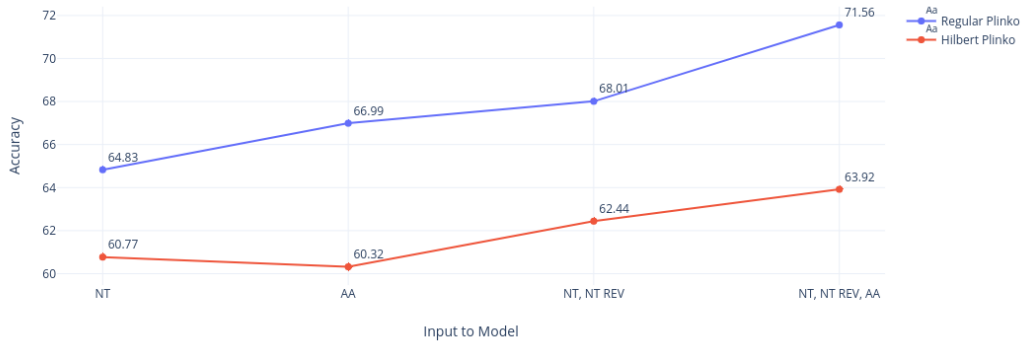

Figure 3.12: Comparing the test accuracy of Plinko architectures that use 2D sentence matrix representations as inputs to architectures that use 3D Hilbert tensor representations as inputs. Sequence Length: 100 NT.

For the 100 NT coding data set, we see that the performance of all Plinko architectures that use 3D Hilbert tensors as inputs perform very poorly compared to when 2D sentence matrices are used as inputs.

Given that the sequences in the coding data set are of length 100 (95, 6-mers), and that Hilbert mappings only create squares of size $2^n \times 2^n$, 4 was the smallest value of n that could cover all 6-mers from the input. One of the reason that Plinko with Hilbert curve inputs performed poorly could have been due to the fact that only 95 of the possible 256 cells in the Hilbert grid were filled by 6-mers from the nucleotide sequence.

In order to identify whether Hilbert cell coverage was the cause for poor performance, we created two more data sets where the length of the DNA sequences were 200 NT (195, 6-mers) and 261 NT (256, 6-mers) respectively. The results of Hilbert analysis on the two data sets are shown in Figure 3.13.

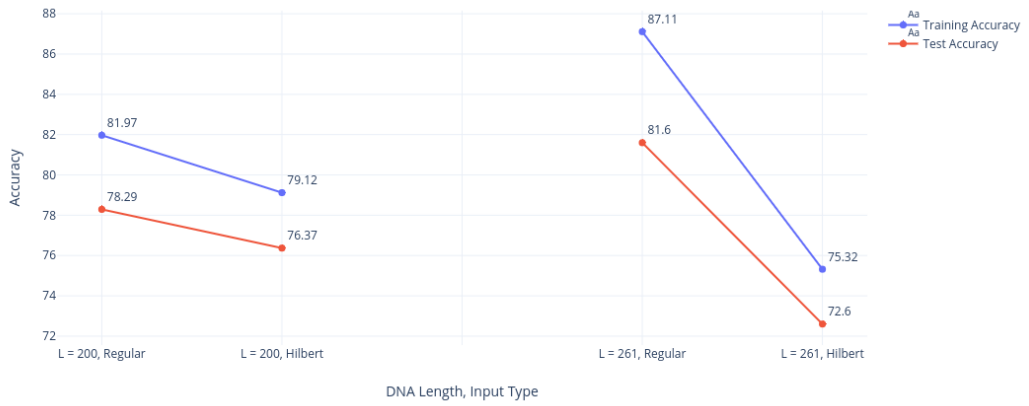

Figure 3.13: Comparing accuracy of 2D sentence Matrix input representation and 3D Hilbert curve input representation on Plinko (NT, NT REV).

As shown in Figure 3.13, a similar trend of low test accuracy is observed for Plinko networks that use 3D Hilbert inputs, regardless of sequence length (200 NT and 261 NT). Thus, we can conclusively say that Hilbert cell coverage completeness does not improve performance of Plinko regardless of the architecture.

3.5 Conclusion

In this chapter we investigated four approaches to represent DNA sequences for the downstream task of taxonomy identification using Convolutional Neural Networks. They are summarized below.

(i) We investigated how k-mer embeddings could be used to represent 1D DNA sequences as 2D sentence matrices, and determined with experimental evidence that a size of 6 was the

highest value for k that had good memory and timing properties for nucleotide sentence matrices. Additionally we observed that setting the embedding trainable flag on in the Plinko CNNs consistently improved taxonomy prediction accuracy.

(ii) We investigated the transformation of nucleotide sequences into all 6 possible amino acid translations and concluded that the concatenation of all 6 reading frames improved Plinko performance, regardless of how many other views were present in the Plinko input layer.

(iii) We investigated Multi-view Convolutional Neural Networks and how they improve the ability of the model to classify the taxonomic labels. We concluded that using all views (one-hot of nucleotides and their reverse complements, sentence matrices of nucleotides and their reverse complements and sentence matrix of concatenated 6-frame amino acid translations) resulted in the best performing models, regardless of the data set used.

(iv) Lastly we investigated how space filling Hilbert curves negatively impacted Plinko performance, regardless of the architecture used in Plinko. We also showed that fully utilizing the Hilbert cells are not beneficial to Plinko's performance.

Based on our findings in this chapter, henceforth, all Plinko architectures will

1. Make use of Multi-views
2. Use the concatenation of all 6 frame translations as the amino acid view to the model
3. Use 6-mer embeddings for nucleotides and 3-mer embeddings for amino acids, and
4. **Not use** Hilbert mappings for view transformations

Chapter 4

Taxonomy Prediction Experiments

4.1 Introduction

In this chapter, we present the complete Plinko tree architecture: A taxonomy prediction algorithm that by construction is a tree, where each node is a multi-view CNN that classifies DNA fragments. Every MVCNN within Plinko performs three tasks (i) input transformation, where the DNA fragment is replaced by a set of nucleotide and amino acid sentence matrices constructed from vector embedded k-mers, (ii) shared convolutions, where 2-dimensional convolutional filters are deployed to extract spatial structure from the sentence matrices, and (iii) concatenation and reshaping of 2-dimensional feature maps, passing them to a set of sequential 1-dimensional convolutional layers with pooling. Every MVCNN in the Plinko tree uses the same neural network architecture, and so a memory bound is guaranteed for Plinko that is a function of only the number of internal nodes in the taxonomy. An important distinction from other taxonomy prediction models is that Plinko's size is independent of the number of input DNA fragments in the data set.

In the following sections, we shed light on the salient features of Plinko, showcase it's advantages, and point out where it currently falls behind other taxonomy prediction algorithms. In our experiments, we compare Plinko with (i) kraken2, a traditional non-learning based metagenomic classification tool, and (ii) Opal, an SVM based metagenomic classifier that

recently achieved state-of-the-art precision and sensitivity in DNA read classification. Although our implementation of Plinko is still a prototype, early experience with the system is encouraging.

4.2 Plinko n-ary Tree Architecture

The Plinko tree architecture follows the exact structure of the taxonomy tree, where each internal node in the taxonomy tree is a decision point implemented using a Plinko MVCNN. At query time, a DNA fragment is first passed to the root node MVCNN which makes a prediction to go down one of it's child sub-trees. This process repeats at every node, until either (i) a leaf node is reached, (ii) prediction probability falls below a certain threshold, or (iii) a user specified final prediction rank (like class, order etc) is reached.

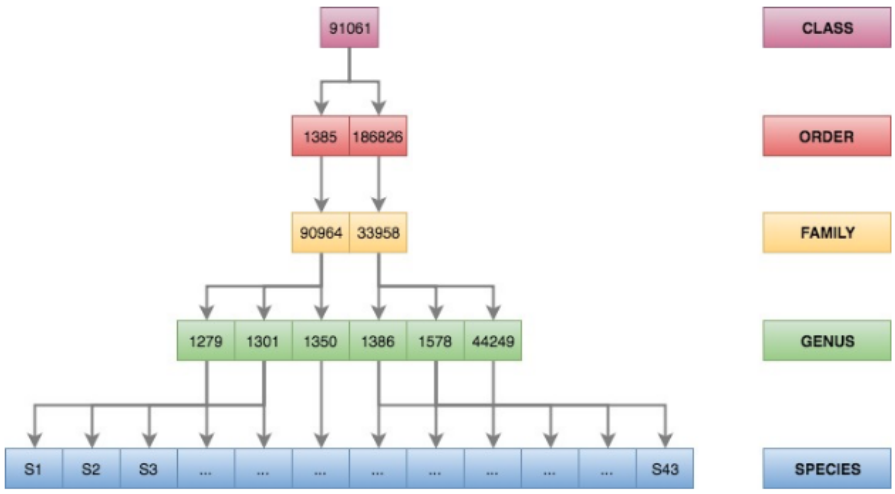

Figure 4.1: Plinko n-ary decision tree architecture where every node is an MVCNN which predicts one of 'n' child Taxonomy IDs. Each node in the tree has a training accuracy associated with it. At inference time, when a new DNA fragment is to be classified, it is first passed to the MVCNN at the root node, after which it is routed to the most probable child MVCNN node

4.3 Plinko Ensemble

To construct a more effective Plinko solution at the superkingdom rank, we used an ensemble of three types of Plinko models trained on DNA fragments from the coding data set.

1. **One Vs One Classifiers**: We developed 6 binary classifiers that were trained to specially differentiate each pair of classes. Example: Bacteria Vs Eukaryotes.

2. **One Vs All Classifiers**: We developed 4 binary classifiers that learnt to distinguish one class from all other classes. Example: Bacteria Vs (Eukaryotes + Virus + Archaea).

3. **n-ary Classifier**: We developed one multi-class classifier that predicts the probability of a DNA fragment belonging to one of the four classes.

Our ensemble solution used gradient boosted decision trees to combine the 11 models into one complete multi-class classifier at the superkingdom rank. We ran several experiments making use of the hardness profiles of the coding data test sequences for Plinko, Plinko ensemble, kraken2 and Opal. Our observations and explanations are provided in the Results section of this chapter.

4.4 Precision, Sensitivity and F-measure for Taxonomy Structures

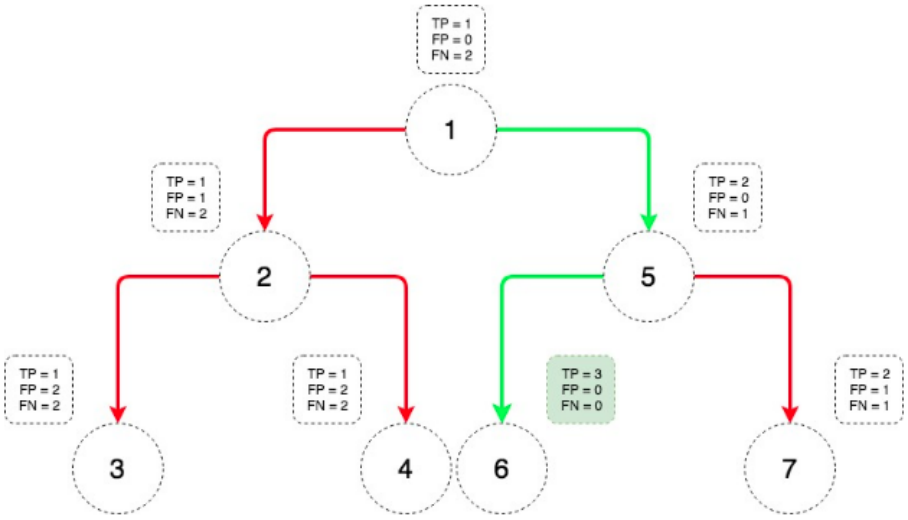

Figure 4.2: Evaluating a prediction path relative to the ground truth path shown in green

To measure the performance of a taxonomy prediction algorithm, we made minor changes to the definitions of true positive, false positive and false negative so as to account for the taxonomy tree structure.

- **True Positive**: At a given level in the tree, a node in the prediction path which aligns with the true label node.
- **False Positive**: At a given level in the tree, a node in the prediction path that does not align with the true label node.
- **False Negative**: At a given level in the tree, a node that exists in the true label path but does not exist in the prediction path or if the node is incorrectly predicted at that level.

4.5 Measuring Hardness

Hardness is a score given to every query DNA sample, and measures the ratio between the degree of similarity of the query sample and the most similar training sample *outside* the query sample's class to the similarity of the query sample and the most similar training sample *within* the query sample's class. In other words, hardness is a way to measure the ability of algorithms to learn patterns beyond sequence similarity.

In our experiments, we created two types of data sets for analysis based on the origin of the query DNA samples:

1. **Type 1** - Where the training samples were complete genomes drawn from a set of species and the query samples from held-out, complete genomes from those same species.

2. **Type 2** - Where the training samples were random length 100 DNA fragments extracted from genomes with replacement, and the query samples were again random length 100 DNA fragments.

The first type is typically encountered in practice, where genomes from novel species that have never been previously identified are queried to the algorithms. This type of data set helps us show the generalizability of each algorithm. We used the Mash Jaccard estimate as a measure of similarity of query DNA samples.

The second type is typical of a machine learning data set, where training samples to the methods are drawn from a data generating distribution, and test samples are drawn from the same generating distribution to see if the algorithms can approximate the data generation

distribution function. We used BLASTn to obtain bit scores to act as a measure of similarity of query DNA samples.

$$Hardness = \frac{Inter - sample\ similarity}{Intra - sample\ similarity}$$

Here, *Inter-sample similarity* is the similarity of the query DNA sample to the closest training sample from another class, and, *Intra-sample similarity* is the similarity of the query sample to the closest training sample within its own class.

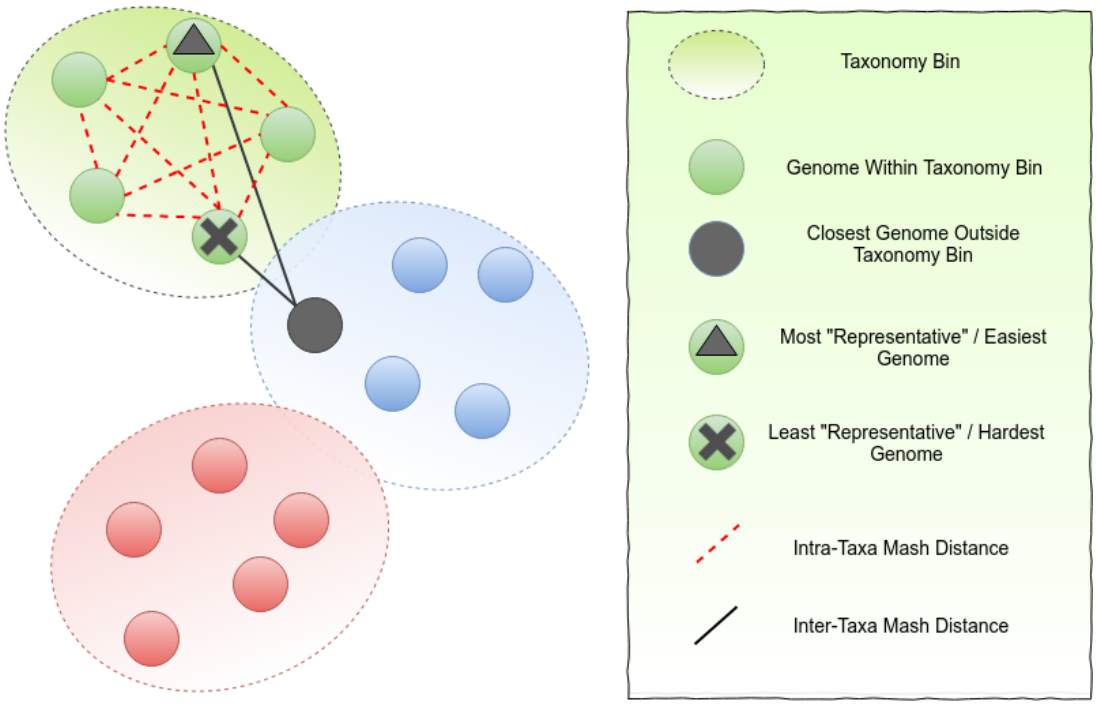

Figure 4.3: Mash hardness estimates for query DNA samples of type 1 that originate from held-out genomes used for testing

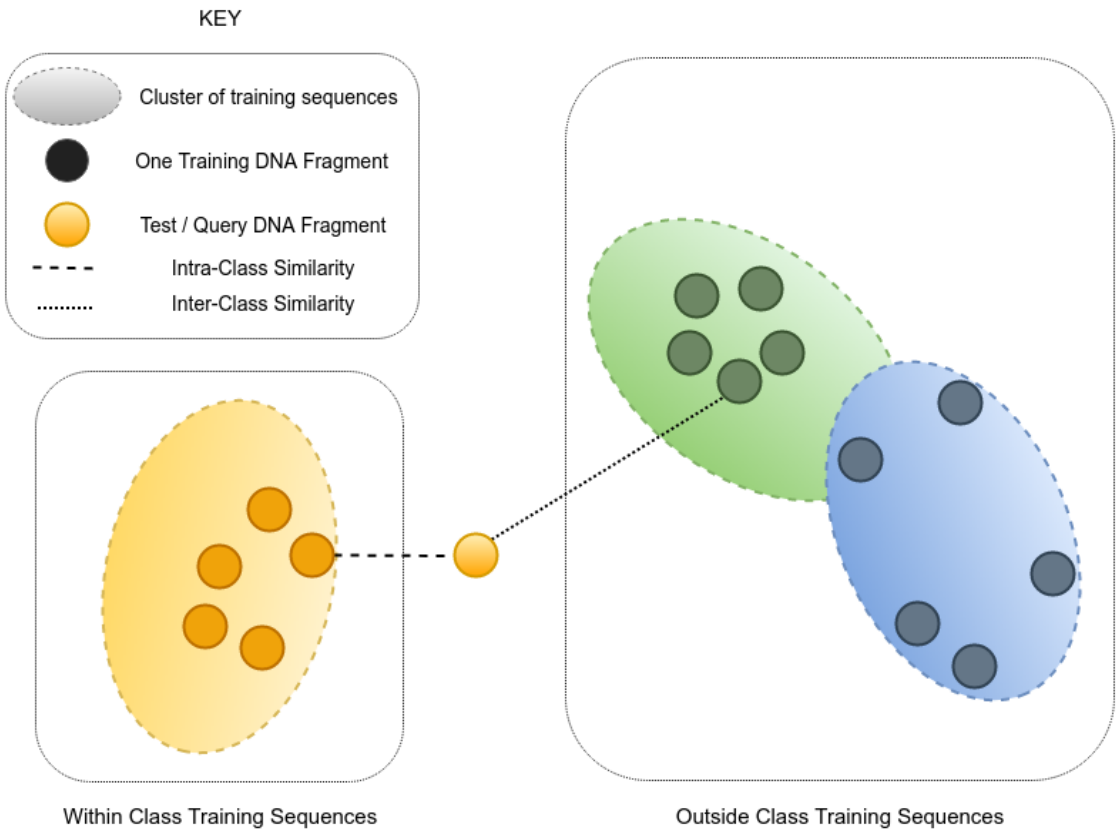

Figure 4.4: BLASTn hardness estimates for query DNA samples of type 2 that originate from unseen portions within genomes that were chosen from training

4.6 Experiment Setup and Results

4.6.1 Superkingdom Coding Sequences

The coding domain data set explained in Section 3.3.1 served as a good type 2 benchmark data set to test each algorithm. Algorithm 1 describes how the BLASTn hardness scores were obtained for each of the 1.8×10^6 query DNA fragments.

Once a hardness score for each query DNA fragment was obtained, we created 10 hardness bins that defined the relative hardness profiles of the sequences in the test set. We first

Algorithm 1 BLASTn Hardness Scores

procedure computeHardness
    initialize:
    $classwiseTrainSets \leftarrow partitionIntoClasswiseTrain(dataset)$.
    $classwiseTestSets \leftarrow partitionIntoClasswiseTest(dataset)$.
    $hardness \leftarrow list()$.
    $i \leftarrow 0$.

    loop:
    $currentClass \leftarrow classOf(classwiseTestSets[i])$.
    $queryDNA \leftarrow classwiseTestSets[i]$.

    $mostSimilarSeqOutsideClass \leftarrow BLASTnFindOut(queryDNA, classwiseTrainSets)$.
    $mostSimilarSeqWithinClass \leftarrow BLASTnFindIn(queryDNA, classwiseTrainSets)$.

    $interBitScore \leftarrow BLASTn(queryDNA, mostSimilarSeqOutsideClass)$.
    $intraBitScore \leftarrow BLASTn(queryDNA, mostSimilarSeqWithinClass)$.

    $hardness[i] \leftarrow \frac{interBitScore}{intraBitScore}$
    $i \leftarrow$i+1.
    goto *loop*.

sorted the hardness scores in increasing order, and then filled each bin such that the number of sequences in every bin was equal. Thus, sequences in hardness bin 1 were relatively the easiest to classify sequences, and sequences in hardness bin 10 were the hardest to classify.

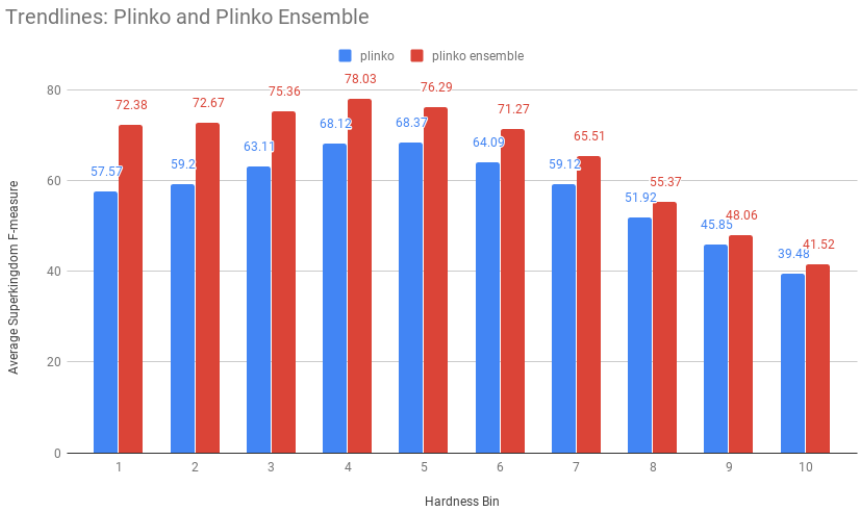

Figure 4.5: Average Superkingdom F-measure of regular Plinko and Plinko ensemble as a function of hardness.

Comparing the average superkingdom F-measure for regular Plinko and the Plinko ensemble shows that, most of the performance gains for the Plinko ensemble is realized when classifying easier sequences. This graph suggests that, in the future, the regular Plinko training procedure can be modified such that for the first few epochs of training, Plinko is only trained on "easy" sequences, and then, with each increasing epoch, Plinko is gradually introduced the harder to classify sequences. Another future direction can be to identify whether all neural network algorithms show similar trends. Answering this research question can help us understand the inner workings of these algorithms further. Based on the positive results from training Plinko ensembles, all of the following coding sequence experiments make use of the Plinko ensembles.

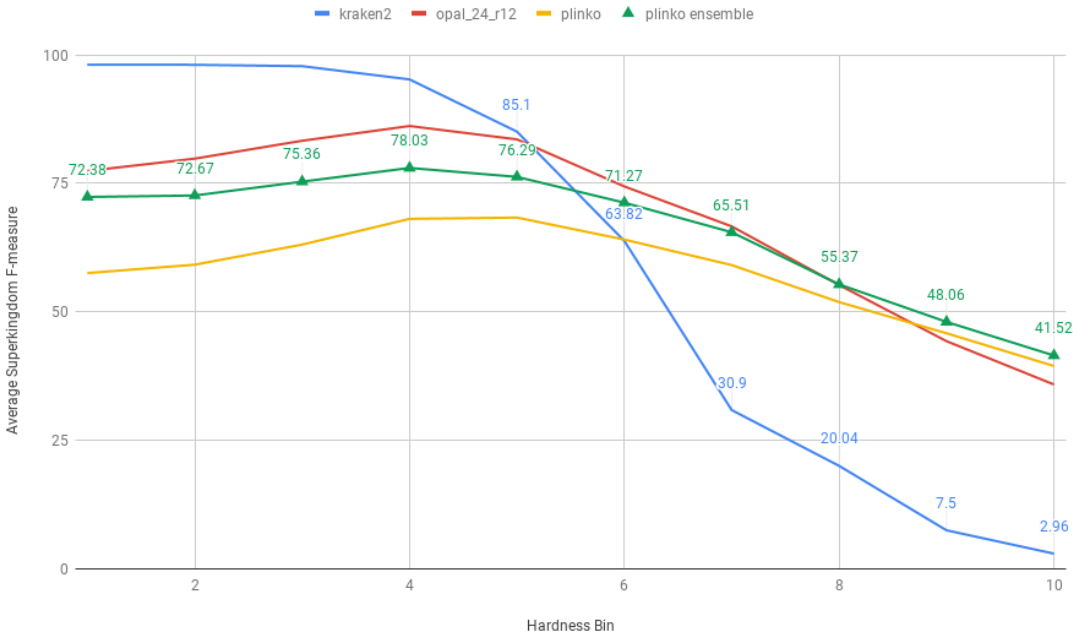

Figure 4.6: Average superkingdom F-measure for Plinko, Plinko Ensemble, kraken2 and Opal. The Plinko Ensemble is an ensemble of 11 Plinko models trained on the same data set.

Comparing each of the methods, we can see that the Plinko ensemble has the best generalization performance for the most difficult to classify test sequences. Additionally, Plinko ensemble has the lowest memory footprint compared to all the methods.

Kraken2, a method that relies solely on sequence similarity performs the best on test sequences that are very similar to training sequences belonging to it's own class. However, with decreasing similarity to sequences within the test sequence's own class, kraken2's performance degrades rapidly. This experiment showcases the true nature of database lookup algorithms in cases where sequence similarity ceases to aid in classification.

Although Opal's performance, on average, is the best amongst all methods, Opal's model

size is roughly 5 times the size of the input data set, and 31 times the size of the Plinko ensemble. Let us now look at how Opal's average F-measure are affected by a change in model training parameters.

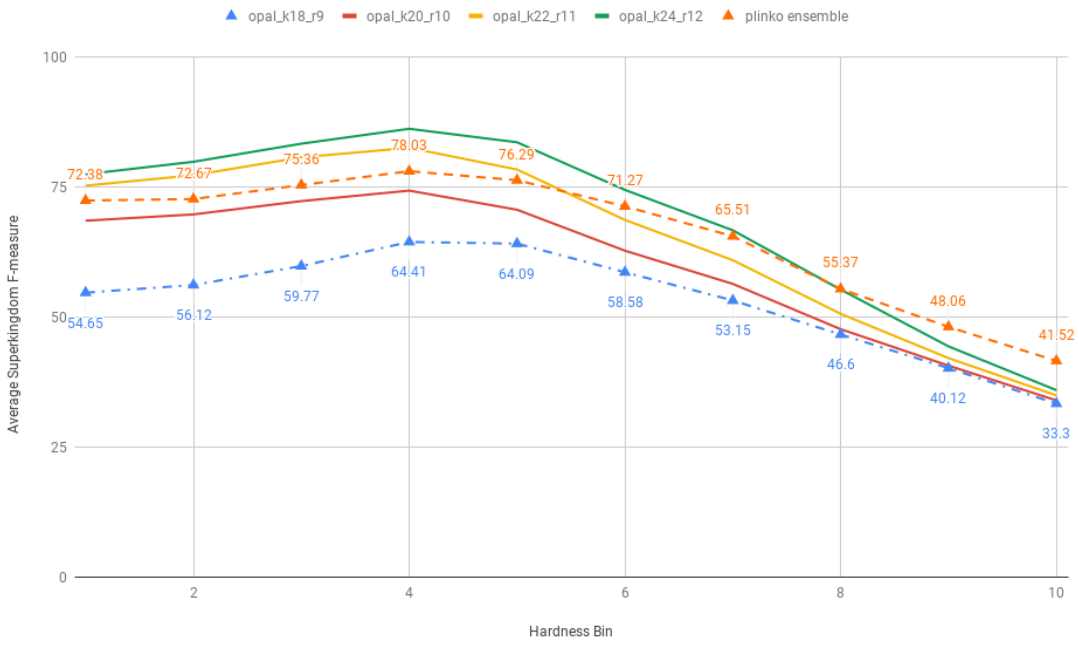

Figure 4.7: Plinko ensemble performance compared to Opal models of varying kmer size and row weight parameters.

Two major parameters that control both the accuracy and the size of the resulting models in Opal are the k-mer length and row weight. Taken together, the two parameters determine the total number of features that are stored for every OneVsAll SVM model. Thus, as the 'k' and 'row-weight' values reduce, the total memory footprint of Opal also reduces.

Figure 4.7 shows that Plinko ensemble's performance lies in between the Opal models with (k=20, r=10) and (k=22, r=11). Whereas the Plinko ensemble model size is 330 MB, Opal(k=20, r=10) has a model size of 794 MB (2.4 times the Plinko ensemble solution) and

Opal(k=22, r=11) has a model size of 2.9 GB (9 times the Plinko ensemble solution). The blue dotted line shows an Opal model (k=19, r=9) with size comparable to that of Plinko. We note from the graph above that the performance of Plinko as the sequences get harder to classify is on par with the performance of the most expensive Opal model (k=24, r=12) with a model size of 10 GB (31 times the Plinko ensemble solution). Let us now understand the performance of Plinko and kraken2.

As shown in Figure 4.8, Plinko generally surpasses kraken2 in the average superkingdom F-measure score only because it's recall is far better that that of kraken2 given any class. Even at harder to classify DNA fragments, where not only sequence dissimilarity, but also sequence confusion increases, Plinko is able to make clear distinctions between the four classes. This further strengthens our belief that Plinko can be useful when classifying novel sequences that have never been encountered before.

It is equally important to note the near perfect precision score of kraken2 on all of the easy to classify test sequences. Since kraken2 uses exact matches of k-mers to make it's prediction on the test sequences, and our hardness metric is a measure of sequence similarity, kraken2's performance is expected to be the highest. Plinko's high precision for the harder to classify Bacterial and Eukaryote sequences however strengthes our argument that Plinko learns features other than only sequence similarity to make it's predictions.

Another important observation from the graphs in Figures 4.8 and 4.9 is that Plinko's precision seems to correlate very highly with the number of sequences in every hardness bin. This trend was seen across all four classes.

Figure 4.8: Each row in the figure breakdown the performance of Plinko Ensemble and kraken2 on a particular class of origin sequences (Archaea, Virus, Bacteria, Eukaryotes). The first column shows the precision and sensitivity of Plinko as a function of the hardness profile. The first column shows the precision and sensitivity of kraken2 as a function of the hardness profile.

Figure 4.9: The graphs show the distribution of the number of test sequences as a function of hardness from each of the four taxonomy classes. Notice that viral sequences form a right skewed graph, suggesting that most of the test viral sequences are very similar to the viral sequences from the training set.

To confirm our observation of high correlation, we plotted a scatter plot between the number of test sequences in a hardness bin to Plinko's precision for that hardness bin. The scatter plot is shown in Figure 4.10. When we ran the Spearman's Correlation test on the two variables, we obtained a correlation of 85.8%, with a p-value of 1.406×10^{-12}.

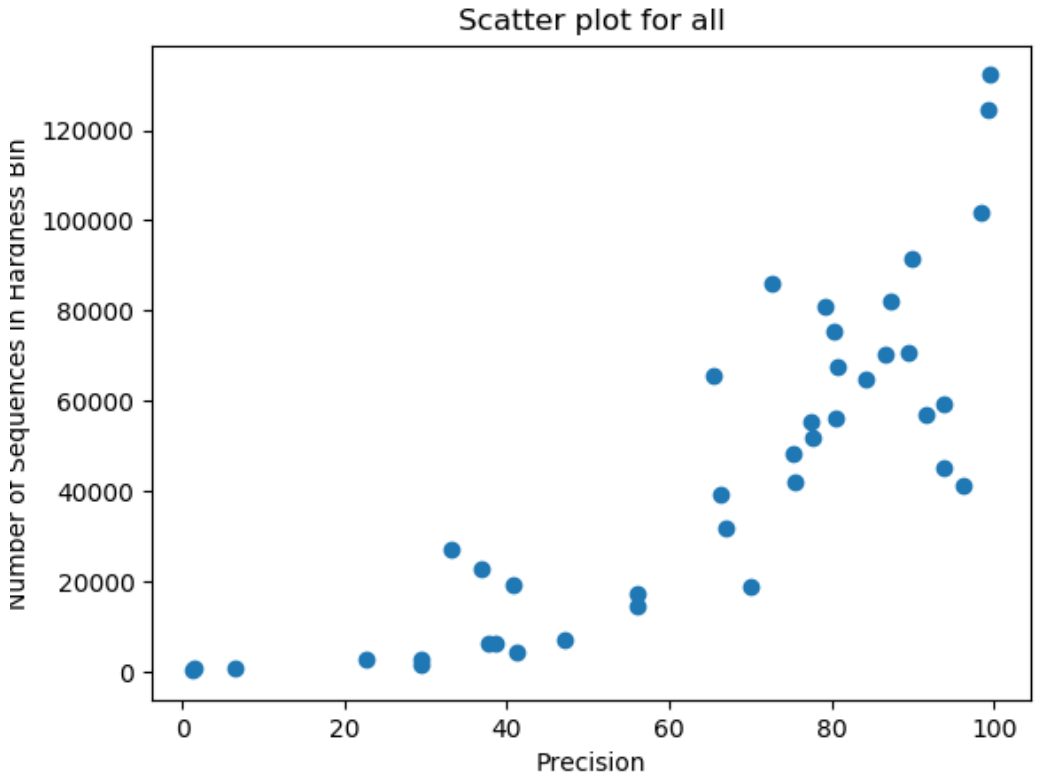

Figure 4.10: Scatter plot between the number of sequences in a hardness bin to Plinko's precision for that hardness bin

If the same trend is true for the training sequences, we believe that at least 3 factors could be responsible for the strong correlation.

1. Plinko may be using the number of sequences in a hardness bin as a feature to predict the class. Observe that the trend of number of sequences in each hardness bin is unique to every class.

2. The number of training sequences in each bin is simply insufficient to effectively learn taxonomy given the dimensionality of the feature space.

3. As hardness increases, this means that a sequence is more similar to sequences in other classes than sequences within its own class. This could be because a sequence has really come from another class. An example of this comes from viruses injecting their genes into host organisms.

To get a sense of how trustworthy Plinko's predictions are, we measured, for each of the 1.8×10^6 test coding sequences, Plinko's precision and probability (or confidence) estimate for the predicted class. The results are shown in Figure 4.11. This shows that Plinko's probability scores are very good predictors of it's precision.

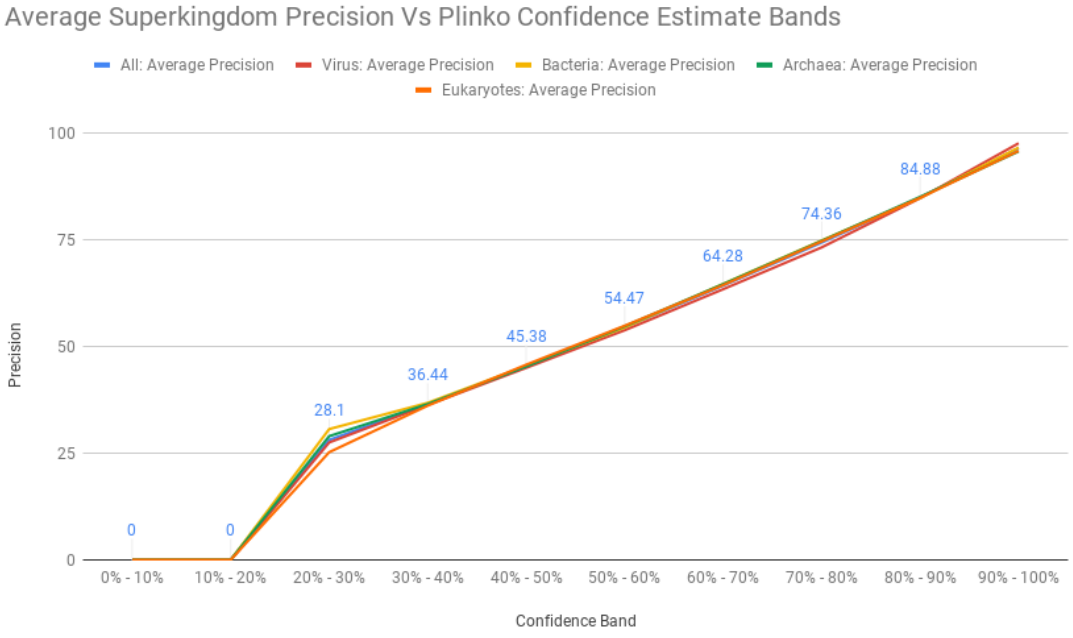

Figure 4.11: Trustworthiness of Plinko's predictions. The x-axis shows 10 probability or confidence ranges. The y-axis shows Plinko's average precision for sequences lying in that band. All four classes have similar trends

4.6.2 Opal and Plinko Ensembles

Considering that Opal classifies easy sequences very well and that Plinko is resilient to hard to classify sequences, we now investigate whether there is benefit in ensembling the two methods. For the Superkingdom coding sequences data set, we created an ensemble of the Opal model and Plinko model using Gradient Boosted Decision trees. The decision trees were trained with probability estimates from Plinko and Opal on each of the 10^7 training sequences. The results are shown in Figure 4.12.

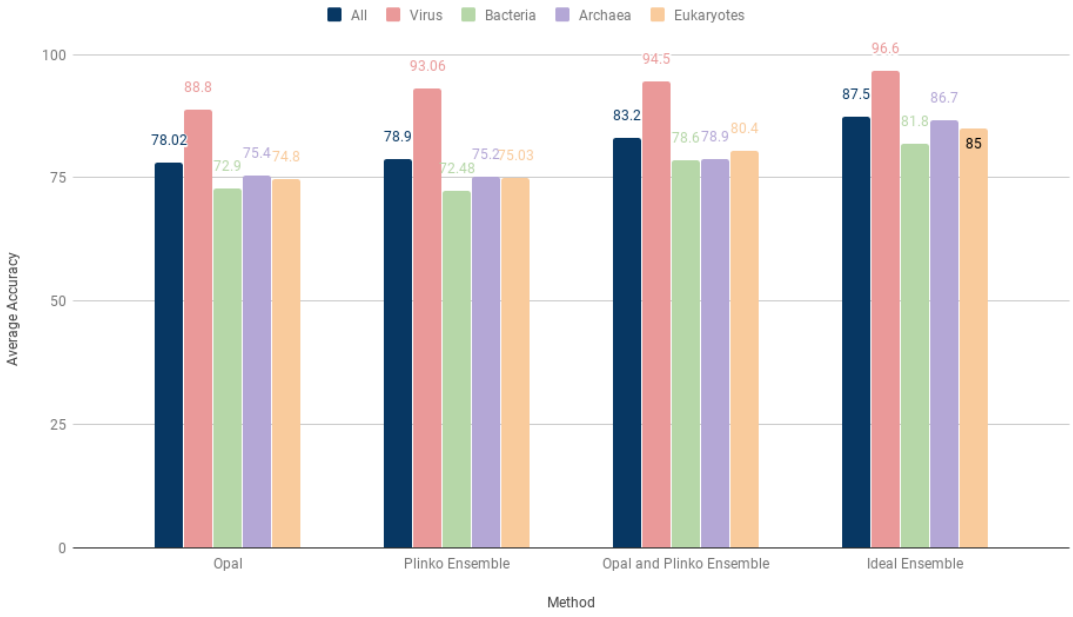

Figure 4.12: Comparing the performance of the Opal and Plinko ensemble solution using gradient boosting and an ideal ensemble

In Figure 4.12, the ideal ensemble refers to a solution which is able to always correctly identify which of the two predictions to trust. However, in practice it is often the case that the ensemble performance falls short of this ideal performance. Nevertheless, our initial results are encouraging, and we show that ensembling Opal and Plinko can attain at least 5% of the total 9% improvements to be had.

4.6.3 Superkingdom Coding and Non-Coding Sequences

To determine patterns that exist outside coding sequences, a data set containing length 100 DNA fragments were generated with the program Radogest. Each length 100 DNA fragment could be either coding, non-coding, mixed (partially coding and partially non-coding) or unknown. Coding DNA is DNA that codes for a gene and non-coding DNA is DNA that does not. These DNA fragments were generated by randomly sampling from genomes downloaded from the National Center for Biotechnology Information. A training data set of 1.8×10^7, 100 length DNA fragments were drawn where approximately 4.5×10^6 fragments were sampled from each superkingdom (Bacteria, Archaea, Eukaryotes, Virus). A test set and a validation set each consisting of 2×10^6 length 100 DNA fragments were sampled in the same manner as the training set.

The training data set was used to train Plinko, and the validation data set was used to determine how many epochs Plinko was trained. The epoch with the highest validation accuracy was used to select the Plinko model. The test data set was used to report accuracy. It should be noted here that the data set was balanced in terms of number of training and test sequences from each of the four superkingdom labels. However, they remain skewed in terms of number of sequences that were coding sequences, number of sequences that were non-coding etc. The statistics for the test set are shown in Table 4.1.

In Section 4.6.1 we saw that there was a strong correlation between the number of sequences in a hardness bin (across all four classes) and Plinko's precision. However, it was unclear if

Table 4.1: Number of test sequences from each category in Superkingdom Coding and Non-Coding sequences data set

Sequence Type	***Number of Sequences***
Coding	1036640
Non-Coding	250602
Mixed (Coding + Non-Coding)	183523
Unknown	530026

the trend was limited to coding sequences alone. To make sure that we were seeing a real trend independent of the type of DNA sequence (coding or non-coding), we characterized the hardness of each test sequence in the Superkingdom Coding and Non-Coding data set using Algorithm 1. After calculating the hardness scores, the test sequences were separated into two categories (i) Coding and (ii) Non-coding. For each category we ran the same analysis as the one presented in 4.6.1. As we see in Figures 4.13 and 4.14, the same trend of number of sequences in a hardness bin correlating with precision is seen for both coding and non-coding sequences.

4.6.4 Bacilli Sequences

We selected genomes from 43 species of the taxonomy class Bacilli (TaxID: 91061). These 43 species were a subset of 190 species used in the evaluation of Opal (Luo et al. [16]) and included multiple species from the Bacillus cereus sensu lato group that have been shown to be challenging to classify at the species level (Zwick et al. [28]). A data selection procedure was performed to sub-select at most five genomes from each of the 43 species in the priority order:

1. Representative Genomes

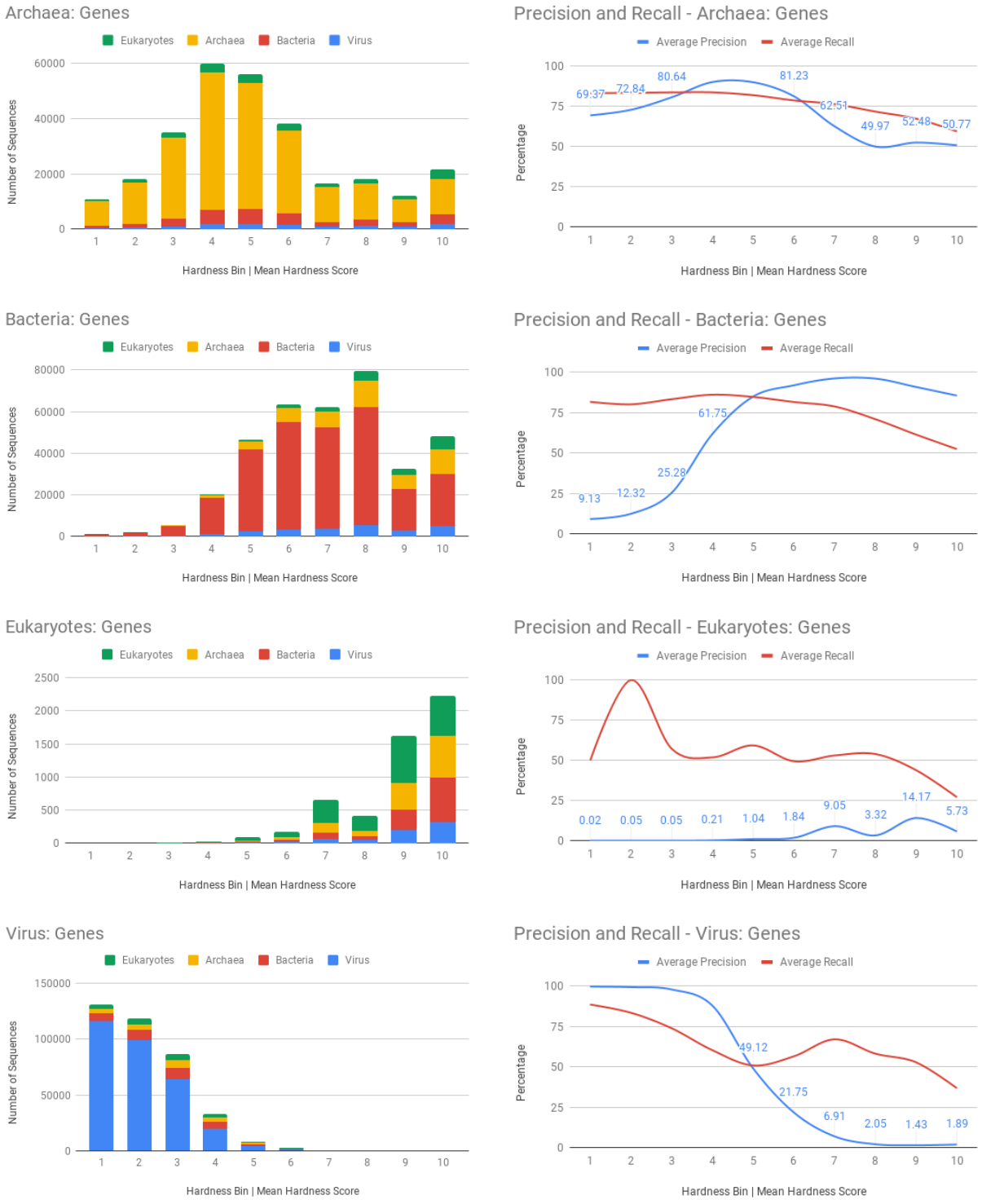

Figure 4.13: Analysis on coding test sequences from the superkingdom coding and non-coding sequences data set. Each row breaks down the performance of Plinko on a particular class of origin sequences (Archaea, Virus, Bacteria, Eukaryotes). The first column shows the confusion profile of Plinko on each class as a function of hardness. The second column shows the precision and sensitivity of Plinko as a function of the hardness.

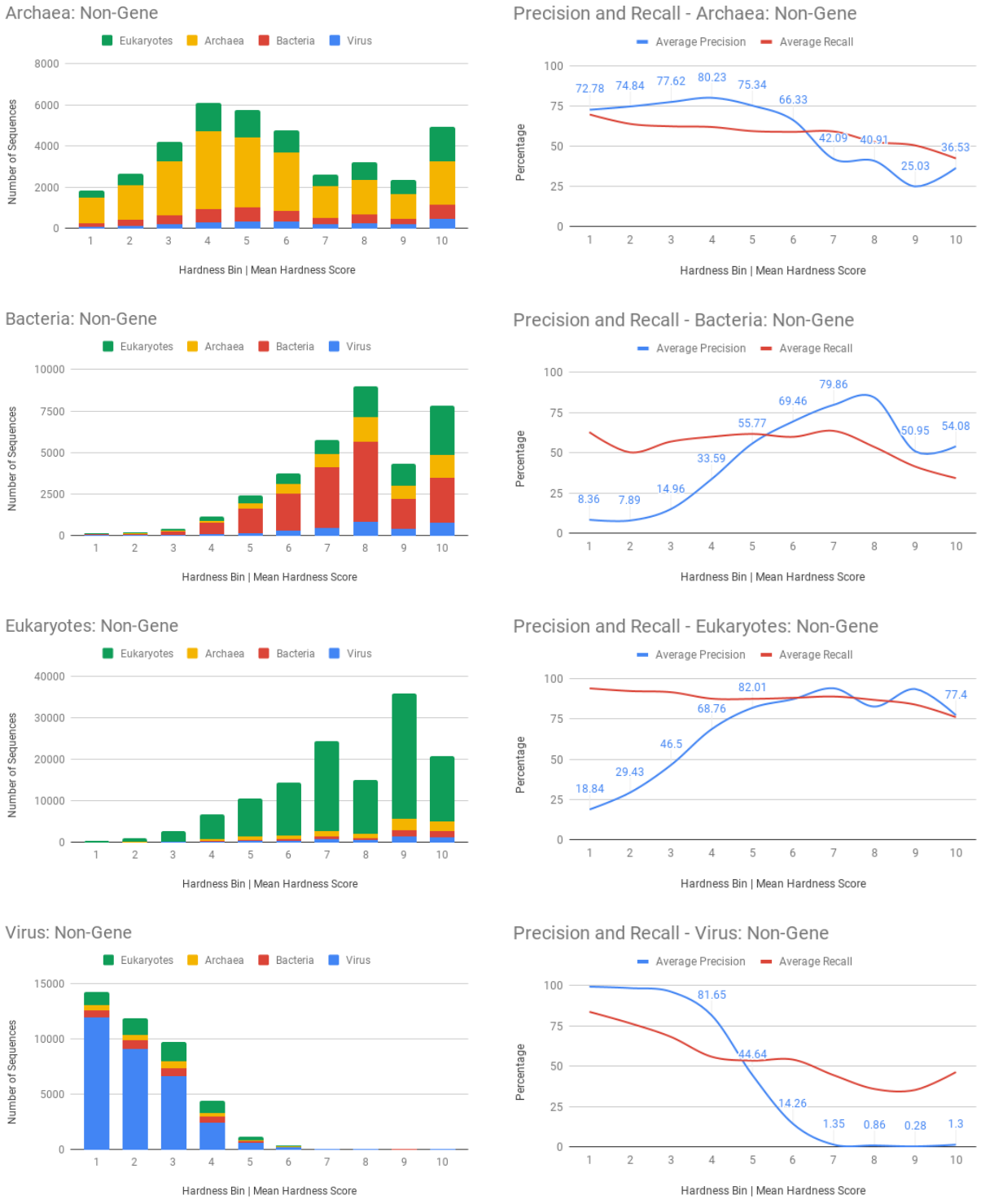

Figure 4.14: Analysis on non-coding test sequences from the superkingdom coding and non-coding sequences data set. Each row breaks down the performance of Plinko on a particular class of origin sequences (Archaea, Virus, Bacteria, Eukaryotes). The first column shows the confusion profile of Plinko on each class as a function of hardness. The second column shows the precision and sensitivity of Plinko as a function of the hardness.

2. Reference Genomes

3. Genomes with more than 10^6 base pairs, sorted in increasing number of contigs

In other words, we first filled up each species' bucket with representative genomes. If there was space left in the bucket, then we filled it with reference genomes. If there was still space left, then we use the "highest quality" genomes, added in the order of increasing number of contigs.

The DNA fragments used to train each method were obtained by extracting all of the non-overlapping sequences of length 100 from contigs within a taxon node's FASTA file. Each 100 length DNA fragment was then associated with the TaxID of the contig from which the sequence was extracted.

Table 4.2: Basic, Easy and Hard Dataset Training and Testing Information

Dataset Type	***Total Number of Genomes***	***Number of Genomes in training set***	***Number of Genomes in test set***
Basic Data Set	210	210	20% of training set
Easy Data Set	167	43	43 "Most Representative" genomes
Hard Data Set	167	43	43 "Least Representative" genomes

Basic Data Set - No Genome Held Out for Testing

The traditional operational framework for Kraken is that it is able to recognize k-mers from genomes which it had already been provided during the construction of it's database. To represent this case, we created a designated "Basic" test data set that contains 20% of the DNA fragments from the training set i.e. 100% of the sequences from the 210 genomes are used during training and a random 20% of all the sequences are re-used for testing. This is not how machine learning classifiers are traditionally evaluated, since it does not test if

the models generalize to new, unseen examples. The Basic test data set, however, helps to establish an upper bound on performance for each of the three methods and will still be impacted by the hardness criteria described below.

Easy Data Set - Most Representative 43 Genomes Held Out for Testing

To create the easy test data set, first, using the Mash software (Ondov et al. [22]), the all-to-all genome Mash distance was estimated with a sketch size of 100,000 and k-mer size of 21. Next, for every species cluster, the *maximum* Mash distance from each of the five genomes to all other genomes within the same species cluster was computed. This resulted in five Mash distances per species cluster. The genome that had the *minimum* of the five distances Mash distances calculated above was the *easiest and "Most Representative"* genome of the species cluster. The intuition behind this being - this genome is the closest to all the genomes within the same species cluster, and thus, holding out this genome from the training set will lead to fewer misclassifications.

This genome was held out during training, and the largest outgoing Mash distance from this genome was defined as the Intra-Mash distance. The Inter-Mash distance was calculated as the Mash distance between this held out genome and the closest genome from another species cluster. Finally, the hardness measure for a held out genome was calculated as:

$$Hardness = \frac{1 - InterMashDistance}{1 - IntraMashDistance}$$

This resulted in a training set of 167 genomes, and a test set consisting of 43 of the most representative genomes, one from each of the 43 species.

Hard Data Set - Least Representative 43 genomes Held Out for Testing

Similarly, to create the hard test data set, first, using the Mash software, the all-to-all genome Mash distance was estimated with a sketch size of 100,000 and k-mer size of 21. Next, for every species cluster, the *minimum* Mash distance from each of the five genomes to all other genomes within the same species cluster was computed. This resulted in five Mash distances per species cluster. The genome that had the *maximum* of the five Mash distances calculated above was the *hardest and "Least Representative"* genome of the species cluster. The intuition behind this being - this genome is the furthest to all the genomes within the same species cluster, and thus, holding out this genome from the training set will lead to high misclassification rates.

This procedure resulted in a training set consisting of 167 genomes, and a test set consisting of 43 of the least representative genomes, one from each of the 43 species.

Now, we present results from comparing kraken, Opal and Plinko on the three bacilli test data sets.

Table 4.3: F1 Scores of Kraken, Opal and Plinko on the three bacilli sequences data sets

Method	***Basic Data Set***	***Easy Data Set***	***Hard Data Set***
Kraken	**98.52%**	93.63%	76.06%
Opal	97.25%	**93.47%**	89.85%
Plinko	94.24%	92.04%	**90.39%**

Both Figure 4.15 and Table 4.3 show that as the "learnability" of taxonomy labels decreases or "hardness" of classification increases, both machine learning methods perform better than kraken. Additionally, the confidence bands show that kraken's predictions are characterized by high variance. This is because kraken may not have the exact queried DNA sequence stored in it's database. On the other hand, Plinko and Opal have very tight confidence

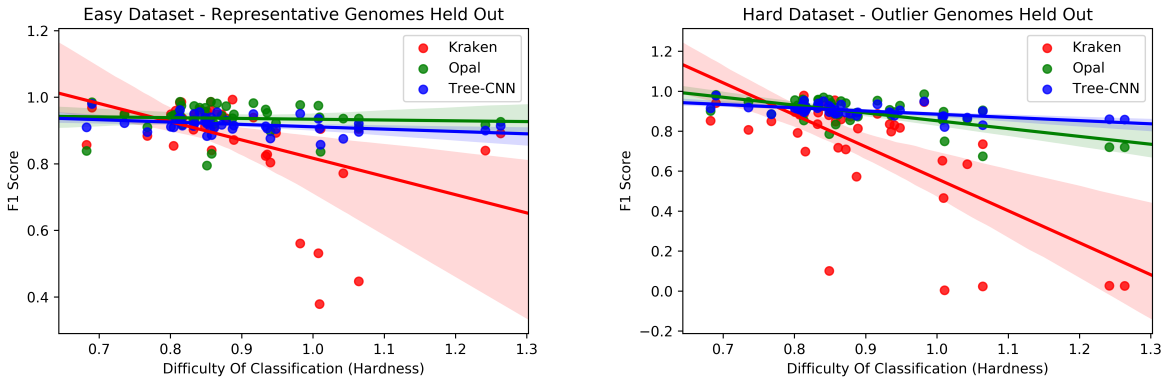

Figure 4.15: Graphs showing the F1 Score and Hardness characteristics of Kraken, Opal and Plinko when queried with DNA strings from the easy and hard data sets. Confidence bands are shown for each method in a lighter color.

bands, showing that these methods may be more trustworthy for unseen, novel sequences. Finally, Plinko outperforms both Opal and kraken at the most difficult to classify case in both the easy and hard data sets.

The memory footprint of CNN models depends on the number of parameters to be trained and k-mer size. CNN model sizes thus do not depend on the number of input sequences used to train the model. Hence, model size is simply the product of the number of taxonomy nodes in the tree and the size a single model, assuming constant k-mer length.

We compared the size of the three methods based on how they scaled to (i) the number of nucleotides and (ii) the number of species labels in a data set containing 100 out of 190 species used in the evaluation of Opal. We saw that CNN models have the smallest memory footprint for all of the data set size ranges.

The image on the right of Figure 4.16 represents the size of the three methods as a function

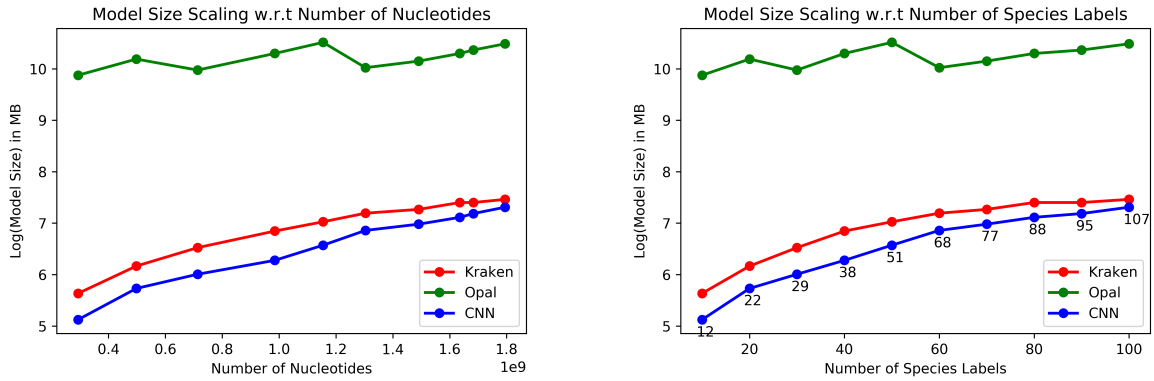

Figure 4.16: Model size of the Kraken, Opal and Plinko at different dataset sizes. The figure on the left shows classifier size as a function of the number of bases, the figure on the right as a function of number of species.

of the number of species labels in the data set. Each point in the graph represents the total number of taxonomy nodes in the data set. This number is used to calculate the size of Plinko. We thus conclude from these experiments that, the Plinko model scales to increasing data set sizes while also providing high precision and sensitivity for hard to classify, novel DNA sequences.

4.6.5 All of Bacteria Sequences

In this section, we discuss the creation of a type 1 data set that simulates real-world taxonomy prediction. With this data set, our overall goal was to measure the performance of algorithms when queried with whole gene sequences (not restricted to length 100) from novel, unseen genomes.

We now provide the step-by-step procedure on how we obtained the all bacteria data set.

1. First, we downloaded all complete bacterial genomes from RefSeq (February 26, 2018 release). There were 6343 unique genomes downloaded.

2. Next, we selected all Representative and Reference genomes from the downloaded complete bacterial genomes. There were 750 unique genera for this set. For each of the 750 genera we chose up to 5 genomes of each species under each genus.

3. Filtration Step: We removed all genera that contain only one genome. The total *number of genera* that were left after filtration was 444 and the *total number of genomes* left after filtration was 4213.

4. From the remaining genomes, we calculated the *total number of species* that have more than 1 genome to be 571.

5. We held out 10% of the data set which is roughly 420 genomes. The remaining 3793 genomes were used to train the models.

6. The total size of 3793 training genomes was 24 GB.

Four test data sets that comprised of gene sequences from the 420 held out genomes were created to test the methods. They are described below.

- **Easy Set**: This set consisted of 10^6 randomly chosen coding sequences of length between 200 and 500 Base Pairs.

- **Medium Set**: 2.1×10^6 randomly chosen coding sequences of length between 100 and 200 Base Pairs.

- **Hard Set**: 2.6×10^5 coding sequences of length between 0 and 100 Base Pairs.

- **All Difficulty**: 1.4×10^6 randomly chosen variable length coding sequences.

Table 4.4: Statistics of Genomes used to create the All Bacteria data set

Rank	***Number of Unique Nodes***	***Average Number of Genomes in Rank***
Species	3050	1.38
Genus	444	9.48
Family	192	21.8
Order	93	45.17
Class	39	105.92
Phylum	21	200.61
Superkingdom	1	4213

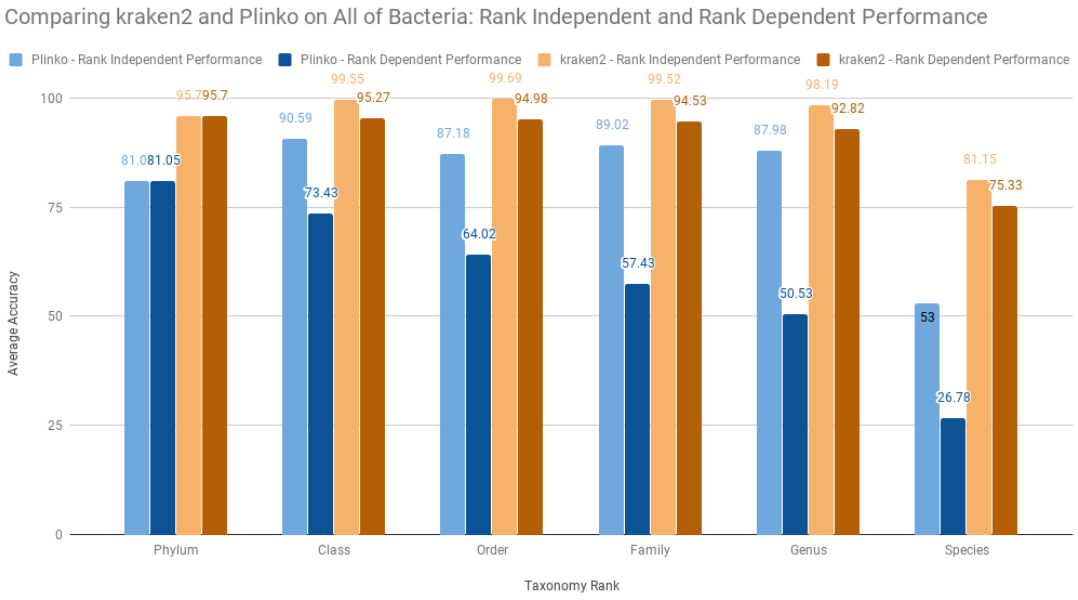

Figure 4.17: Rank independent and rank dependent prediction performance of Plinko and kraken2 on the All Difficulty test dataset at all taxonomy ranks

Figure 4.17 shows the performance of Plinko and kraken2 on the "All Difficulty" test data set consisting of variable length coding sequences. Since Plinko's inference strategy is based on routing test fragments level-by-level down the tree, we describe the performance of both methods using two measures: Rank independent and rank dependent prediction performance. Rank independent performance at a taxonomy rank refers to the performance at only that rank, regardless of the predictions made at previous taxonomy ranks. Rank dependent per-

formance at a taxonomy rank refers to the performance of a method at that taxonomy rank, given the cumulative performance of the method in all previous taxonomy ranks.

In general, although Plinko achieves an average classification accuracy of 87% up to the taxonomy rank genus, the effect that the error accumulation down the tree has on Plinko's performance is evident. A future direction of Plinko can be to develop an efficient inference strategy that avoids this error accumulation at the lower ranks of the taxonomy.

To figure out where Plinko can be useful in taxonomy prediction, it may be useful to take a second look at the test sequences that kraken2 fails to classify.

Table 4.5: Table showing the performance of kraken2 on the easy, medium and hard test data sets

Data Set Type	***Total Number of Sequences***	***Fraction Unclassified***	***Number of Sequences Unclassified***
Easy	1.0×10^6	4.56%	1.2×10^4
Medium	2.12×10^6	5.98%	1.27×10^5
Hard	2.62×10^5	5.72%	5.7×10^4

We see in Table 4.5 that there are on the order of 10^4 to 10^5 test sequences that are consistently missed by kraken2. As we saw earlier in Section 4.6.1, Plinko has the unique ability to classify hard sequences that are incorrectly predicted by kraken2. These harder to classify sequences are where Plinko can aid kraken2 in taxonomy classification. Let us now take a look at performance of a Plinko assisted kraken2 solution on the three variable difficulty data sets.

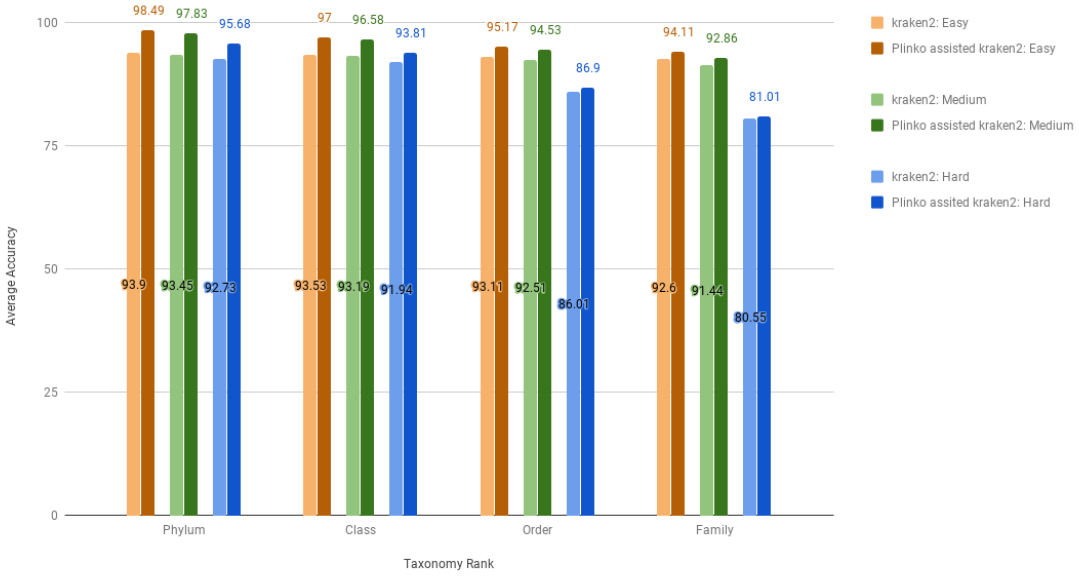

Figure 4.18: Comparing performance of kraken2 and Plinko assisted kraken2

As we see in Figure 4.18, there are gains to be had with a Plinko assisted kraken2 solution at the top four levels in the taxonomy tree, regardless of the sequence length. Additionally, since the memory footprint of the Plinko solution is a fraction of the overall kraken2 solution, we believe this merger may prove to be very useful, especially for novel, hard to classify sequences.

Table 4.6: Training performance of kraken2 and Plinko on the 'all difficulty' test data sets

Method	*Configuration*	*Memory Usage (GB)*	*Size (GB)*	*Training Time (hours)*	*F-measure*	*Precision*	*Sensitivity*
Plinko Parallel Train	20 GPUs	221	**18.1**	168	35.52	90.28	30.58
kraken2	20 CPUs	42	**73**	1.51	42.19	95.1	37.2

Table 4.6 and 4.7 show the training and inference performance of Plinko and kraken2. Plinko uses GPUs to build models and perform inference while kraken2 uses CPUs to do the same. Plinko's main advantage over kraken2 is it's model size of just 18 GB, while kraken2's solution exceeds 70 GB. During inference, Plinko using two GPUs is faster than kraken2 with 20 CPU threads running in parallel. However, with respect to the F-measure, Plinko in it's current

Table 4.7: Inference performance of kraken2 and Plinko on the 'all difficulty' test data sets. One machine with 20 CPU cores and 4 NVidia V100 GPUs were used for the test.

Method	***System Memory Usage (GB)***	***GPU Memory Usage (GB)***	***Speed (sequences / min)***
Plinko, 1 GPU	3	7	1.01×10^5
Plinko, 1 GPU	4.8	14	1.67×10^5
Plinko, 1 GPU	8.1	28	2.35×10^5
Plinko, 2 GPUs	12.9	56	4.7×10^5
Plinko, 4 GPUs	21.6	102	9.41×10^5
kraken2, 20 threads	21	0	3.36×10^5

form is still not competitive with kraken2 on the All Bacteria data set, proving that machine learning approaches to taxonomy classification are still limited to small to medium scale data sets. Plinko can nevertheless be helpful to perform accurate taxonomy classification when sequence similarity approaches like kraken2 are not able to reach a solution. We believe that a pipelined approach where, kraken2 is first applied to classify sequences based on sequence similarity alone, and the unclassified sequences from kraken2 are classified by a GPU parallel Plinko, will help in achieving state-of-the-art precision, sensitivity and inference times.

Chapter 5

Plinko Inference Properties and Conclusion

5.1 Plinko Inference Speed

Taxonomy prediction algorithms need to be both accurate and fast in classifying DNA fragments. Let us now try to get a better sense of the inference performance of the n-ary Plinko tree algorithm in detail.

In our analysis, we assume the following:

1. Every Plinko MVCNN network in the n-ary decision tree has the same architecture.
2. The length of the input DNA fragment is represented by L.
3. k-mer length for nucleotide sequences is represented by K, and, k-mer length for amino acid sequences is represented by K'.
4. Both nucleotide and amino acid k-mer embedding dimensionality is represented by e.
5. The convolution kernel length is represented by k.
6. Number of convolutional kernels is represented by m.

7. Number of output labels for a network is represented by o.

8. Nucleotide and Amino Acid word index dictionaries have been pre-loaded into system memory.

With the above assumption set, let us now break down the time complexities of the major computations necessary to classify an input DNA fragment of length L using one Plinko MVCNN network. The time complexity to classify a DNA fragment by the entire Plinko n-ary tree will simply be scaled by the maximum depth h of the tree.

5.1.1 Stage 1: Input Preprocessing

The major computations performed at this stage are as follows:

1. Compute reverse complement of the input DNA fragment. $[\mathcal{O}(L)]$

2. Lookup word index dictionary that associates every nucleotide k-mer with a row index in the nucleotide embedding matrix. Do this for both the input DNA fragment and it's reverse complement. $[\mathcal{O}(L-K+1)+\mathcal{O}(L-K+1)]$

3. Compute all six frame amino acid translations of the input DNA fragment using translation table 11. $[6\mathcal{O}(L)]$

4. Concatenate all six frame translation strings. $[\mathcal{O}(1)]$

5. Lookup word index dictionary that associates every amino acid k-mer with a row index in the amino acid embedding matrix. Do this for the concatenated six frame translations string $[6\mathcal{O}(\frac{L}{3}-K'+1)]$

The total time complexity from stage 1 is given by:

$$\mathcal{O}(L) + \mathcal{O}(L - K + 1) + \mathcal{O}(L - K + 1) + 6\mathcal{O}(L) + 6\mathcal{O}(\frac{L}{3} - K' + 1)$$

The above equation reduces to:

$$3\mathcal{O}(L) + 6\mathcal{O}(L) + 6(\frac{L}{3} - K' + 1) \approx 10\mathcal{O}(L)$$

5.1.2 Stage 2: Plinko Forward Propagation

Knowing that most operations in neural networks are dot products, and in order to make our computation of the time complexity of a Plinko neural network inference step simpler, we assume that the fused multiply-add operation (or MACC operation) takes $\mathcal{O}(1)$ time. Every fused multiply-add operation roughly equates to 2 FLOPS. We also assume time complexities of all neural network layers other than the convolution layers and fully-connected layers as a constant. This includes the biases, activation layers, batch normalization, dropout etc. We can safely treat them as constants because, on average, they take an order of 3 times lesser time to compute than convolution and fully-connected layers.

Nucleotide input layer (Applied to both Nucleotide and Nucleotide reverse complement)

1. Input sentence matrix construction. $[\mathcal{O}(e(L - K + 1))]$
2. Convolution 2D layer that performs $(kem((L - K + 1) - k + 1))$ MACC operations.
3. 1-Max Pooling. $[\mathcal{O}(1)]$

Amino Acid input layer

1. Input sentence matrix construction. $[\mathcal{O}(6e(\frac{L}{3} - K' + 1))]$

2. Convolution 2D layer that performs $(kem((2L - K' + 1) - k + 1))$ MACC operations.

3. 1-Max Pooling. $[\mathcal{O}(1)]$

One-Hot input layer (Applied to both Nucleotide and Nucleotide reverse complement)

1. Input matrix construction. $[\mathcal{O}(4L)]$

2. Convolution 1D layer that performs $(4km(L - k + 1))$ MACC operations.

3. 1-Max Pooling. $[\mathcal{O}(1)]$

The total time complexity after applying the Plinko input layers will then be:

$$2[\mathcal{O}(e(L - K + 1)) + \mathcal{O}((kem((L - K + 1) - k + 1))]+$$

$$\mathcal{O}(6e(\frac{L}{3} - K' + 1)) + \mathcal{O}(kem((2L - K' + 1) - k + 1))+$$

$$2[\mathcal{O}(4L) + \mathcal{O}(4km(L - k + 1))]$$

Since L $\gg$ K, K' and k, we can simplify the above equation into

$$2[\mathcal{O}(eL) + \mathcal{O}(keLm)] + [\mathcal{O}(2eL) + \mathcal{O}(ke2Lm)] + [2[\mathcal{O}(L) + \mathcal{O}(kLm)]$$

Taking the integer constant out of each component from equation, we get

$$[\mathcal{O}(eL) + \mathcal{O}(keLm)] + [\mathcal{O}(eL) + \mathcal{O}(keLm)] + [\mathcal{O}(L) + \mathcal{O}(kLm)]$$

Removing common terms, we get

$$\mathcal{O}(eL) + \mathcal{O}(keLm) + \mathcal{O}(L) + \mathcal{O}(kLm)$$

$$\approx \mathcal{O}(L(1+e)) + \mathcal{O}(kLm(1+e))$$

The output from each input layer will be 1-dimensional vectors of length m. Each vector originating from the five input layers is concatenated into one long 1-dimensional vector represented by D. The vector D is passed through a series of convolution 1D layers, where each Convolution 1D layer performs $(km(D-k+1))$ MACC operations.

Representing the number of convolutional layers following the input layers as c, we have,

$$\mathcal{O}(cm(k(D-k+1)))$$

Again, since $D \gg k$, we can simplify equation into

$$\mathcal{O}(cDm)$$

A series of f fully-connected layers are applied to the output of the c^{th} 1D convolution layer. Each of these fully-connected layers performs approximately (m^2) MACC operations.

For the series of f fully connected layers, the total time complexity comes out to be

$$\mathcal{O}(fm^2)$$

The f^{th} fully-connected layer is finally flattened into a 1-dimensional vector and connected

to a softmax function. This step has a time complexity of

$$\mathcal{O}((\frac{moD}{2^c}))$$

Together, the convolution and fully-connected layers, upto the last layer have a time complexity

$$\mathcal{O}(cDm) + \mathcal{O}(fm^2) + \mathcal{O}((\frac{moD}{2^c}))$$

$$\mathcal{O}(m((cD) + (fm) + (\frac{oD}{2^c})))$$

Thus, the total time complexity for one Plinko inference operation on a DNA fragment of length L is given by

$$\mathcal{O}(L(1+e)(1+km) + m((cD) + (fm) + (\frac{oD}{2^c})))$$

For the complete n-ary Plinko tree time complexity, we simply scale this operation by the maximum depth h of the tree to get

$$\mathcal{O}(h(L(1+e)(1+km) + m((cD) + (fm) + (\frac{oD}{2^c}))))$$

5.2 Plinko Inference Memory Usage

Let us now look at the amount of GPU memory expended in classifying one DNA fragment of length 100 by one Plinko MVCNN network with nucleotide k-mer length 6 and amino acid k-mer length 3. Also, we assume here that the number of output labels is 5.

————————-NUCLEOTIDE SENTENCE MATRIX————————————

INPUT_NT_EM: **95x100x1** memory: 95*100*1 = 10K params: 409,700

CONV2D: **86x1x256** memory: 86x1x256 = 22K params: 256,256

MAXPOOL: **1x1x256** memory: 256 params: 0

————————-NUCLEOTIDE REVERSE COMPLEMENT SENTENCE MATRIX—-

INPUT_NTREV_EM: **95x100x1** memory: 95*100*1 = 10K params: 0 (shared)

CONV2D: **86x1x256** memory: 86x1x256 = 22K params: 0 (shared)

MAXPOOL: **1x1x256** memory: 256 params: 0 (shared)

————————-AMINO ACID SENTENCE MATRIX————————————

INPUT_AA_EM: **204x100x1** memory: 204*100*1 = 20K params: 1,278,200

CONV2D: **195x1x256** memory: 195x1x256 = 50K params: 256,256

MAXPOOL: **1x1x256** memory: 256 params: 0

————————ONE HOT NUCLEOTIDE MATRIX————————————-

INPUT_NT_OH: **100x4** memory: 100*4 = 400 params: 0

CONV2D: **91x1000** memory: 91x1000 = 91K params: 41,000

MAXPOOL: **1x1x1000** memory: 1K params: 0

————————ONE HOT NUCLEOTIDE REVERSE COMPLEMENT MATRIX—

INPUT_NT_OH: **100x4** memory: 100*4 = 400 params: 0 (shared)

CONV2D: **91x1000** memory: 91x1000 = 91K params: 0 (shared)

MAXPOOL: **1x1x1000** memory: 1K params: 0 (shared)

CONCAT: **2768x1** memory: 3K params: 0

CONV1D: **2759x128** memory: 2759x128 = 353K params: 1408

MAXPOOL: **1379x128** memory: 1379x128 = 176K params: 0

CONV1D: **1370x128** memory: 1370x128 = 175K params: 163,968

MAXPOOL: **685x128** memory: 685x128 = 88K params: 0

CONV1D: **676x64** memory: 676x64 = 43K params: 81,984

MAXPOOL: **338x64** memory: 338x64 = 22K params: 0

CONV1D: **329x64** memory: 329x64 = 21K params: 41,024

MAXPOOL: **164x64** memory: 164x64 = 10K params: 0

FC: **164x256** memory: 164x256 = 42K params: 16,640

FC: **164x64** memory: 164x64 = 11K params: 16,448

FLATTEN: **10496** memory: 10496 = 10K params: 0

FC: **1x5** memory: 1x5 = 5 params: 52,485

TOTAL MEMORY: 1.273M*4 bytes $\approx$ 5.092MB per DNA Fragment

TOTAL PARAMS: 2,618,185

5.3 Conclusion

In this thesis, we introduced Plinko, a state-of-the-art, large-scale deep learning architecture for taxonomy prediction. We showed how Plinko's models make use of information encoded within DNA to improve classification accuracy. We showed the advantage of using word embeddings and how Plinko makes use of different word usage patterns corresponding to different levels of evolutionary divergence. Plinko consistently outperformed machine learn-

ing and non machine-learning methods when identifying the taxonomy of novel, hard to classify sequences in more than one experiment. We strongly believe that Plinko's inference time and memory efficiency characteristics make it a good candidate taxonomy prediction method that can be deployed on both personal computers as well as high performance computers.

Bibliography

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Chong Wang, Jue Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *ICML*, 2016.

[2] Md. Monowar Anjum, Ibrahim Asadullah Tahmid, and M. Sohel Rahman. Chilenpred: Cnn model with hilbert curve representation of dna sequence for enhancer prediction. *bioRxiv*, 2019. doi: 10.1101/552141. URL https://www.biorxiv.org/content/early/2019/02/27/552141.

[3] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13 (4):27–31, Oct 1994. ISSN 0278-6648. doi: 10.1109/45.329294.

[4] Ammar Belatreche. *Biologically Inspired Neural Networks: Models, Learning, and Applications*. VDM Verlag, Saarbrücken, Germany, Germany, 2010. ISBN 363922826X, 9783639228267.

[5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 153–160, Cambridge, MA, USA, 2006. MIT Press. URL http://dl.acm.org/citation.cfm?id=2976456.2976476.

[6] NCBI Resource Coordinators. Database resources of the national center for biotechnology information. *Nucleic Acids Res*, 44(D1):D7–D19, Jan 2016. ISSN 1362-4962. doi: 10.1093/nar/gkv1290. URL https://www.ncbi.nlm.nih.gov/pubmed/26615191. 26615191[pmid].

[7] Marta R. Costa-jussà. From feature to paradigm: Deep learning in machine translation. *J. Artif. Int. Res.*, 61(1):947–974, January 2018. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=3241691.3241715.

[8] Yann Le Cun. A theoretical framework for back-propagation, 1988.

[9] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 766–774. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5548-discriminative-unsupervised-feature-learning-with-convolutional-neural-networks.pdf.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

[11] Daehwan Kim, Li Song, Florian P Breitwieser, and Steven L Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome research*, 26(12): 1721–1729, 2016.

[12] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Lin-

guistics. doi: 10.3115/v1/D14-1181. URL https://www.aclweb.org/anthology/D14-1181.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[14] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66722-9. URL http://dl.acm.org/citation.cfm?id=646469.691875.

[15] Byunghan Lee, Seonwoo Min, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in Bioinformatics*, 18(5):851–869, 07 2016. ISSN 1467-5463. doi: 10.1093/bib/bbw068. URL https://doi.org/10.1093/bib/bbw068.

[16] Yunan Luo, Jian Peng, Yun William Yu, Jianyang Zeng, and Bonnie Berger. Metagenomic binning through low-density hashing. *Bioinformatics*, 35(2):219–226, 07 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty611. URL https://doi.org/10.1093/bioinformatics/bty611.

[17] Romain Menegaux and Jean-Philippe Vert. Continuous embeddings of dna sequencing reads, and application to metagenomics. *bioRxiv*, 2018. doi: 10.1101/335943. URL https://www.biorxiv.org/content/early/2018/05/31/335943.

[18] Peter Menzel, Kim Lee Ng, and Anders Krogh. Fast and sensitive taxonomic classifi-

cation for metagenomics with kaiju. *Nature Communications*, 7:11257 EP –, Apr 2016. URL https://doi.org/10.1038/ncomms11257. Article.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999792.2999959.

[20] Daniel J. Nasko, Sergey Koren, Adam M. Phillippy, and Todd J. Treangen. Refseq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology*, 19(1):165, Oct 2018. ISSN 1474-760X. doi: 10.1186/s13059-018-1554-6. URL https://doi.org/10.1186/s13059-018-1554-6.

[21] Patrick Ng. dna2vec: Consistent vector representations of variable-length k-mers. *arXiv preprint arXiv:1701.06279*, 2017.

[22] Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome Biology*, 17(1):132, Jun 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0997-x. URL https://doi.org/10.1186/s13059-016-0997-x.

[23] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C.

Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[25] Rudy Setiono and Huan Liu. *Feature extraction via Neural networks*, pages 191–204. Springer US, Boston, MA, 1998. ISBN 978-1-4615-5725-8. doi: 10.1007/978-1-4615-5725-8_12. URL https://doi.org/10.1007/978-1-4615-5725-8_12.

[26] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[27] Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, Mar 2014. ISSN 1474-760X. doi: 10.1186/gb-2014-15-3-r46. URL https://doi.org/10.1186/gb-2014-15-3-r46.

[28] Michael Zwick, Sandeep Joseph, Xavier Didelot, Peter E Chen, Kimberly Bishop-Lilly, Andrew Stewart, Kristin Willner, Nichole Nolan, Shannon Lentz, Maureen Thomason, Shanmuga Sozhamannan, Alfred J Mateczun, Lei Du, and Timothy Read. Genomic characterization of the bacillus cereus sensu lato species: Backdrop to the evolution of bacillus anthracis. *Genome research*, 22:1512–24, 05 2012. doi: 10.1101/gr.134437.111.