LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Erasmus Mundus Joint Master Programme (**EMJMD+**) in Evolutionary Biology (**MEME**)

Master's Thesis

Submitted as a part of a

Master of Science

In

Ecology, Evolution and Systematics

Leveraging Deep Learning for Evolutionary History Inference in Population Genetics: A Comparative Analysis with Traditional Methods

Jaskaran Singh Gill

Start Date: **15th September 2023**
Submission Date: **23rd April 2024**
Supervisor: **Prof. Dr. Dirk Metzler**

Ludwig-Maximilians-Universität München
Faculty of Biology, Department of Biology II
Division of Evolutionary Biology

“Without that passion and urge, there is a gradual oozing out of hope and vitality, a settling down on lower levels of existence, a slow merging into non-existence. We become prisoners of the past and some part of its immobility sticks to us.”

- Pandit Jawaharlal Nehru
(Freedom Fighter and first Prime Minister of Republic of India)

„Ohne diese Leidenschaft und diesen Drang kommt es zu einem allmählichen Verschwinden von Hoffnung und Vitalität, einem Sich-Ablassen auf niedrigeren Ebenen der Existenz, einem langsamen Verschmelzen mit der Nicht-Existenz. Wir werden zu Gefangenen der Vergangenheit und ein Teil ihrer Unbeweglichkeit bleibt uns haften.“

- Pandit Jawaharlal Nehru(Freiheitskämpfer und erster Premierminister der Republik Indien)

Abstract

This study explores the application of deep learning techniques, specifically convolutional neural networks (CNNs), for distinguishing between different evolutionary scenarios in population genetics data (recent bottleneck vs recent selective sweep). Simulated data was generated using a combination of forward and coalescent simulators, capturing selection and bottleneck scenarios across varying parameters.

CNNs were trained on three data modalities: windowed site frequency spectrum (SFS), compressed SNP matrix image, and SNP matrix data. The performance of CNNs was compared against traditional machine learning approaches like Support vector machines (SVMs) trained on summary statistics such as Tajima's D, nucleotide diversity, and haplotype lengths.

The results demonstrated that CNNs trained on windowed SFS data achieved comparable accuracy to SVMs, slightly outperforming them in the selection scenario. The CNN trained on other data modalities was comparable to the one trained on summary statistics. The multimodal CNN combining SFS and SNP data did not significantly improve performance over single modality models.

The thesis also addresses key considerations for reproducible CNN models, such as setting random seeds and platform-independent implementations. Principal component analysis (PCA) was conducted on summary statistics, revealing clustering based on evolutionary scenarios, although with some instances of misclustering

Overall, this research highlights the potential of deep learning models like CNNs to effectively capture patterns in genetic data and infer underlying evolutionary processes, without relying on hand-crafted summary statistics. The CNN-based approach shows promise for robust analysis in scenarios where theoretical models are lacking, paving the way for more accurate and efficient population genomics studies.

Table of Contents

Introduction:

1. Overview of methods in Population History Inference:

Present population genetic diversity and patterns is shaped by past demographic and genomic events. These genomic footprints of our evolutionary history, enigmatic in their nature, store the history of all species on this planet. For example, recent origin of our species in Africa followed by "out of Africa" migration events found robust support in present day genomic analysis of diverse human populations(Ingman et al., 2000; Li et al., 2008; Ragsdale et al., 2023). Furthermore, a multitude of other evolutionary events such as population bottleneck, expansions, admixture and selective sweeps leave a discernible mark on our genetic landscape, enriching the rich history of our evolutionary past (MacLeod et al., 2014) and our knowledge of complex human disease(Saeb & Al-Naqeb, 2016).

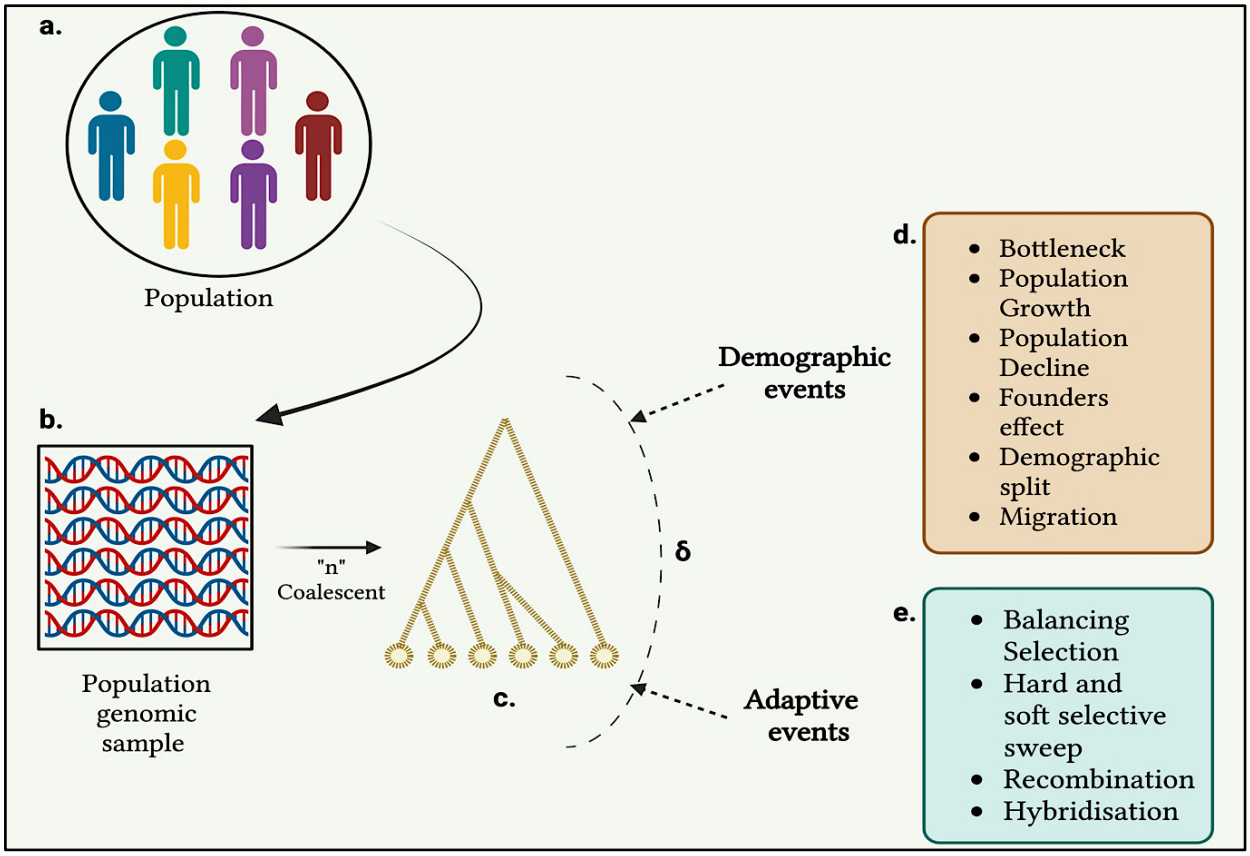

Figure 1: General Workflow in Evolutionary History Inference. Genomic samples (b) are collected from a population sample (a). The genome is then sequenced and aligned, and subsequently, the Kingman n-coalescent model is applied to generate a coalescent tree with individual tips. This coalescent/genomic tree is influenced by both demographic (d) and adaptive events (e), denoted by (δ). Potential events are inferred by comparing the coalescent tree with a null model coalescent tree.

Population history Inference methods stands as a powerful tool, offering insights into the mechanism of these evolutionary and demographic forces and their influence and

maintenance of genetic diversity(Rodrigues et al., 2023). Genetic diversity is not just a mere reflection of past; it is directly related to the adaptability of the species and serves as a raw material for selection(Jones et al., 2012; Lai et al., 2019; Reid et al., 2016). This diversity renders species ability to resist new diseases, to fight against changing environment and overall gives species better odds to survive(Agha et al., 2018; Cortés, 2017). Consequently, population history inference is pivotal for conservation genomics(Hogg, 2024; Hohenlohe et al., 2021; Theissinger et al., 2023). Population history and parameter inference also provide insights into possible speciation mechanism and give broader understanding of the speciation process(Butlin, 2010; Noor & Feder, 2006; Rutherford et al., 2018).

Decoding this evolutionary story from our genomic data posed a significant theoretical and computational challenge to the population biologists in the 20th century. Initially, genetic data consisted of only allele frequencies gathered using the technique of gel electrophoresis. For example, Lewontin and Hubby famous study of genic diversity of *Drosophila pseudoobscura* (Lewontin & Hubby, 1966) .To infer from this data, previously developed statistical techniques such as Maximum Likelihood by Ronald Fisher (Fisher, 1922) and Bayesian analysis (named after Reverend Thomas Bayes) (Stigler, 1982) were first modified for phylogenetic inference (Cavalli-Sforza & Edwards, 1967) and then for population history inference. With the advent of sequencing (Sanger et al., 1977) and increased computational power (Moore, 1965) we entered in the world of molecular sequences; these methods were again modified to suit our changing data needs by pioneers like Joseph Felsenstein and Neyman (Felsenstein, 2001; Neyman, 1971) . This collaborative endeavor, uniting genetics, statistics, sequencing technology, and computational biology has yielded numerous techniques for inferring population history. The underlying framework on which most of these techniques rely is the population model known as “n-coalescent” which was theorized and invented by JFC Kingman(Kingman, 1982). “n-coalescent” is the continuous time Markov chain process on a finite set of states, which describes the familial relationship among “n” samples drawn from a larger haploid population(Kingman, 1982). The neutral Kingman “n-coalescent” serves as a natural null model in order to detect selection, bottleneck or other evolutionary events (R. B. Harris & Jensen, 2020).

The numerous techniques used for inferring evolutionary history and population parameters can be broadly classified into three main classes, namely 1.) Likelihood-based approach, 2.) Summary statistics-based approach and 3.) Deep learning-based approach. The most commonly used approach among three is the methods relying on population genetic summary statistics: value or set of values designed to encapsulate the information derived from aligned sequence data originating from one or multiple populations(Arnab et al., 2023; Hejase et al., 2020).The summary statistics based approach can be broadly classified into two categories i.e. ABC by summary statistics and Machine learning based on summary statistics. The basic idea behind summary statistic approach is that whenever evolutionary and demographic forces acts on the population it causes a change in the structure of genealogy in that population(Willi et

al., 2020). For example, a population bottleneck would lead to faster coalescence of lineages at the point of bottleneck compared to the expectation under neutral Kingman Coalescent model(Bunnefeld et al., 2015), whereas population expansion would lead to longer leaf branches as compared to ones from neutral Kingman coalescent model (Nordborg & Krone, 2002). Change in the shape of genealogy results in the change of allele site frequency spectrum (SFS) (Hobolth & Wiuf, 2009). SFS is the distribution of allele frequencies across polymorphic sites. SFS distribution is generally summarized by population summary static known as Tajima's D, created and named after Japanese Population geneticist Fumio Tajima(Tajima, 1989). Tajima's D is generally defined as the difference between the two unbiased estimators of population scaled mutation rate, $\theta = 4 * N_e * \mu$ (Ne is Effective Population Size, μ is mutation rate) divided by the standard deviation of the difference (equation 1). Two unbiased estimators are Tajima's estimator θ_π and Watterson's estimator θ_w (Watterson, 1975). Tajima's estimator is the average number of pairwise difference whereas Watterson's estimator is number of polymorphic sites divided the total length of the coalescence tree. As Both of the estimators estimate θ under neutral model, the difference between the two is expected to be close to 0 when null hypothesis of neutral evolution hold. Tajima's D is positive in case of excess of common allele and negative when rare alleles are in excess. Tajima' D can take similar sign in case of different evolutionary scenario for example Tajima's D can take negative value in case of balancing selection or recent population decline. Ideally, we want our statistic to inform us only about the evolutionary scenario we are studying but generally in practice different evolutionary force can lead to same genealogy making it hard to differentiate between them using summary statistic-based approach. There is plethora of population summary statistics capturing different aspect of genomic sample such as squared correlation coefficient ($\mathbf{r^2}$) capturing Linkage Disequilibrium (VanLiere & Rosenberg, 2008) and Fixation index ($\mathbf{F_{ST}}$) capturing population differentiation (Nei, 1986).

$$\boldsymbol{Tajima's\ D = d/\sqrt{variance(d)}}$$
$$\boldsymbol{d = \theta^{\pi} - \theta^{s}}$$
where $\boldsymbol{\theta^{\pi}} = meanwise\ pairwise\ difference$ (Tajima's π)
$\boldsymbol{\theta^{s} = S/(\sum_{1}^{i-1} 1/i)}$ (i = # individuals in sample, S = # segregating sites)

Equation 1: Mathematical formula to calculate Tajima's D

The major concern with the statistic-based inference techniques is generalizing the large amount of information into a single value. This loss of information generally expected to decrease our resolution to differentiate between evolutionary events. For example Tajima's D can take similar sign in case of evolutionary events(Tajima, 1989). The major challenge in population genetics is to utilizae as much input data as possible. Maximum Likelihood/Bayesian based methods for population history inference does not suffer from this loss of information. Maximum Likelihood based methods were pioneered by Griffiths and Tavare (Griffiths & Tavaré, 1994) and Mary Kuhner (implemented in her software **"Lamarc"**) (Kuhner, 2006).

The Maximum Likelihood approach operates by systematically exploring the population parameter space, thereby identifying parameters that maximize the probability of our data while averaging over all the feasible genealogical structures denoted (Beerli & Felsenstein, 1999)Complementing the frequentist Maximum Likelihood methods are Bayesian approaches, which typically incorporate uniform distributions or informed priors for parameters, facilitating the derivation of posterior parameter distributions following the examination of genetic DNA samples (Nielsen, 2000).

Bayesian methodologies are acknowledged for their efficacy in navigating complex, multidimensional parameter spaces for enhanced parameter estimation (Beerli, 2006). It is important to note that both methodologies are strongly influenced by the selection of population demographic models, underscoring the critical importance of this aspect in the inference process (Ripplinger & Sullivan, 2008).

It is generally difficult to calculate precise Likelihood for large genomic dataset due to shared history of the different sites along our DNA sequence which emerges as covariance among sites making it difficult to calculate likelihood which lead to the emergence of the composite Likelihood approach/pseudo likelihood approach (Garrigan, 2009). Composite likelihood is the product of marginal likelihood acting as an approximation for true maximum likelihood (Besag, 1974). This approach is implemented in different methods including ***∂a∂i*** (Diffusion approximation for Demographic inference)(Gutenkunst et al., 2009) and **"Jaatha"** (Metzler, 2011).

Returning to summary statistics-based approaches, two main population summary statistic-based approaches include Approximate Bayesian Computation (ABC) and Machine Learning based on statistics. ABC was first introduced and applied by JK Pritchard (Pritchard et al., 1999).ABC is Bayesian based paradigm includes the simulation of data under the prior distribution of parameter and accept only those parameters which produce population statistics close to our sample within certain margin of error. The advantage of ABC includes the use of Monte Carlo simulations instead of computationally expensive likelihood computation, which makes it a preferable choice for complex model which have large number of parameters(Sadegh & Vrugt, 2014). It is robust, flexible and bypass exact likelihood calculation, which makes it quite useful. Practically, ABC implementation requires the combination of simulator for ex Hudson's **"ms"** (Hudson, 2002) , **"SIMCOAL 2.0"** (Laval & Excoffier, 2004) and ABC regression software such as **"msbayes"** (Hickerson et al., 2007) and **"serial SimCoal"**(Anderson et al., 2005).

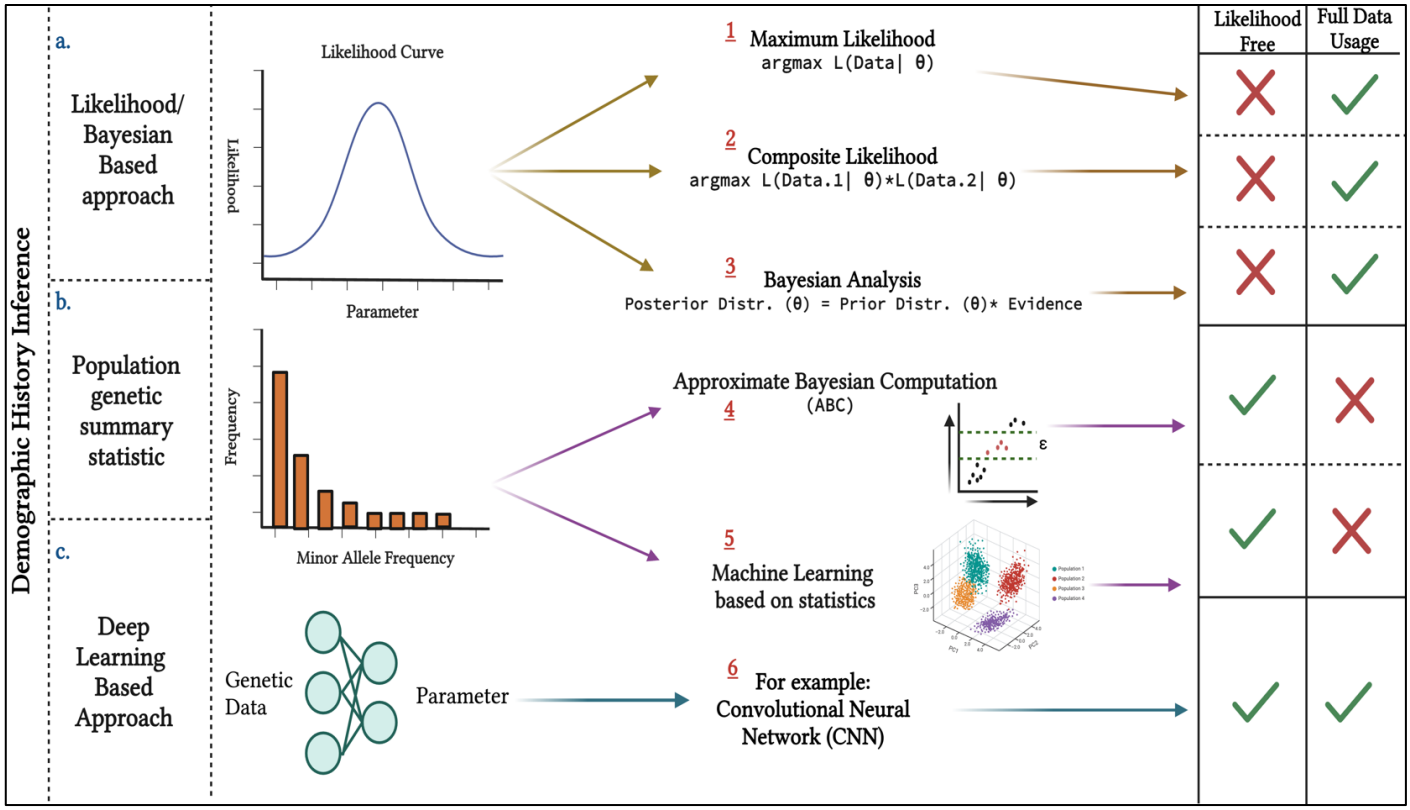

Figure 2: Classification of methods used in population history inference. Likelihood based method (a) has three subclasses i.e. Maximum Likelihood (1), Composite Likelihood (2) and Bayesian Analysis (3). The summary statistics based approach (b) has two subclasses i.e. Approximate Bayesian Computation (4) and Machine Learning based on summary statitistics. Deep learning based approach (c) include multiple architectures such as CNN. On the right, each model is characterized whether it is likelihood free and uses full input data.

Machine learning has become increasingly used in population data analysis over the last decade(Xu & Jackson, 2019). Machine learning approaches are broadly divided into two main categories: supervised and unsupervised machine learning. Unsupervised machine learning involves learning based on the features contained within the data itself, such as variance, in the absence of predefined labels or clusters of input data (Wong, 2021).Principal Component Analysis (PCA) invented by Karl Pearson (Pearson, 1901) is one of the most widely used Unsupervised Machine learning algorithm. On the other hand, supervised machine learning includes learning from labeled data points for accurate predictions for example support vector machine (SVM) (Cortes & Vapnik, 1995).

Machine learning models in population genetics are generally trained on the multitude of different population statistic each capturing different aspect of aligned population genetic data instead of single statistic-based approach, thus giving better accuracy and prediction. Machine learning methods are likelihood free which makes them generally robust, flexible and computationally inexpensive. The software S/HIC (Schrider & Kern, 2016) used to differentiate between soft and hard sweep is a good example of such an approach. Machine learning models also don't suffer from the "curse of dimensionality" of parameter space (Poggio et al., 2017) whereas methods like ABC based on

population summary statistic are sensitive to dimensionality of parameter space i.e. with increasing dimensions less and less parameters combinations are accepted. Machine learning models just trained on summary statistics still does not use full extent of data as they are trained on features extracted from data (population genetic statistics) rather than actual raw data.

Deep learning based approach, third and final class of methods, solves the problems associated with traditional machine learning meanwhile retaining the advantages of it. "Deep Learning" as a term is coined by data scientist Geoffrey Hinton ("Deep Learning," 2020). It represents a subset of machine learning where network learn directly from raw data. The theoretical roots of deep learning can be traced back to the seminal work of Alan Turing and John von Neumann (Mühlenbein, 2009). Both believed that logic does not fully capture the computational processes of our brains. The other major contributions in field of deep learning includes Geoffrey Hinton work in the realm of backpropagation which propelled deep learning into the mainstream (Lillicrap et al., 2020). Deep neural networks excel in learning patterns and extracting features directly from data, thereby bypassing the need for population summary statistics. By leveraging the entire dataset for feature extraction, deep learning maximizes the utilization of available information. Moreover, deep learning methods are less susceptible to the "curse of dimensionality" compared to other models, thanks to their inherent architecture and non-linearity. Deep learning methods possess all the desired qualities such as being likelihood-free meanwhile utilizing the entire dataset for learning purposes. Neural Networks can learn increasingly complex function just by increasing the number of nodes per layer and number of hidden layers in our neural network. Neural Network are universal function approximators (UFL) because of their ability to learn any function to any degree of precision (Hanin, 2019). Neural networks rely on a large amount of data to learn features from it, but determining how much data is "large enough" is challenging before training. It depends on both the complexity of the problem and the complexity of the neural architecture itself (Najafabadi et al., 2015).

2. Question Addressed by our study:

Low Genetic diversity among species or population is not something unique but observed quite frequently in plant and animal populations(Caspermeyer, 2019; Hagenblad et al., 2015). This reduced diversity then expected can be explained by higher inbreeding, non-panmictic mating, background selection at the physical linked loci or bottleneck(Ivana Cvijović et al., n.d.; Lozada-Soto et al., 2021; Sánchez & Woolliams, 2004; Sonsthagen et al., 2017). In numerous instances, a recent occurrence in the population's history is often cited as an explanation for decreased variability (Luikart et al., 1998). These diversity reducing events can be classified into two major category i.e. demographic events or selection events. Demographic events include bottleneck or founder events, both leads temporary population size decrease which increases the genetic drift leading to lower diversity(Sonsthagen et al., 2017). The selective events include rapid fixation events (selective sweep) at multiple loci or single

loci which decrease the diversity at physically linked to selected loci due to hitchhiking also known as background selection (Stephan, 2019). Identifying bottlenecks holds significance in conservation biology as it provides insights into mechanism of bottleneck and possible solutions to overcome negative impacts of it (Aronne, 2017). Conversely, the detection of selective sweeps is a crucial objective in the examination of evolutionary mechanisms (Chen et al., 2016). Both events have a high variance in results due to the variability in mutation, drift, and random sampling (Wein & Dagan, 2019). Therefore, we might observe quite different patterns among different loci acted upon by the same demographic or selective force. Given their biological and evolutionary significance it's very essential to accurately differentiate between the recent bottleneck and selection sweep(Galtier et al., 2000; Jensen et al., 2005). The recent methods to differentiate between the selective and demographic factors include composite likelihood method as defined in the (Kim & Stephan, 2002)

In our study, we explore the accuracy and robustness of the deep learning approach, specifically the Convolutional Neural Network (CNN) method, in differentiating between the diversity-reducing early bottleneck and selection sweep scenarios using the simulated SNP (Single Nucleotide Polymorphism) dataset. We then compare it to machine learning models such as PCA and Support Vector Machine (SVM) trained on summary statistics.

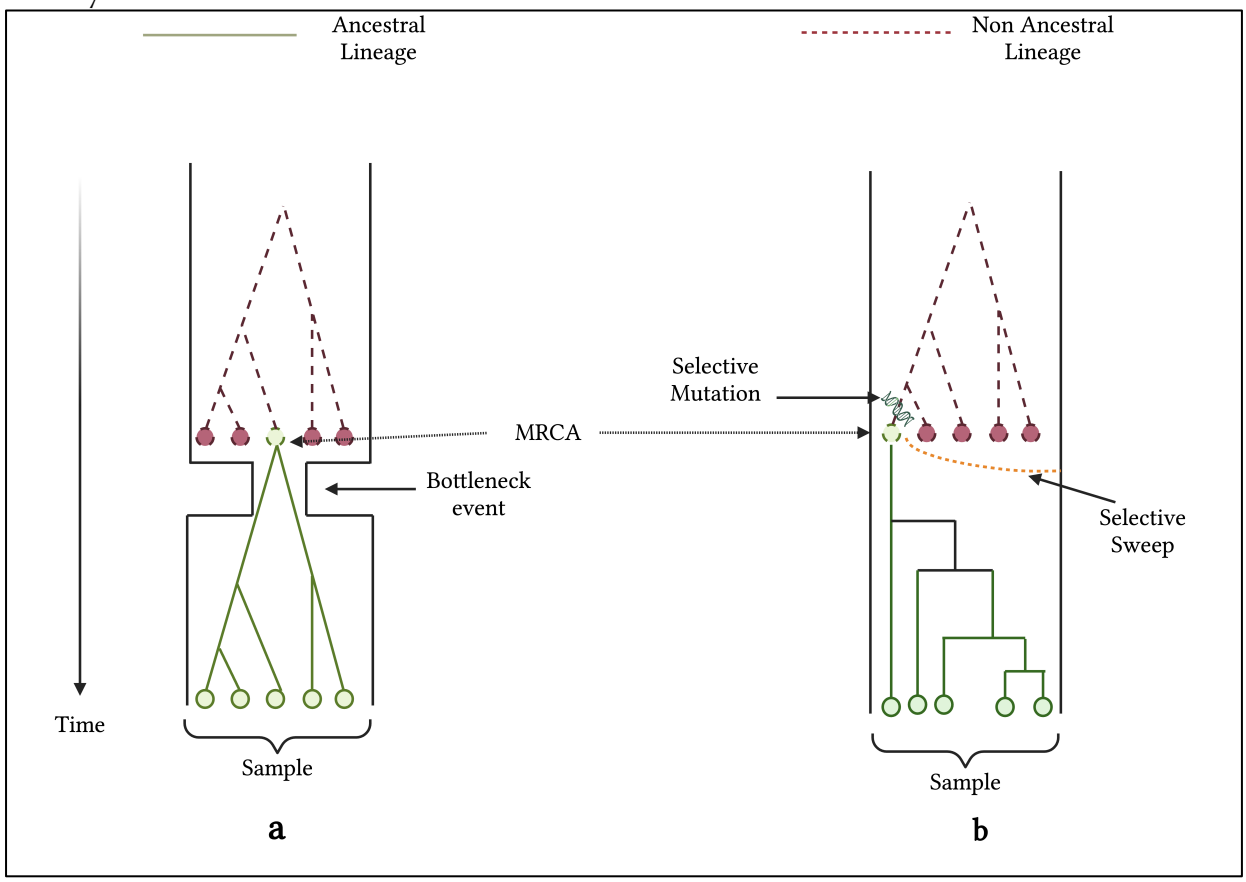

Figure 3: The genealogy from strong bottleneck event is shown on the left side (a) whereas the genealogy from the hard selective is shown on the right side (b). Both genealogies have similar structure showing the possibility of both the scenario lead to same genealogy and therefore same summary statistics

Convolutional Neural Networks are inspired by the 1960 work of David Hubel and Torsten Wiesel on the visual cortex of the cat (Hubel & Wiesel, 1959) where they found different cells responded to different local patterns of stimulation. Each layer of cells was acting as a filter for local pattern which combines to form complex pattern. These results inspired computer vision scientist which led to Neural Network containing convolutional filters pioneered by Yann LeCun (Lecun et al., 1998) . The detailed introduction to CNN and neural network is provided in the next section.

We simulated genomic data evolving under bottleneck or selection scenario using the combination of forward and backward simulator. Then we trained our CNNs to distinguish between two events within a simulated population evolving under the Wright-Fisher demographic model, utilizing windowed SNP data. The windowed SNP analysis involved aligning sequences and tallying the mutations within each window for individual samples. These mutation counts were stored in a matrix column, and the window was shifted by a specified step size, iterating until the sequence's end. CNN was trained on the resultant matrix. We also trained the CNN on the compressed greyscale image of the same matrix but 2-5 times lower in size. A CNN was also trained on the Windowed Site frequency spectrum (SFS). Finally, we also trained a multimodal CNN om windowed site frequency spectrum and compressed greyscale images of SNP matrix.

Additionally, we explored classical machine learning techniques, including Principal Component Analysis (PCA) and Support Vector Machines (SVM), as part of our analysis. PCA, unsupervised linear dimensionality reduction technique, was applied to summary statistics such as Tajima's D (Tajima, 1989), whole genome site frequency spectrum, number of mutations, Tajima's pi, and average haplotype length. This technique aimed to assess whether the summary statistics clustered into distinct groups based on evolutionary history. Furthermore, we employed Support Vector Machines (SVM), a supervised learning algorithm used for classification by maximizing the margin between two classes, on summary statistic data. We rigorously evaluated the accuracy and performance of both PCA and SVM models and compared them against the CNN-based approaches.

We demonstrated the CNN's capability to distinguish between the two scenarios. The accuracy of CNNs trained on windowed site frequency spectrum was found to be slightly higher compared to CNNs trained on grayscale images of windowed SNP data. The CNN trained on compressed greyscale image found to have approximately same accuracy as the one trained on the uncompressed SNP matrix. However, overall, the accuracy of CNNs was comparable to the statistics-based PCA and SVM approaches. This highlights the remarkable ability of the model-free CNN-based approach in inferring population history. The multimodal CNN did not perform better than single modality CNN hinting that possible combination of data structure would not accuracy especially if the additional data is just summary statistic of the current data. A CNN-based approach can also be applied to systems where theoretical models for inference are lacking, showcasing their remarkable flexibility.

We are still at the early stages of the deep learning era in population genomics. With increased capabilities and improved architectures, it is foreseeable that deep learning based would eventually surpass statistics-based approaches in terms of accuracy and robustness.

3. Overview of Neural Networks and Convolutional Neural Network (CNN):

The cornerstone of deep learning is Artificial neural networks (ANN). ANN contains multiple layers of artificial neurons which act as mathematical function. Each artificial neuron can take an input and convert it into an output value (Ansari, 2023). Fully connected Artificial layer contain one or more hidden layer which takes input from input layer and convert it to the prediction in the output layer. The output of an $\boldsymbol{y^{th}}$ neuron in a single layer neural network is describe by the following equation:

$$a(\sum_{i}^{n} w_{iy}x_i + b_y)$$

where $\boldsymbol{x_i}$ is the input to the neuron $\boldsymbol{y}$ either from the previous hidden layer or directly from the input if it is the first hidden layer. Weights in a neural network can be understood as the connections between the input and the neuron. The higher the weight, the more the impact input has on that neuron. Here, w_{iy} represents the weight between the input $\boldsymbol{x_i}$ and the neuron $\boldsymbol{y}$. The bias term ($\boldsymbol{b_y}$)is added to the product of the input and weight; it acts as a fine-tuning parameter, allowing the network to fit the data better. The variable $\boldsymbol{a}$ denotes the activation function, which is generally a nonlinear function. This characteristic enables the neural network to learn nonlinear function, which is typically the case in most situations. The figure 4a shows the input and output in a hypothetical ANN (Ünal & Başçiftçi, 2022)

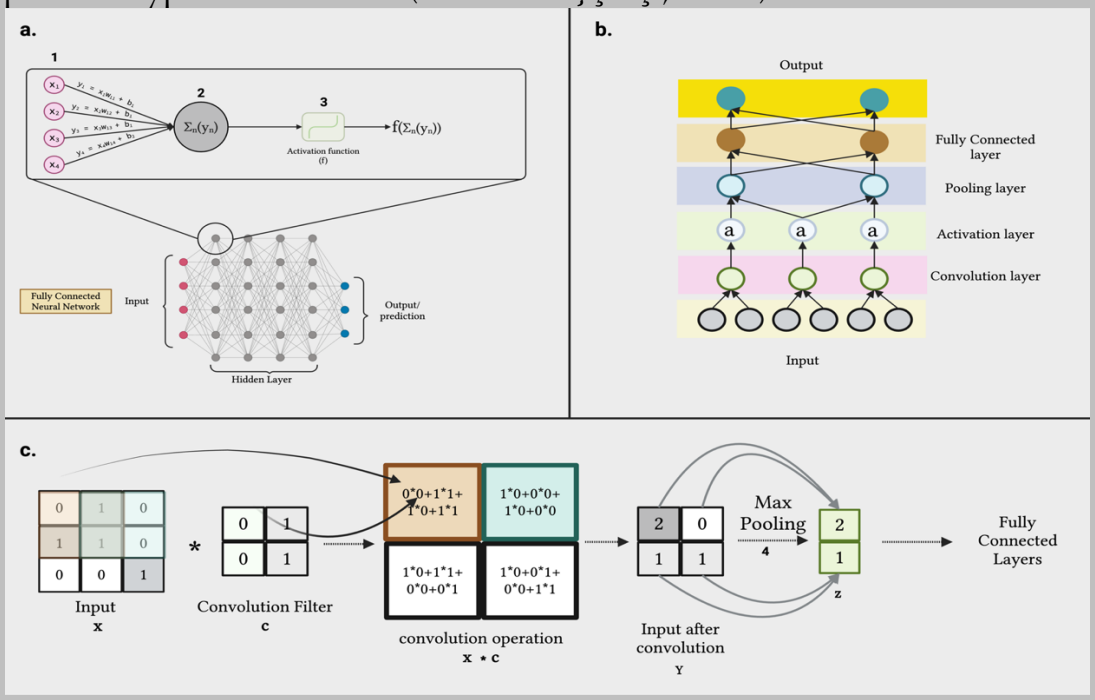

Figure 4: a. Figure 4: a.) Fully Connected Artificial Neural Network (ANN) – an artificial neuron with artificial neural network is depicted, consisting of four input nodes (1), each with individual weights

but the same bias. The inputs are summed up after multiplication with their respective weights and the shared bias (2). This aggregated output is then passed through an Activation function (3), aiding in learning nonlinearity in the data. b.) Illustrates the typical order of different layers in a convolutional neural network. c.) Demonstrates the convolution operation (x * c) on a 3x3 matrix (x) using a 2x2 kernel (c), resulting in an output (y). The output from the convolution (y) is then passed through a max-pooling layer with a kernel size of 1x2, producing a summarized output (z), which is subsequently fed into a fully connected layer after flattening.

The weights and biases of a neural network are iteratively optimized through the training process, employing techniques such as stochastic gradient descent (Newton et al., 2018) and backpropagation (Rojas, 1996) within a multidimensional parameter space. Loss function also known as error or cost function quantifies the difference between the actual and predicted output, it is also critical for optimization. Stochastic Gradient descent (SGD) involves updating the parameters in the opposite direction of the gradient of loss function to minimize the loss (Newton et al., 2018). Backpropogation described simply is the backward propogation of the loss function into the network using the chain rule of calculus, which helps network to optimiz parameter according to SGD (Rojas, 1996).

Convolutional Neural Networks is a class of artificial neural network which excel in computer vision and significantly used in population genetics. The CNN are known to be affective in multitude of classification problem example "AlexNet"(Krizhevsky et al., 2012). CNN first learn lower-level features in the image such as edges. These lower-level features are combined to form high-level feature such as ear, nose or eye, Thus CNN can learn the hierarchy in our dataset thank to its architecture (Aurélien Géron, 2019) . The CNN derives it names from the convolution layers deployed before the fully connected layer in CNN (figure 4b). In a convolutional layer, a filter consisting of a group of kernels gathers data within a localized section of the input using a mathematical process known as convolution (Mathematical expression in Equation 2). Convolution involves overlaying one function typically filter/kernel over another typically image or signal (Equation 2). As the Kernel slides over the input signal, it computes the weighted sum of overlapping values generating a feature map (Figure 4c). Each filter produces a feature map which captures certain unique pattern from the image. Convolution filters can be classified into two types: one-dimensional (1D) or two-dimensional (2D), based on the shape of the filter. 1D convolutions are typically used for processing time series data. They consist of rectangular matrices (**Matrix M (m, n) with m rows and n columns, where all M_{ij} are initialized with independent random values at beginning and learned during training**) with one dimension spanning the same range as either dimension of the input. 1D convolutional operations transform 2D input data into a single-dimensional vector. On the other hand, 2D convolutions are commonly applied to image data. They involve rectangular or square matrices (**Matrix M (m, n), m & n < dimension of input image, if m ≠ n -> rectangular and if m = n -> square, where all M_{ij} are initialized with independent random values at beginning and learned during training)** with dimensions that are smaller than those of the input image.

Convolution with 2D image H and 2D kernel F

$$G = H * F$$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

- G[i,j] represent the value of resulting image at position (i,j) after convolution
- H[u,v] represent the value of pixel at position (u,v) in image
- F [i-u, j-v] represent value of kernel at relative position after flipping the kernel.

Equation 2: Mathematical expression representing convolution of two functions.

Parameters of the kernel/filter are learned during the training process using algorithm such as backpropagation and stochastic gradient descent (described before in this section). Convolution layers are generally followed by nonlinear activation layer and Pooling layer. Activation layers introduces nonlinearity also known as detector layer (as it informs whether the pattern is observed or not), whereas Pooling involves a condensation process, where the larger images are summarized into small image. Pooling function slides over the image, summarizing the information in that window of data. There are different kind of pooling function such as Max Pooling, Average Pooling. Max Pooling selects the maximum data value in a particular data window to summarize it whereas Average Pooling takes the average of all the values in that window to summarize it. Pooling layers provides some translational invariance to our network i.e. Patterns can be observed regardless their position in our original data set (Mouton et al., 2020). A typical CNN generally contain series of convolutions and pooling layers.

2D Max Pooling Operation on input x over fixed pooling window of shape (f, f)

$$y[i,j] = max_{(a,b) \in pooling\ window(f,f)} x[i \cdot s + a, j \cdot s + b]$$

x: Input matrix,
y: Output matrix,
(f, f): size of rectangular pooling window
s: Stride (Amount by which pooling window move after each operation)
(i, j): Indices of the output matrix
(a, b): indices of input within pooling window

Equation 3: Mathematical expression representing 2D Max Pooling

Methods:

1. Code Availability:

The code used in this section, including that for simulation, CNN training, and SVM training, is openly hosted on the GitHub page (vicegill/Master_thesis) available at ***https://github.com/vicegill/Master_thesis***. The repository includes the training notebooks and the saved parameters for different models. Additionally, the model accuracy and loss data for different models and parameters are saved in the form of an Excel sheet saved within the repository itself. The supplementary figures are also present in the repository in the form of pdf. Any other data related to the study can be obtained by emailing ***Jaskaran.gill@evobio.eu***.

The methodology can be segregated into two distinct phases:

2. Phase 1: Data Generation

The generation phase focuses on generating simulated data, which serves as the foundation for subsequent analysis.The "snakemake" (`version 8.5.3`) (Mölder et al., 2021)is used to execute the simulation pipeline.

Demographic model:

In case of bottleneck situation (Figure 4a), we have multiple parameters in the demographic model such as population size before bottleneck (N_1), Generation at which bottleneck happen (T_B) **population size during bottleneck (N_B), bottleneck intensity (I), duration of bottleneck (d_B), population size after the bottleneck (N_2). Conversely**, In the case of selection (Figure 4b) we have parameter such as population size (N), selection coefficient of selective mutation (s), Dominance coefficient of selective mutant (h) and Generation at which selection happen (T_s).

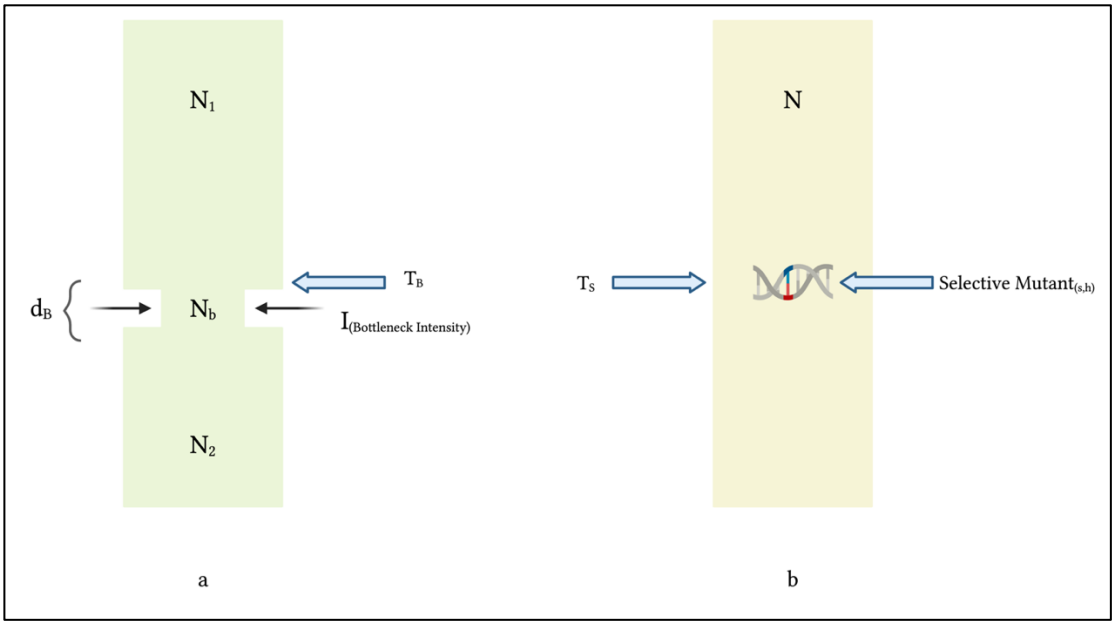

Figure 5: a) Demographic model used to simulate the Bottleneck Scenario: N_1 is initial population size before bottleneck, N_b is size during bottleneck, N_2 is population size after recovery from bottleneck, dB is duration of bottleneck, I refer to bottleneck Intensity and T_B denotes time of bottleneck. b) Demographic model used for selective sweep simulation: N is the population size, T_S is the time of selection and Selective mutant (s, h) refers to selective mutation with selection coefficient s and dominance coefficient h.

We simulated the panmictic population of 50000 diploid individuals evolving under the Wright-Fisher Model (software implementation detail mention in following section). Each of these diploid individuals contains two 100 kilo base strands of DNA. The evolutionary forces acting on this hypothetical organism's genome includes the mutation rate of $1e-08\ per\ bp\ per gen$ and recombination rate in Selection Scenario of $3e-09\ per\ bp\ per\ gen$ (one third of mutation rate).

Parameters	**Value**
Population Size (N_e)	*50000* diploid Individuals
Length of genomic element	*100* Kilo Bases
Mutation Rate	*1e-08* per base pair per generation
Recombination Rate in Selection Scenario	*3e-09* per base pair per generation
Recombination Rate in Bottleneck Scenario	*1e-09* per base pair per generation
Population model	Wright – Fisher Model
Time of Bottleneck and Selection occurrence	*5000* generation before
Bottleneck Intensity	Uniform (0.001,1.0)
Selection coefficient (s)	Uniform (0.001,1.0)
Bottleneck population size	*(1-bottleneck intensity) *50000*
Dominance coefficient of selective mutation (h)	*0.5*
Selective Mutant position	Uniform (1,100000)
Number of sampled Individuals	100 diploid Individuals (200 genomic elements)

Table 1: Parameters and their distribution or values used for the simulations of bottleneck and selection scenario using SLiM and PySLiM.

All the parameters for simulation are stored in the YAML file (*YAML Ain't Markup Language*)(`version 1.2`) (Ben-Kiki & Evans, 2009) each for bottleneck and simulation and can be updated using the graphical interface app (by running command python workflow/app_selection.py or python workflow/app_bottleneck.py on kernel). These apps are created using the python flask library (version 3.0.1) (Grinberg, M, 2018) and JavaScript (Mozilla Developer Network,2023).

In the case of selection, the selective mutant is introduced 5000 generation before in the population in the single individual at the single chromosome. The selective mutant has the selection coefficient ranging between $0.001 - 1$ and have fixed dominance coefficient of 0.5. In the case of bottleneck, the population size decreases to $(1 - bottleneck\ intensity) * 50000$ diploid individuals. The bottleneck intensities range from 0.001 to 0.999.

Before simulating the genomic data, we generated the selection coefficient and bottleneck coefficients. It is done using the "random" module of NumPy library of python.(**version 1.24.3**)(C. R. Harris et al., 2020). The coefficients are sampled from the uniform distribution between 0.001-1.0. It was ensured that all the coefficients are unique, we sampled the coefficients until we had 5000 unique coefficients. We made the "snakemake" pipeline for the same. The coefficients are saved in the meta-file.

Selective Sweep Simulation:

Forward Simulator

For the simulation of selected scenario, we used both the forward simulator SLiM (**version 4.1**) (Haller & Messer, 2023) and the coalescent simulator PySLiM (**version 1.0.4**)4/23/24 3:39:00 PMand tskit (**version 0.5.6**). The forward simulator (SLiM) is used during the last 5000 generations when we introduce the selective mutant. The fitness of the selected individual (with the selective mutant) is (1 + h*s) where "h" and "s" are the dominance coefficient and selection coefficient respectively. The fitness of the rest of the Individuals is 1. The selective mutant is introduced at the random position (x_{sweep}) along the genome.

A SLiM template written in Eidos Language (Benjamin C. Haller, 2016) was populated with the recombination rate, selection coefficient, dominance and selection coefficient contained in the YAML file. The number of generations SLiM run was also updated from the YAML file.

It was ensured that the selective mutant get fixed in the population. In every generation we check the count of selective mutation (m2) was checked. If m2 count is zero it's either fixed in the population (substitution) or lost, SLiM keep tracks of substitutions. If the selective mutant is lost, we reset the generation by resetting the tick (in SLiM, 1 tick is equal to 1 generation under the wright fisher model) and reverted the genetic data from that time. The neutral mutation (m1) rate is set to the zero. The neutral mutations are later overlayed on the genealogical tree obtained after the coalescence-burnin of selection data. All the above used notation is mentioned in SLiM manual (Benjamin C. Haller & Philipp W. Messer, 2016).

The output at the end of the 5000 generation was saved in the "ts" (tree-sequence) format (Kelleher et al., 2018). It was essential to save the output in the ts format as "pyslim" (coalescence simulator) only work with the tree-sequences.

Coalescence Simulator

The population might not have reached the demographic equilibrium (coalesced entirely) during the small number of generations during which forward simulator runs compared to the effective population size of the generation, which leads us to use the coalescence simulator for the complete coalescence and demographic equilibrium of population. Coalescence simulator was used because they are faster compared to forward simulator as they track only the genealogy of sample in contrast of the entire population.

The tree-sequences are read using the PySLiM library(Haller et al., 2019) of python. PySLiM is used to perform the coalescence burnin on the SLiM data. The pyslim implementation to obtain fully coalesced tree contain three important steps namely 1. Recapitation 2. Simplification and 3. Adding Neutral Mutations. Recapitation involves providing the "prior history" to the first few initial generation of simulation, it reduced the number of total roots to one (MRCA – most

recent common ancestor). The genealogical tree of whole current population is simplified in the simplification step by selecting the sample from current population and tracking only their genealogy from the population. The genealogical tree obtained is ancestral recombination graph (ARG). We sampled 100 diploid individuals (200 genomic elements). The genealogical of the sampled individuals are later overlayed with the neutral mutations specified by the mutation rate in YAML file. We obtained the genetic sequences from the tree sequence and saved it in the FASTA (Fast Adaptive Shrinkage/ Thresholding algorithm) format (Goldstein et al., 2016).

Statistics:

The "tskit" library (Kelleher et al., 2018) also provide some inbuilt function to calculate some statistics from the simulated genomic elements. Four statistics were calculated from the sampled genomes. The Tajima's D, Tajima's pi, number of mutations, Side frequency spectrum (SFS). These statistics are not windowed and calculated for the whole genomic element in our sample. The overall
SFS (Side Frequency spectrum) was divided into eight SFS classes based on the frequency of mutation in the population.

SFS class	The frequency of Mutation
1	**1**
2	**2**
3	**3 to 4**
4	**5 to 7**
5	**8 to 9**
6	**10 to 19**
7	**20 to 49**
8	**50 to 75**
9	**76 to 100**

Table 2: Different classes of site frequency spectrum and the allele frequencies that include in the class.

These statistics are saved in a tab file (Tab separated files). The windowed SFS (Side Frequency Spectrum) was also calculated with the window size of `1000` bases, in total `100` windows for the genomic element of `100` kilo bases were calculated and saved it in the form of the NumPy array (version 1.24.3) (C. R. Harris et al., 2020).

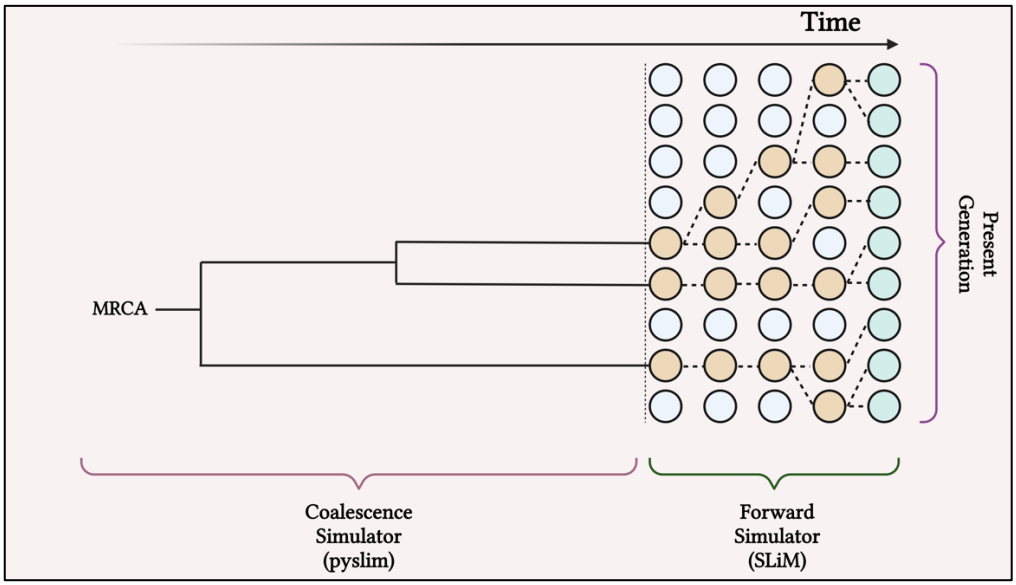

Figure 6: The diagrammatic representation shows how the genealogical tree are generated by the efficient use of forward and backward simulator thereby maintaining the complexity of demographic and selective event.

Bottleneck Scenario Simulation:

To ensure consistency in data simulation across both scenarios, a similar simulation process was followed in the bottleneck Scenario. Both forward and coalescence simulator were utilized. The intensity of bottleneck depends on the bottleneck intensity(`i`)
The population size decreases to `(1 - i) * 50000` diploid individuals around `5000` generation before. The population remains in bottleneck size for `2000` generations before recovering back to `50000` diploid individuals at `3000` generations before.

The SLiM template was used to simulate last 5000 generation, which was populated with bottleneck intensity and recombination rate parameters from YAML file during the snakemake run. The tree-sequences generated by the SLiM were read using pyslim library of python. The number of roots from the tree-sequences file is decreased to one (Most Recent Common Ancestry) using the recaptation feature of pyslim. The recapitated tree is simplified by sampling 50 diploid individuals from the present generation and keeping the nodes connected to only these 50 diploid individuals in our genealogical tree. The genealogical tree obtained in the end is overlayed with the neutral mutations using SLiM mutation model which is a modified infinite alleles model (Haller & Messer, 2019)

The nucleotide sequence generated from this genealogical tree are saved in FASTA format.
. Four statistics i.e. Tajima's' D, Tajima pi, number of mutation and SFS, for the whole genomic element
was also calculated like in the case of selection scenario. The windowed SFS was also calculated with the window size of `1000` and the result saved in NumPy array format.

SNP Matrix:

The FASTA files generated from the bottleneck and selection scenario were read and aligned using the "Bio"(`version 1.6.2`) (Cock et al., 2009) library of python. Following Alignment, the sequences were analyzed to find most common base at each position under the assumption

that the predominant base would be ancestral. This hypothetical ancestral sequence inferred under predominant base assumption was saved in other "FASTA" file. There is a possibility that most common base might not be ancestral one.

To conduct a windowed analysis, a window size of **200** base positions and step size of **100** bp for SNP'S was used to identify the SNP count in that window. It works by dividing the DNA sequence into smaller chunks (windows) and comparing them to an ancestral DNA sequence. Any differences between the ancestral sequence and the individual organism's sequence within a window are considered mutations. The count of such mutations was noted in a matrix form. The windowed analysis resulted in the matrix of shape **100*1000** (number of genomic elements* number of windows in **100kb**). The matrices are saved for training CNN on uncompressed SNP matrices.

Then The matrices are turned into the greyscale images using the "imshow" function within the "Matplotlib" (`version 3.8.1`) (Hunter, 2007) library of the python. The resulting image was saved in the PNG (Portable Network Graphics) format. During training these images are compressed into 200*200 pixel image using "resize" method for Image object within "pillow" library (version 8.3.1) (Jeffrey A. Clark, 2024)

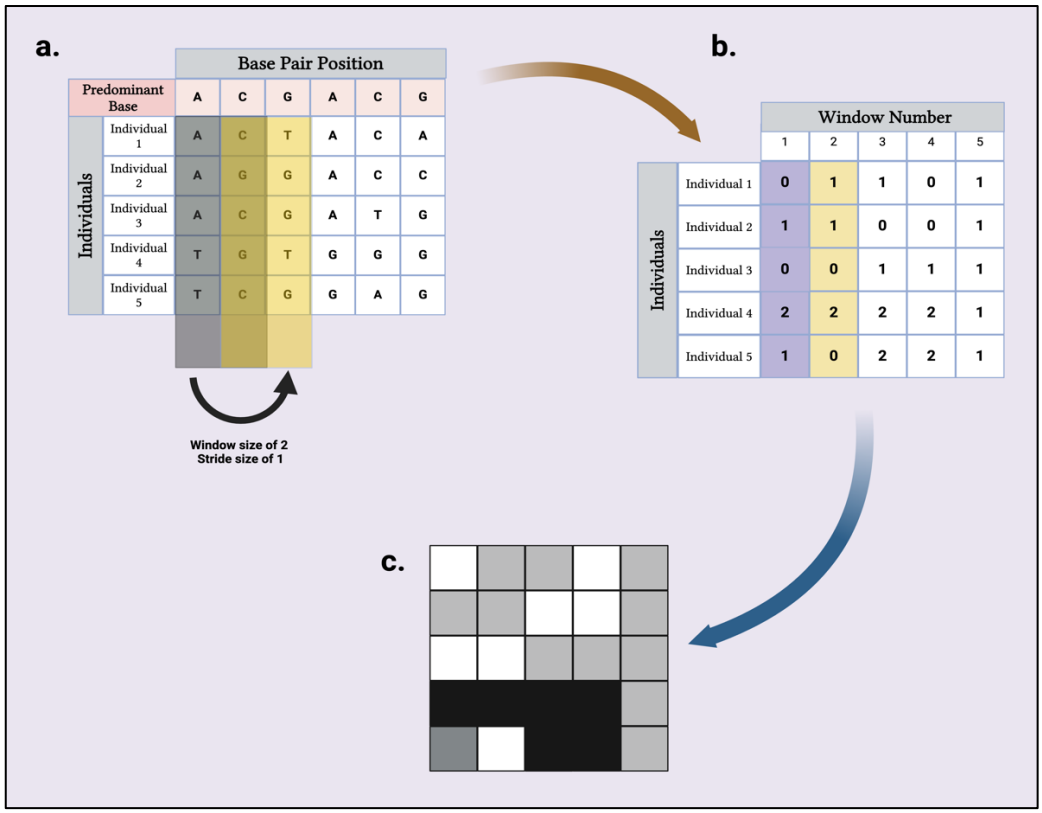

Figure 7: Process of Creating Grayscale Image from Windowed SNP Data. a.) This process employs a kernel with a window size of 2 and a stride size of 1. It involves calculating the number of mutations (relative to the predominant base) across individuals within the window. b.) The output derived from this operation is represented in (b). c.) The output obtained in (b) is then converted into a grayscale image. In this conversion, higher intensities of black correspond to a greater number of mutations.

Haplotype Scans:

The average haplotype length is defined as the average length of pairwise homozygosity(Ferdosi et al., 2016). Specifically, for a given genomic position (x), it quantifies the average distance between the first left and right polymorphic sites relative to that position. This metric was computed for every polymorphic site using H-scan (version 1.3) (Schlamp et al., 2016) and the average value across all polymorphic sites was incorporated into the metadata file of statistics. Furthermore, average haplotype length offers additional insights into demographic history that are not captured by other metrics such as the Site Frequency Spectrum (SFS).

Computational Implementation of Simulation process:

The simulation process described was executed using the Snakemake pipeline, which comprised four major rules (Figure 8). Firstly, "**Rule SLiM**" was employed for generating tree sequences via SLiM, a framework for forward-time population genetics simulations. Secondly, "**Rule pyslim**" simulated a fully coalesced population and facilitated the generation of FASTA files. Additionally, the majority of statistical calculations were performed within this rule using the built-in functions of the pyslim library. Thirdly, "**Rule SNP**" was responsible for conducting windowed SNP analysis and subsequently converting the results into PNG images. Finally, "**Rule H-scan**" computed the average H-statistic across all positions in the population.

The "snakemake" pipeline was deployed on the BioHPC cluster (`IP address: 129.187.20.102`), an HPC (High Performance Computing) nodes offered by the LRZ (Leibniz Supercomputing Center). Utilizing the cluster's resources, the entire pipeline was parallelized to enhance efficiency. To address challenges such as server overload and account quotas, the pipeline executed in the batches of 50 simulations running concurrently in a single run. Each simulation varied in terms of selection coefficients or bottleneck intensity. On average, each batch took approximately one hour to complete all the steps before producing results. The outputs are transferred to the GPU cluster hosted at the LMU Bio campus.

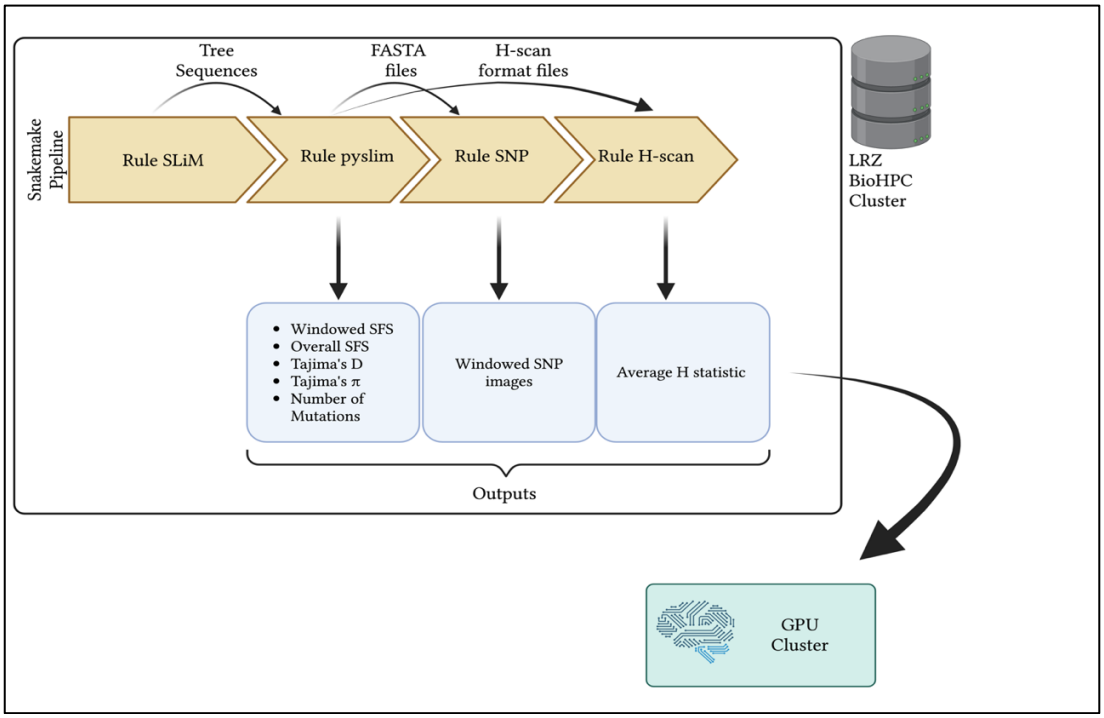

Figure 8: Schematic Representation of the Main Workflow of the Snakemake. Rule SLiM: This rule generates tree sequences using a forward simulator, which are then passed into the next rule. Rule

PySLiM: In this step, coalescence burn-in is performed and important statistics are calculated. Additionally, FASTA files are generated. The output from this rule feeds into the subsequent rule. Rule SNP: This rule evaluates the SNP matrix and produces a greyscale SNP matrix image. Rule H-scan: Finally, this rule evaluates the average haplotype length from the H-scan format obtained in Rule PySLiM. Output from each rule is transferred to GPU cluster.

3. Phase 2: Training Convolutional Neural Network (CNN)

The CNN (convolutional neural network) were used to differentiate between the bottleneck and the selection scenario.

Computational Environment for CNN training's

We developed the convolutional neural network using the open-source deep learning framework PyTorch (version 1.10.2) (Paszke et al., 2017), developed by the Facebook AI Research lab (FAIR). We opted for PyTorch over TensorFlow (Martín Abadi et al., 2015) due to its Pythonic API (Application Programming Interface) and dynamic computation graphs, which make implementation and debugging more straightforward. We utilized the "torch.nn" module, analogous to "Keras," to construct neural networks based on predefined fully connected layers, convolutional layers, pooling layers, and activation functions. The CUDA (Compute Unified Device Architecture) API (Nickolls et al., 2008) was employed for parallel computing, enabling us to harness the computational power of NVIDIA GPUs.

The networks were trained using a GPU server (**IP address: 10.163.69.14**) consisting of four **NVIDIA A100-SXM4 GPUs**, each with a capacity of **80 gigabytes (GB).** Training times varied across different networks, with a mean of approximately **10** minutes per epoch.

Different Data Modalities and Validation Strategy:

The data generated was categorized into three distinct modalities: SNP image Data, Matrix Data (which served as the basis for generating SNP images using the "imshow" function of the Matplotlib library (Hunter, 2007)), and windowed SFS data (with a window size of 1000 bases). Each of these data modalities was stored in separate folders, each containing two subfolders: "bottleneck" and "selection," housing the data relevant to those respective scenarios (see Code Availability section).

We developed custom classes tailored to handle each of these data modalities. These classes accept the root folder as input and compile the data contained within, organizing each dataset into lists along with corresponding binary label denoting their association with either the selection (1) or bottleneck scenario (0). Additionally, these custom classes store the bottleneck coefficient and selection coefficient values associated with each dataset. Each of these custom classes contain special methods such as "__get_item__" and "__len__" which allows it to be iteratively accessed during training and testing.

We segmented each of these datasets into two distinct categories: a training dataset and a validation dataset. The division of data was executed utilizing the "torch.util.data.random_split" function available within the torch library. Our approach involved splitting the data into 80 percent for training purposes and 20 percent for validation, a widely accepted practice in the

machine learning literature.To ensure reproducibility and consistency in our data splitting process, we fixed the "torch generator," akin to setting a random seed. This deliberate action was taken to obtain reproducible results in the data split, allowing for consistent evaluation and comparison of models. The network was trained on the training dataset. The testing dataset was used to evaluate the performance metrices (such as F1 score, accuracy, precision and recall) of the neural network. It should be noted that test data was previously unseen by the model and not used for training.

Data processing and loading:

We converted our data into pytorch tensors as pytorch works exclusively on them. This ensures compatibility with pytorch, enabling automatic differentiation feature (automatic gradient computation during the forwards and backward pass, gradients are crucial during learning process) and GPU acceleration, as pytorch array can be effortlessly transferred to the GPU.

The compressed SNP images were transformed into NumPy (C. R. Harris et al., 2020) arrays using the “numpy.array()” function. Finally, to prepare the data for consumption by PyTorch, we converted these NumPy arrays into PyTorch tensors using the “ToTensor()” function available within the torchvision module of PyTorch. To ensure consistent scaling of the features and improve the convergence of optimization algorithms during model training, the PyTorch tensors were further standardized normalized within the range of [-1,1]. The normalization process was automated within the custom data classes mentioned earlier, utilizing the "normalize()" function provided by the torchvision module. In addition to the compressed SNP image data, the SNP matrix data (which served as the precursor to compressed images) and the Windowed SFS array data underwent standard normalization as well. Within their respective custom data classes, these datasets were also standardized and converted into PyTorch tensors.

To efficiently load and iterate over our dataset we used “DataLoaders” function of “torch.utils.data” module with the setting that data is reshuffled at every epoch (option shuffle = True). During training, we opted for batch sizes that were multiples of 8. This choice of batch size is often determined empirically depending on factors such as the size of the dataset, the complexity of the model, and the available computational resources (Radiuk, 2017).

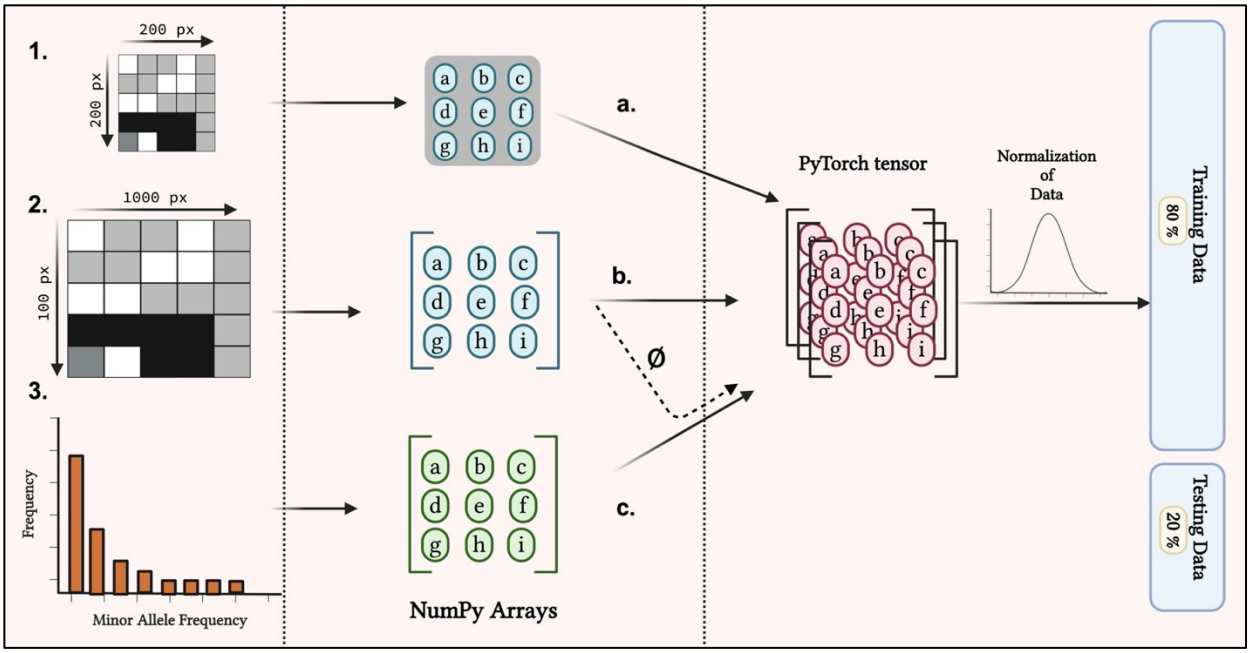

Figure 9: Data Processing before Training: 1.) The compressed grayscale SNP image (200x200 pixels) is transformed into a PyTorch tensor (a). 2.) The SNP matrix (100x1000) is transformed into a PyTorch tensor (b). 3.) The windowed Allele Site Frequency Spectrum is transformed into a PyTorch tensor. In the case of a multimodal CNN, the windowed allele frequency spectrum is combined with the SNP matrix, represented as (ϕ). Each PyTorch tensor obtained is normalized and divided into an 80:20 ratio for training and testing.

Architecture and training of Convolutional Neural Network (CNNs):

To design the structure of our neural network, we crafted a custom class called "CNN" inherited from the base class "`torch.nn`". This class outlines the architecture of our convolutional neural network and dictates how our data flows through it, accomplished through the "`forward`" method. We initialized our convolutional layers using the "`nn.Conv2d`" function. To ensure that the output of convolution preserves similar dimensions as the input, we applied padding using the "padding" option. Moreover, we controlled how much the filter moves at each step with the "stride" option, and we determined the shape of the filter using the "`kernel_size`" option. Additionally, we specified the number of desired feature maps using the "`out_channels`" option(Crowley, 2023).

The output of the convolutional layer is further processed by passing it through the Rectified Linear Unit (ReLU) activation function, which essentially outputs the input if it's greater than 0; otherwise, it returns 0 (Agarap, 2019). After this activation, the resulting output is then fed into pooling layers. Pooling can be performed using either Max pooling or Average pooling methods(Gholamalinezhad & Khosravi, 2020). The specific kernel size for the pooling layer is determined based on the model's performance, essentially fine-tuning it for optimal results.

To mitigate overfitting of the training data, we utilized the dropout method as a regularization technique (Poggio et al., 1987). Dropout randomly deactivates neurons and their connections by setting some data points to zero based on a specified probability (Baldi & Sadowski, 2013). Following a sequence of convolutional and pooling operations, the resulting output is flattened (using “nn.flatten") before being passed into the fully connected layer initialized by "nn.Linear". We conducted experiments with various sets of parameters such as the number of convolutional and fully connected layers, kernel size for convolution, number of filters in each convolutional layer, and kernel size for the pooling function to achieve optimal accuracy and performance.

During training, we set the seed for both "CUDA" and "CPU" operations to achieve reproducible training results. Seeding ensures that the initialization parameters within the model remain consistent, which ultimately leads to similar trained parameters across different runs.

During the initial training run, we conducted training for **`20`** epochs and monitored the epoch number at which both loss and accuracy began to decline, signaling overfitting. In subsequent runs, we limited training to this specific epoch. For our loss function, we utilized cross-entropy loss (nn.CrossEntropyLoss) (Mao et al., 2023), tailored for classification problems. Moreover, we employed the "Adam" optimization algorithm (Kingma & Ba, 2017) to update parameters post-loss calculation. Adam stands as an extended stochastic gradient descent method, adept at maintaining distinct learning rates for each parameter, in contrast to classical stochastic gradient descent methods. For reproducibility we saver our trained model parameters, we saved them in the form of a dictionary using "torch.save".

4. *Principal Component Analysis*:

Principal Component Analysis (PCA) was conducted using summary statistics. We utilized site frequency spectrum classes outlined in a table, along with Tajima's D, Tajima's pi, number of mutations, and average haplotype length as features for our PCA analysis. The analysis was executed using the "prcomp()" function from the "stats" (version 4.3.0) (Team, 2014) library. To ensure proper normalization, the variables were zero-centered by setting the option "center" to true.

5. *Support Vector Machines*:

A Support Vector Machine (SVM) was employed to classify between bottleneck and selection scenarios. The SVM utilized the same features as those used in PCA, and we applied a "polynomial" kernel for classification. Data was split into an 80:20 ratio for training and testing purposes. The trained model was then evaluated on the remaining 20 percent of the data, and the accuracy of the model was recorded. We utilized the "svm" function from the "e1071" library (version 1.7.14) (Meyer et al., 2020) to fit the SVM model within the R environment (version 4.3.0).

Results:

1. Discrepancy in Number of Bottleneck Simulations:

The number of bottleneck simulations was only 4970 instead of the expected 5000. This discrepancy arose from the naming convention used for saving the files, which appends the bottleneck coefficient at the end to maintain a unique name for each file. The convention was based on a bottleneck coefficient with three decimal places, but some simulations differed only in the fourth decimal place. As a result, only one copy of such simulations was saved.

2. Descriptive statistical analysis of various summary statistics:

We conducted a descriptive statistical analysis of diverse summary statistics to observe their variations between bottleneck and selection scenarios. Non-parametric one-sided Wilcoxon rank sum test was conducted to compare the Tajima's D values between two scenarios. As the assumption of normality doesn't hold for the distribution of Tajima's D (**Shapiro-Wilk normality test, W = 0.857, p-value < 2.2e-16**), we performed nonparametric test. The test revealed a significant difference (**W = 23814008 , p-value < 2.2e-16**), indicating that the mean Tajima's D value for bottleneck scenario (**mean = 0.146**) is higher compared to the selection scenario.

The Number of mutations was significantly greater in case of Bottleneck Scenario (**One tail Wilcoxon rank sum test, W= 23638890, p-value < 2.2e-16**), whereas the average haplotype was significantly greater in the case of the selection scenario (**One tail Wilcoxon rank sum test, W = 23652696, p-value < 2.2e-16**). The rare variants reported by SFS category 1was significantly higher in selection scenario compared to the selection scenario (**One tail Wilcoxon rank sum test, W= 23169570, p-value < 2.2e-16**). There were significantly higher common variants in case of Bottleneck as compared to the selection scenario when we compare common variant categories namely SFS category 6 (**One tail Wilcoxon rank sum test, W= 23569611, p-value < 2.2e-16**), SFS category 7(**One tail Wilcoxon rank sum test, W= 23960834, p-value < 2.2e-16**), SFS category 8 (**One tail Wilcoxon rank sum test, W= 23620994, p-value < 2.2e-16**). The most common variant category of side frequency spectrum namely SFS category 9 (**One tail Wilcoxon rank sum test, W= 13315727, p-value < 2.2e-16**) reported higher proportion for Selection scenario compared to the Bottleneck Scenario.

Scenario ⇨	Bottleneck			Selection		
Summary Statistic ⇩	Mean	Median	$\sqrt{\sigma^2}$	Mean	Median	$\sqrt{\sigma^2}$
Tajima's D	0.1458	0.1659	5.737248e-01	-2.4669	-2.6297	4.241868e-01
Tajima's π	1.860e-03	1.890e-03	4.279270e-04	1.900e-04	1.305e-04	1.738553e-04
Average Haplotype length	5208 bp	4305 bp	3.496327e+03	18770	18885	2.943085e+03
Number of Mutations	915.9	944	1.782569e+02	332.3	306.0	9.643626e+01
SFS category 1	0.09019	0.08610	2.857480e-02	0.18064	0.17964	4.402595e-02
SFS category 2	0.056633	0.054713	2.106400e-02	0.09107	0.08795	2.958297e-02
SFS category 3	0.076544	0.075398	2.773590e-02	0.08375	0.07917	2.941041e-02
SFS category 4	0.07656	0.07660	2.936662e-02	0.043220	0.036688	2.532954e-02
SFS category 5	0.05532	0.05466	2.367553e-02	0.014556	0.008596	1.736668e-02
SFS category 6	0.1225	0.1237	3.975321e-02	0.011125	0.003210	2.246717e-02
SFS category 7	0.1817	0.1808	5.513583e-02	0.004397	0.000000	1.987252e-02
SFS category 8	0.07001	0.06708	2.924670e-02	0.002459	0.000000	1.233394e-02
SFS category 9	0.07119	0.06664	3.379701e-02	0.08940	0.08520	7.371467e-02

Table 3: The mean, median, and standard deviation of different statistics grouped by evolutionary scenario. The statistics are based on 5000 selection scenario simulations and 4970 bottleneck scenario simulations.

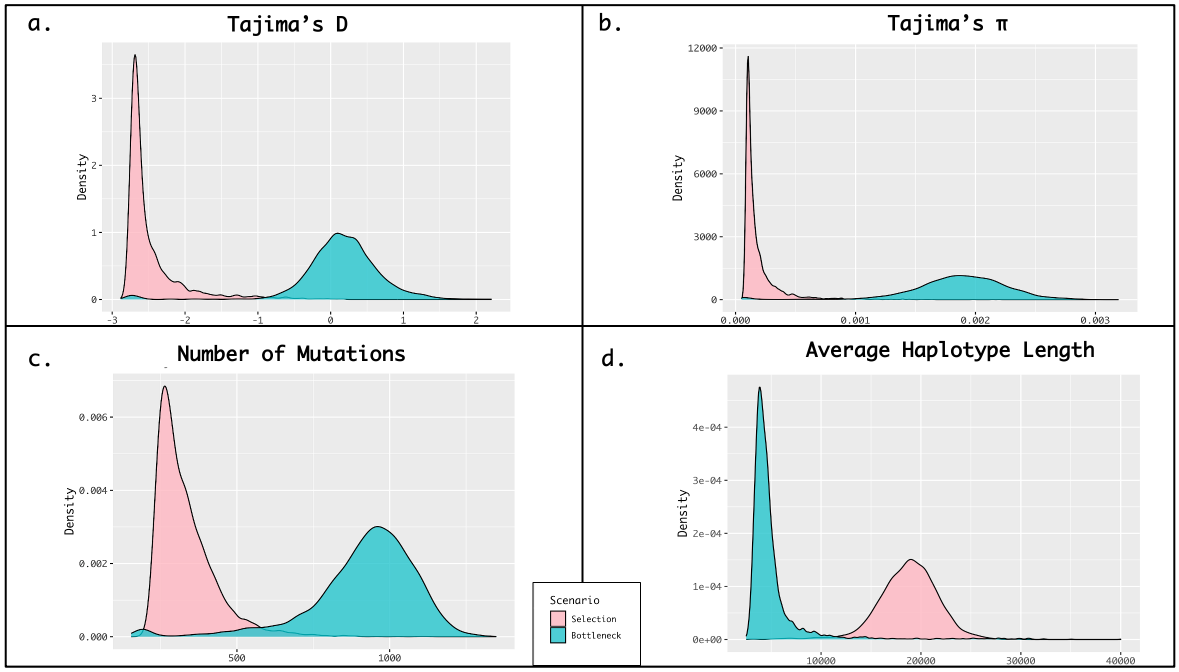

Figure 10: Density plot of different statistics grouped by evolutionary scenario: a) Tajima's D b) Tajima's pi c) Number of mutations d) Average haplotype length.

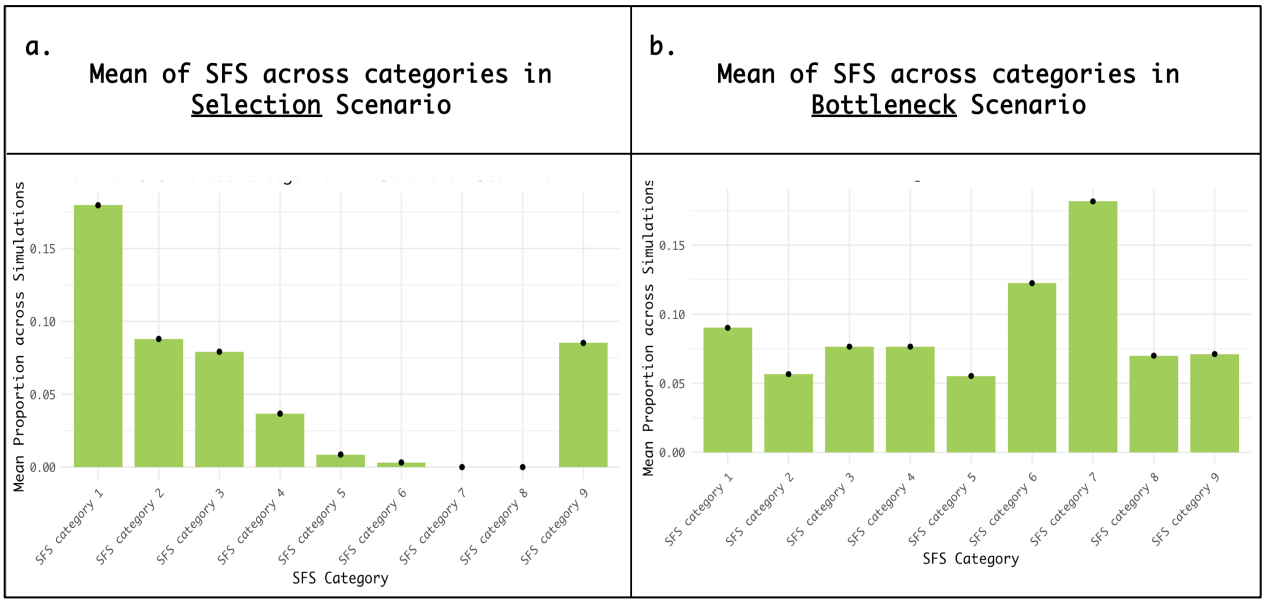

Figure 11. Bar plot representing the mean of SFS categories along simulations by evolutionary scenario: a) Selection scenario (based on 5000 simulations) b) Bottleneck scenario (based on 4970 simulations)

3. *CNN can accurately predict the Evolutionary Scenario from alignment images.*

The Convolutional Neural Network (CNN), trained on **9970** PNG images comprising **5000** Selection Scenario images and 4970 Bottleneck Scenario images, achieved the lowest estimated.

misclassification error of 0.027. Each image measured 200x200 pixels. The final network consisted of a single 2D convolutional layer with an 8 square convolutional filter of shape 7x7. The confusion matrix for the final model was visualized in Figure ().

During experimentation with learning rates, we determined the optimal learning rate for this specific architecture and complexity of data to be **0.0001**. In total, **20818404** parameters (calculation in supplementary) were trained using stochastic gradient descent, including 391 parameters belonging to the convolutional layer. The order and associated parameters per layer are presented in tabulated form in Table (). The estimated accuracy, precision, recall and F1 score calculated from the confusion matrix were **0.9738, 0.9656, 0.9815** and **0.9735** respectively. In the context of evaluating the performance of a machine learning model, several metrics are commonly used. Accuracy refers to the proportion of correctly predicted observations out of the total number of observations. Precision measures the correctness of positive predictions by calculating the ratio of true positives to the sum of true positives and false positives. Recall, also known as sensitivity, quantifies the ability of the model to identify all relevant instances by determining the ratio of true positives to the sum of true positives and false negatives. Finally, the F1 score provides a balance between precision and recall by taking their harmonic mean. It serves as a single score that represents both precision and recall,

making it a useful metric for evaluating the overall performance of a model in classification tasks.

Layer	Description	Input Shape	Output Shape	Parameters
Conv2d	Convolutional layer with 8 filters	(1, H, W)	(8, H, W)	(8 * 7^2)
ReLU	Rectified Linear Activation	(8, H, W)	(8, H, W)	0
MaxPool2d	Max pooling layer with 2x2 kernel	(8, H, W)	(8, H/2, W/2)	0
Flatten	Flattens the input into a 1D tensor	(8, H/2, W/2)	(8 * H/2 * W/2,)	0
Linear	Fully connected layer with 256 units	(8 * H/2 * W/2,)	(256,)	(8 * H/2 * W/2 * 256) + 256
ReLU	Rectified Linear Activation	(256,)	(256,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(256,)	(256,)	0
Linear	Fully connected layer with 128 units	(256,)	(128,)	(256 * 128) + 128
ReLU	Rectified Linear Activation	(128,)	(128,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(128,)	(128,)	0
Linear	Fully connected layer with 2 units	(128,)	(2,)	(128 * 2) + 2

Table 4: Description of Layers in a Convolutional Neural Network Architecture trained on compressed SNP matrix. Each row represents a layer in the network, including its functionality, input shape, output shape, and the number of parameters. The input and output shapes are represented in the convention (channels, height, width) for 2D data.

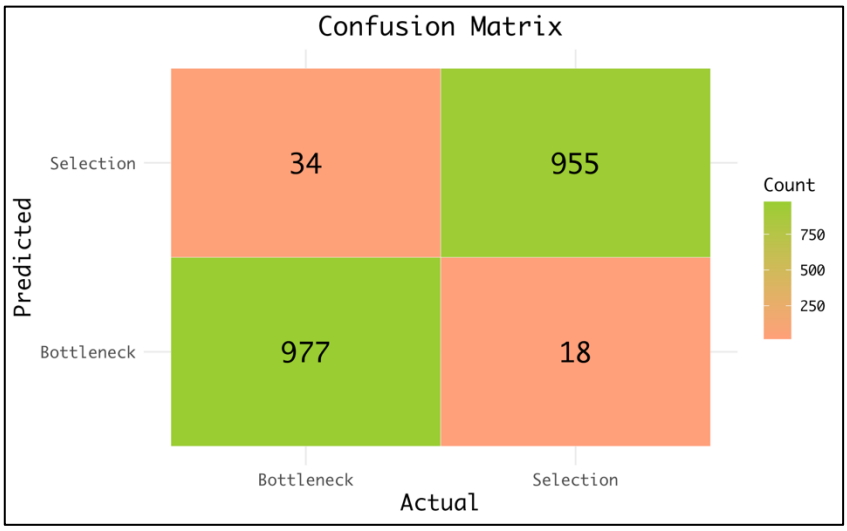

Figure 12: Confusion matrix for CNN model trained on compressed SNP matrix images. The x-axis represents the actual classes, while the y-axis represents the predicted classes by the model.

Effect of Training Size on Model Performance

To analyze the impact of varying training data sizes on both the accuracy and precision of the model, we conducted a series of training sessions. Specifically, we trained the model using progressively larger training datasets, starting from 2000 data points and incrementing by 2000 up to a maximum of 9970 data points. Each training session consisted of 20 epochs, with a fixed learning rate of 0.001. It's noteworthy that we deliberately opted for a learning rate of 0.001 instead of the optimum value of 0.0001. This decision was made to intentionally amplify the variance during the initial epochs of training. By doing so, we aimed to allow the model to

explore a wider range of solutions early in the training process, potentially leading to a more comprehensive understanding of the data and generalization capabilities based on number of training simulations. The linear model (Accuracy ~ Training size) fitted at randomly chosen **epoch 8** shows that accuracy is linearly dependent on training size (**intercept = 47.22, slope = 0.0050, p-value = 0.0458)**. The ANOVA test indicated significant difference in mean of loss among different training sizes (**F (4,95) = 12.36, p-value = 3.94e-08**). There was also a significant difference in mean of accuracy among different training size test (**F (4,95) = 5.258, p-value = 0.00072**). Post-hoc analysis, conducted using Tukey Honestly Significant Difference (HSD), aimed to identify groups with significantly different accuracy and loss (as depicted in the figures). The model achieved its highest accuracy when trained on the full dataset of 9970 simulations, yielding a misclassification error of 0.0529. Conversely, the lowest accuracy was observed in models trained on 4000 simulations, with a closely followed misclassification error of 0.1198, and on 2000 simulations, with an error of 0.0885. Notably, the observed accuracy difference between models trained on 2000 and 4000 simulations was not statistically significant (**Adjusted p-value = 1**), as shown in the figure.

We discern two main groupings concerning accuracy: one with smaller training sizes (**2000** and **4000** simulations) and another with larger sizes (**6000**, **8000**, and **9970** simulations), each exhibiting adjusted p-values < 0.05 compared to the other group (while more than 0.05 compared within the group). The larger training size has significantly higher accuracy compared to the one with smaller training size (**One tail Wilcoxon rank sum test, W= 1932, p-value = 1.323e-07**). Non-parametric test was conducted as accuracy wasn't normally distributed (**Shapiro-Wilk normality test for larger training dataset group W = 0.695, p-value = 7.186e-10 and smaller training size group W = 0.877, p-value = 0.00044**).

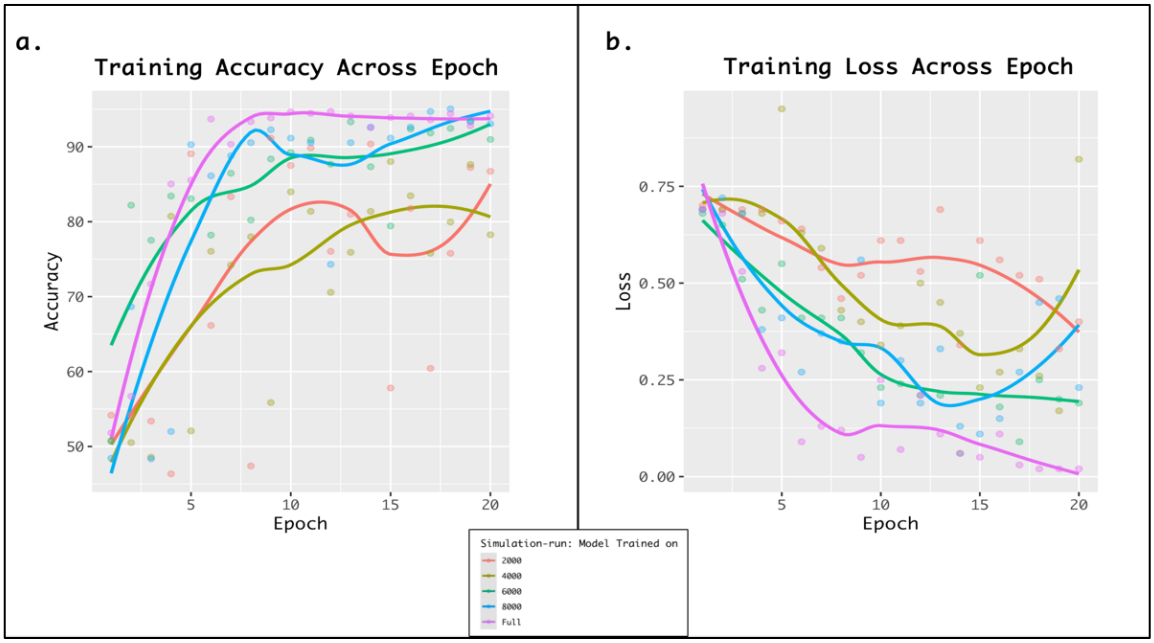

Figure 13: Smooth line plot obtained using the loess method. a) Represents the training accuracy epoch-based colored based on training size. b) Represents the training loss epoch-based colored based on training size.

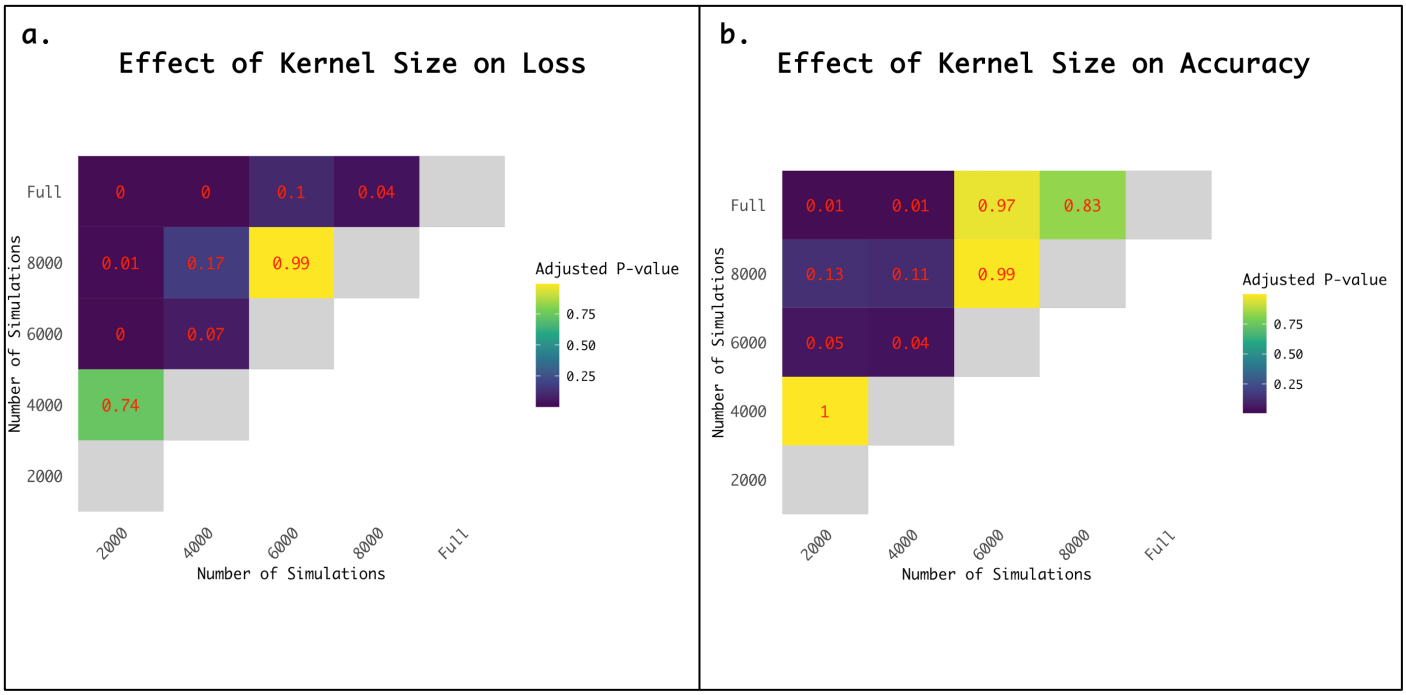

Figure 14: Adjusted pairwise p-values plotted as filled upper triangular matrix obtained from Tukey's test results between different training sizes. a) Represents the p-value for pairwise accuracy comparison. b) Represents the p-value for pairwise loss comparison.

4. CNN directly trained on SNP Matrix outperformed.

CNN trained directly on the 9970 SNP matrix before converting them into visual image gave the estimated accuracy of 0.9793 and misclassification error of 0.0207. The model outperformed the one directly trained on the image. Each of SNP matrix has the dimensions of 100*1000. The final network has the single 2D Convolutional filter with 32 rectangular convolutional filter of shape 15*9. The confusion matrix for the model is shown in figure (). The model had 50980110 parameters trained using stochastic gradient descent. The CNN architecture with layers and parameter is tabulated in table (). The estimated precision, recall and F1 score for the trained model was 0.9765, 0.9805 and 0.9785.

The misclassified bottleneck case had a relatively high bottleneck intensity, with a mean of 0.982, a median of 0.985, and a standard deviation of 0.012. Conversely, the misclassified selection case had a relatively low selection coefficient, with a mean of 0.033, a median of 0.015, and a standard deviation of 0.058.

Layer	Description	Input Shape	Output Shape	Parameters
Conv2d	Convolutional layer with 32 filters	(1, H, W)	(32, H, W)	(32 * 7^2)
ReLU	Rectified Linear Activation	(32, H, W)	(32, H, W)	0
MaxPool2d	Max pooling layer with 3x3 kernel	(32, H, W)	(32, H/3, W/3)	0
Flatten	Flattens the input into a 1D tensor	(32, H/3, W/3)	(32 * H/3 * W/3)	0
Linear	Fully connected layer with 256 units	(32 * H/3 * W/3)	(256,)	(32 * H/3 * W/3 * 256) + 256
ReLU	Rectified Linear Activation	(256,)	(256,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(256,)	(256,)	0
Linear	Fully connected layer with 128 units	(256,)	(128,)	(256 * 128) + 128
ReLU	Rectified Linear Activation	(128,)	(128,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(128,)	(128,)	0
Linear	Fully connected layer with 2 units	(128,)	(2,)	(128 * 2) + 2

Table 5: Description of Layers in a Convolutional Neural Network Architecture trained on uncompressed SNP matrix. Each row represents a layer in the network, including its functionality, input shape, output shape, and the number of parameters. The input and output shapes are represented in the convention (channels, height, width) for 2D data.

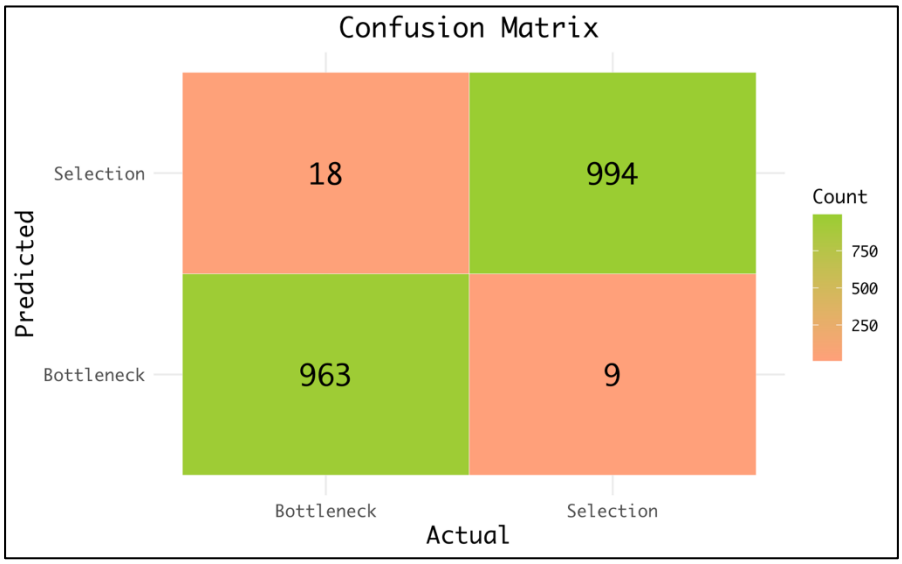

Figure 15: Confusion matrix for CNN model trained on uncompressed SNP matrix. The x-axis represents the actual classes, while the y-axis represents the predicted classes by the model.

Effect of Kernel shape on Model Performance

We analyzed the effect of changing kernel size on the model by training the same model using kernel shapes of 7*7, 9*9, 15*9, and 9*15 for 20 epochs. The square filters, i.e., 7*7 and 9*9, performed significantly better than the rectangular filters, i.e., 15*9 and 9*15 (One-tailed Wilcoxon rank sum test, W = 1600, p-value = 6.137e-15). However, the square filters (7*7 and 9*9) performed worse than a random classifier, with an accuracy less than 0.50. On the contrary, the rectangular filters (15*9 and 9*15) achieved the highest estimated accuracy of 0.9798 and a misclassification error of 0.0202.

During the first epoch of the model with a filter size of 9*9, the accuracy was 0.8206. However, accuracies in subsequent epochs were less than 0.5 (refer to the figure). This

discrepancy in performance suggests that the high accuracy during the first epoch might be due to favorable parameter initialization rather than meaningful learning from the training data.

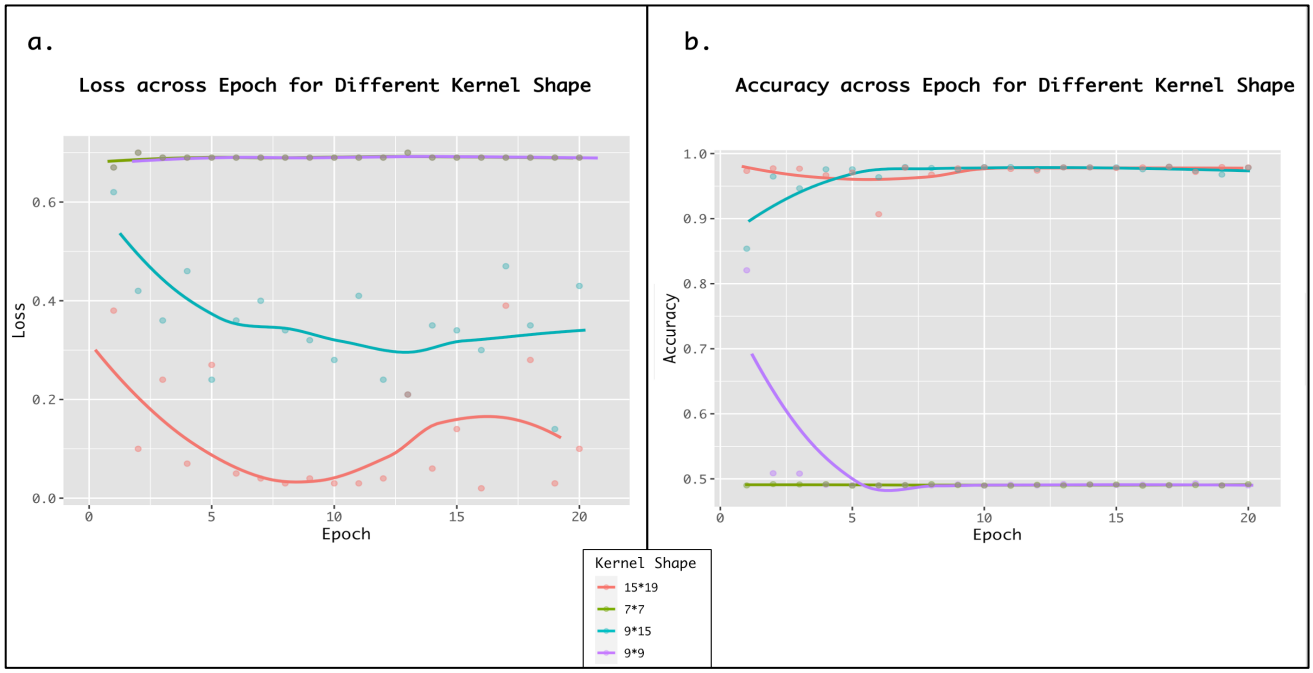

Figure 16: Smooth line plot obtained using the loess method. a) Represents the training loss epoch-based colored based on Kernel Shape. b) Represents the training accuracy epoch-based colored based on Kernel Shape.

5. *CNN trained on the Windowed Site Frequency Spectrum (SFS):*

Layer	Description	Input Shape	Output Shape	Parameters
Conv2d	Convolutional layer with 32 filters	(1, 100, 101)	(32, 100, 101)	(32 * 5^2)
ReLU	Rectified Linear Activation	(32, 100, 101)	(32, 100, 101)	0
MaxPool2d	Max pooling layer with 2x2 kernel	(32, 100, 101)	(32, 100/2, 101/2)	0
Flatten	Flattens the input into a 1D tensor	(32, 100/2, 101/2)	(32 * 100/2 * 101/2,)	0
Linear	Fully connected layer with 256 units	(32 * 100/2 * 101/2,)	(256,)	(32 * 100/2 * 101/2 * 256) + 256
ReLU	Rectified Linear Activation	(256,)	(256,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(256,)	(256,)	0
Linear	Fully connected layer with 128 units	(256,)	(128,)	(256 * 128) + 128
ReLU	Rectified Linear Activation	(128,)	(128,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(128,)	(128,)	0
Linear	Fully connected layer with 2 units	(128,)	(2,)	(128 * 2) + 2

Table 6: Description of Layers in a Convolutional Neural Network Architecture trained on windowed Site frequency spectrum data. Each row represents a layer in the network, including its functionality, input shape, output shape, and the number of parameters. The input and output shapes are represented in the convention (channels, height, width) for 2D data.

CNN model trained on the Windowed Side Frequency spectrum gave the highest estimated accuracy of 0.9869 and misclassification error of 0.0131. The final architecture has one 2d convolutional filter with 32 filters of the shape 5*5. There are two hidden fully connected layer, each is followed by dropout regularization with probability value of 0.2. The model has

20719010 parameters including 800 shared parameters for the convolutional filter trained using stochastic gradient decent. The confusion matrix for the model is shown in the figure whereas the architecture of model is tabulated in the Table (). The precision, recall and F1 score for the final model was 0.9882, 0.9910 and 0.9866 respectively.

Effect of Kernel Size on Model Performance

We analyzed the effect of kernel size on the performance of the model. The model was trained with increasing kernel sizes of 3x3, 5x5, 7x7, and 9x9. Based on the Kruskal-Wallis rank sum test, there was a significant effect on the accuracy of the model depending on the kernel size (χ^2 = 19.91, df = 3, p-value = 0.00018). However, during the post hoc analysis using Tukey's Honestly Significant Difference (HSD) test, we didn't find a significant difference in accuracy between any pairwise comparisons (at the significance level of 5 percent). Furthermore, there were no significant differences in the loss based on the kernel size (Kruskal-Wallis rank sum test: χ^2 = 0.34, df = 3, p-value = 0.94). Visualizations of the Tukey HSD results for both accuracy and loss based on kernel size are presented in the figure.

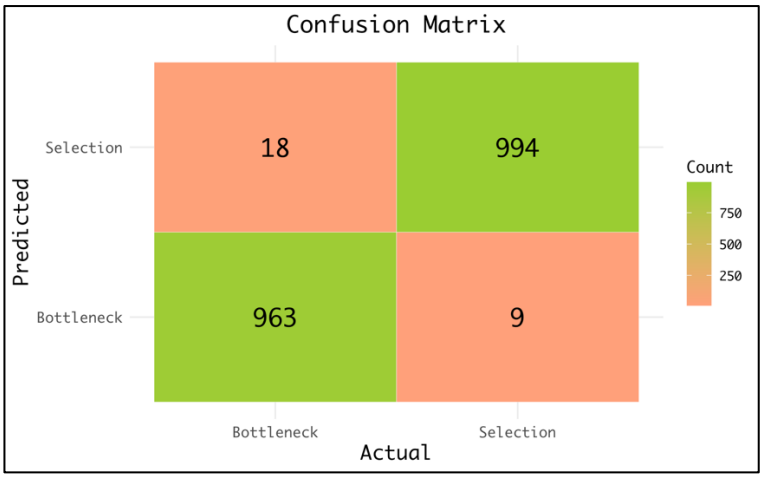

Figure 17: Confusion matrix for CNN model trained on Windowed Site frequency spectrum. The x-axis represents the actual classes, while the y-axis represents the predicted classes by the model.

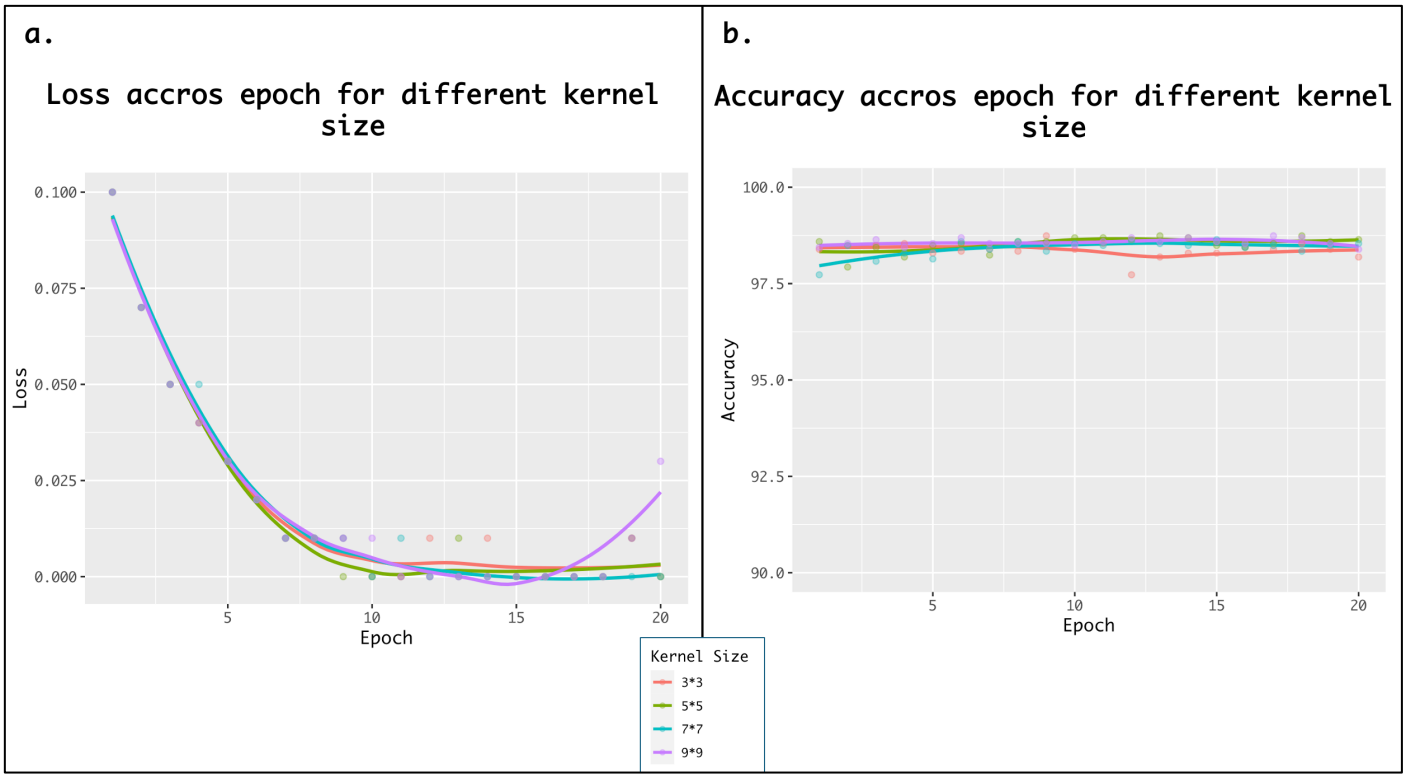

Figure 18: Smooth line plot obtained using the loess method. a) Represents the training loss epoch-based colored based on Kernel Size. b) Represents the training Accuracy epoch-based colored based on Kernel Size.

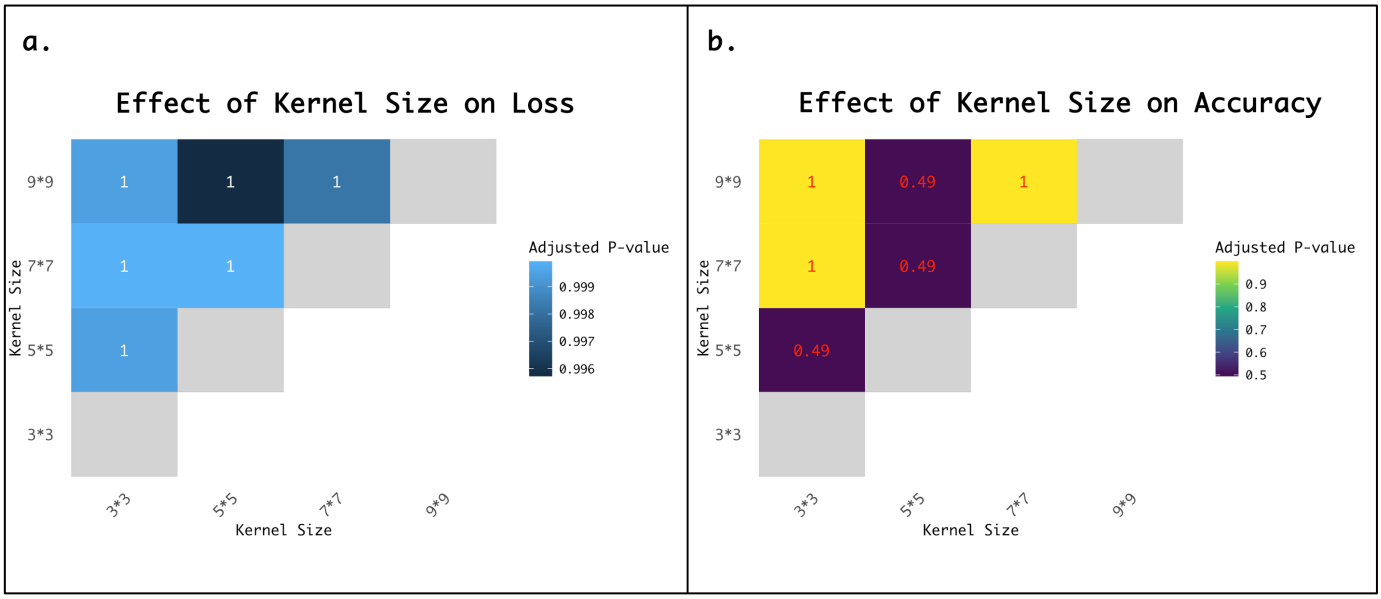

Figure 19: Adjusted pairwise p-values plotted as filled upper triangular matrix obtained from Tukey's test results between different training sizes. a) Represents the p-value for pairwise loss comparison. b) Represents the p-value for pairwise accuracy comparison.

6. *Multimodal CNN trained both on windowed Site Frequency Spectrum (SFS) and SNP matrix*:

We trained the CNN model using two different modalities: SNP matrix and Side frequency spectrum (SFS). Prior to feeding the data into the fully connected layer, we combined the information from both modalities. Before the fusion, each modality underwent distinct convolutional operations. The SNP matrix data underwent convolution using 16 filters with a shape of 9x15. On the other hand, the windowed SFS data was subjected to convolution using 32 filters with a shape of 9x9.

The estimated minimum misclassification error for the final multimodal CNN model was 0.01411 and accuracy of 0.9859. The confusion matrix for the model is shown in the figure. The precision, recall and the F1 score is 0.9774, 0.9901 and 0.9837 respectively. The model didn't does perform much better compared to the one trained on single modality.

Layer	Description	Input Shape	Output Shape	Parameters
SNP Matrix CNN				
1. Conv2D	Convolutional layer with 16 layer	(1, H^m, W^m)	(16, H^m, W^m)	(9*15) *16
2. ReLu	Rectified Linear Activation	(16, H^m, W^m)	(16, H^m, W^m)	0
3. MaxPool2d	Max Pooling Layer of 2x2 Kernel	(16, H^m, W^m)	(16, H^m/2, W^m/2)	0
4. Flatten	Flattens the input into 1D Tensor	(16, H^m/2, W^m/2)	16*H^m/2*W^m/2	0
Windowed SFS CNN				
1. Conv2D	Convolutional layer with 32 layer	(1, H^a, W^a)	(32, H^a, W^a)	(9*9) *32
2. ReLu	Rectified Linear Activation	(32, H^a, W^a)	(32, H^a, W^a)	0
3. MaxPool2d	Max Pooling Layer of 2x2 Kernel	(32, H^a, W^a)	(32, H^a/2, W^a/2)	0
4. Flatten	Flattens the input into 1D Tensor	(32, H^a/2, W^a/2)	32*H^a/2*W^a/2	0
Linear	Fully connected layer with 256 units	((32*(H^a/2)*(W^a/2)+ (16*(H^m/2)*(W^m/2),)	(256,)	(((32*(H^a/2)*(W^a/2)+ (16*(H^m/2)*(W^m/2))*256)+256
ReLU	Rectified Linear Activation	(256,)	(256,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(256,)	(256,)	0
Linear	Fully connected layer with 128 units	(256,)	(128,)	(256*128) + 128
ReLU	Rectified Linear Activation	(128,)	(128,)	0
Dropout	Dropout layer with p=0.2 dropout rate	(128,)	(128,)	0
Linear	Fully connected layer with 2 units	(128,)	(2,)	(128*2) + 2

Table 6: Description of Layers in a Multimodal Convolutional Neural Network Architecture trained on windowed Site frequency spectrum data and uncompressed SNP matrix. Each row represents a layer in the network, including its functionality, input shape, output shape, and the number of parameters. The input and output shapes are represented in the convention (channels, height, width) for 2D data.

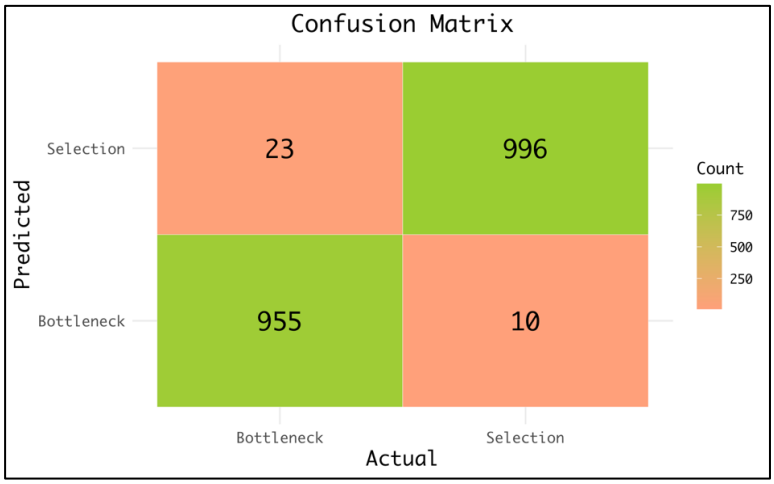

Figure 20: Confusion matrix for multimodal CNN model trained on Windowed Site frequency spectrum and uncompressed SNP matrix. The x-axis represents the actual classes, while the y-axis represents the predicted classes by the model.

7. PCA (Principal Component Analysis):

Principal component analysis performed on summary statistics visually displays clustering based on the evolutionary scenario, specifically the bottleneck or selection scenario, when plotted across Principal Component 1 and Principal Component 2. The first two principal components account for 73.217 percent of the variance in our data, with Principal Component 1 contributing the most variance (63.72 percent). However, our analysis reveals several cases of misclustering, predominantly observed in instances where selection scenario clustering intersects with the bottleneck scenario. The other principal component when plotted against each other showed no major clustering based on evolutionary scenario.

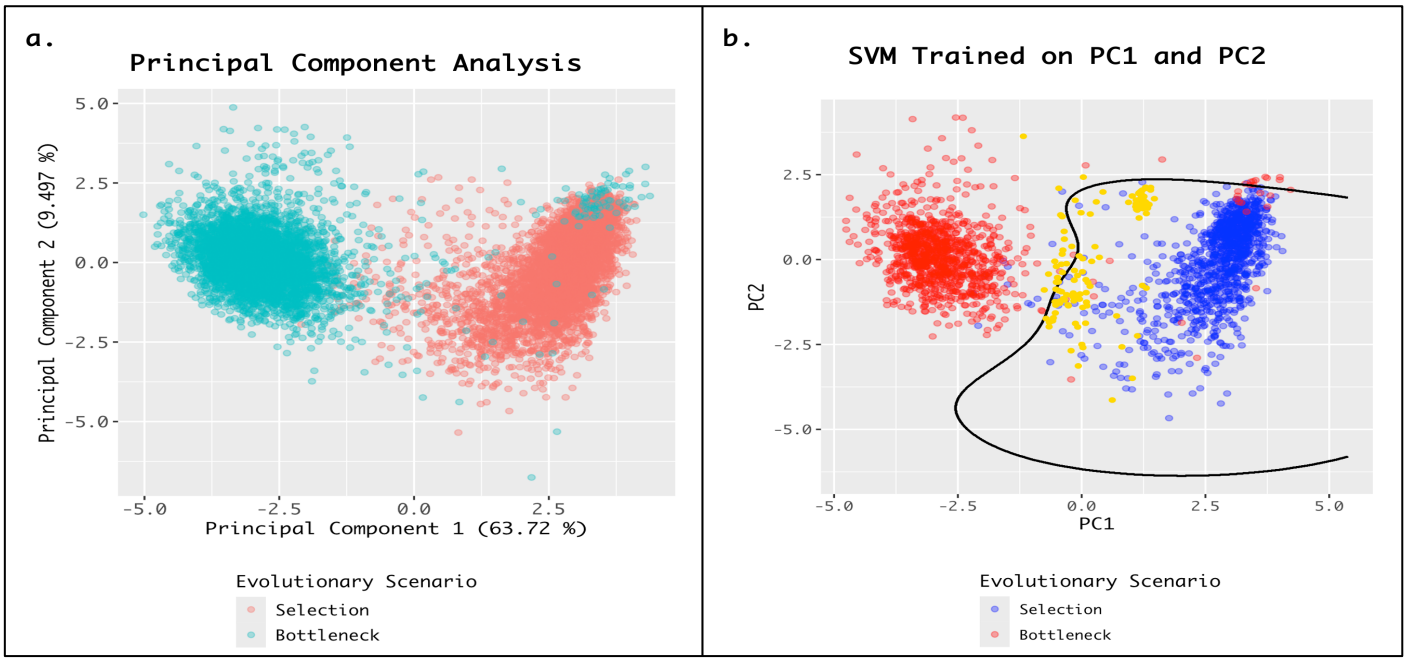

Figure 20: a) Principal component analysis plot with PC1 along the x-axis (accounting for 63.72% variation in our data) and PC2 along the y-axis (accounting for 9.497% variation in our data), colored based on evolutionary scenario. b) Support Vector Machine (SVM) boundary plot based on a model

trained on PC1 and PC2. The golden yellow represents support vectors, while the black line indicates the decision boundary.

8. SVM (Support Vector Machine) analysis:

We trained two different SVM models to predict the evolutionary scenario, namely bottleneck or selection. The first model was trained on the summary statistics obtained from 9759 simulation runs (**Evolutionary scenario ~** $\sum_1^8 Categorical\ SFS_i$ **+ Tajima's D + Tajima's π + Number of Mutations + Average Haplotype Length**) using a radial kernel (kernel = "radial"). The estimated misclassification error for the final trained model was 0.01119, with 307 simulations serving as support vectors. The confusion matrix for the model is visualized in the figure, and its precision, recall, and F1 score calculated from the confusion matrix are 0.9918, 0.9858, and 0.9888, respectively. We subsetted the data based on the misclassified simulation runs. The misclassified selection scenario (**mean = 0.213 and** $\sqrt{\sigma^2}$ **= 0.273**) has a statistically lower selection coefficient compared to the correctly classified one (**One tail Wilcoxon rank sum test, W= 1556, p-value = 0.0017**, whereas the misclassified bottleneck scenario (**mean = 0.968 and** $\sqrt{\sigma^2}$ **= 0.042**) has a statistically higher bottleneck intensity compared to the correctly classified one (**One tail Wilcoxon rank sum test, W=13138, p-value = 1.159e-09**).

The second SVM model was trained on the first two principal components obtained from Principal Component Analysis (**evolutionary scenario ~ Principal Component 1 + Principal Component 2**). The estimated misclassification error for this model was 0.01882, with 370 support vectors. Although the misclassification error is slightly higher compared to the model trained with summary statistics, its confusion matrix is visualized in the figure, and its precision, recall, and F1 score are 0.9884, 0.9731, and 0.9807, respectively. A radial kernel was chosen for both models as it provided higher accuracy compared to linear or polynomial kernels. The decision boundary for this model is visualized in Fig. Firstly, we created a grid with 100 divisions along PC1 and 1000 divisions along PC2, resulting in a total of 10000 data points. Then, we predicted the output for each data point.

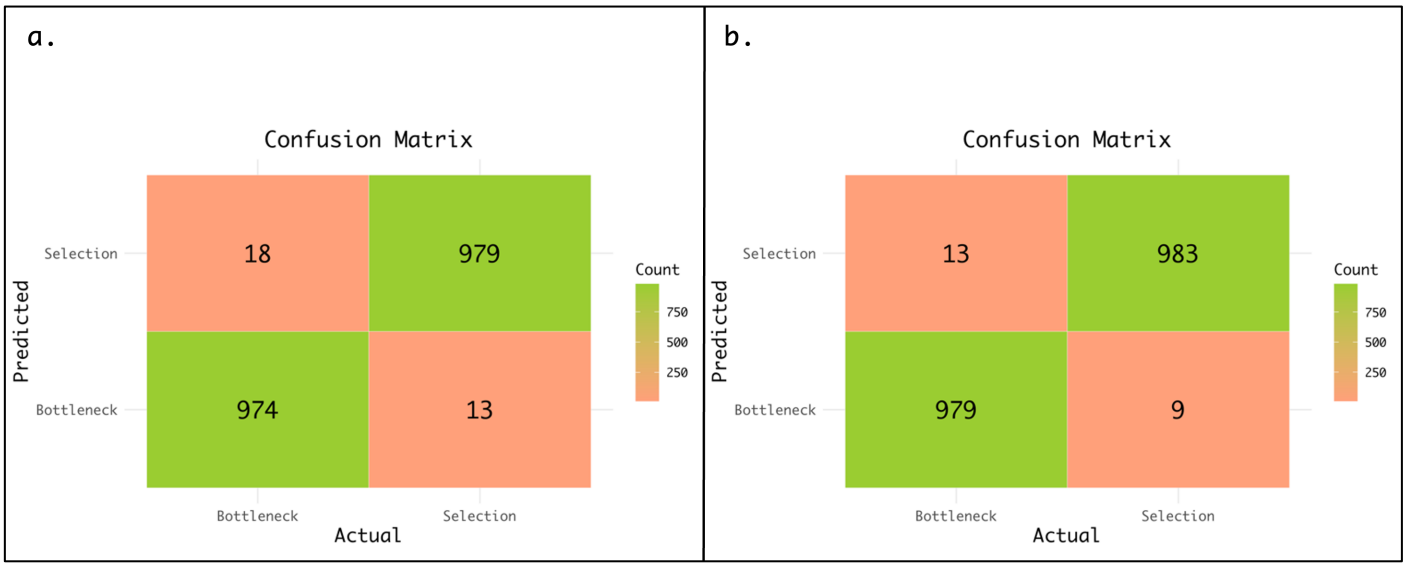

Figure 20: Confusion matrix for SVM model trained on a.) First two Principal Component b.) Summary Statistics. The x-axis represents the actual classes, while the y-axis represents the predicted classes by the model.

9. *Comparison of different models and methods*:

All models, trained on different data modalities or frameworks, are compared based on their misclassification rates for both selection and bottleneck scenarios. The misclassified bottleneck case in particular model is divided by total number of bottleneck case in the testing set on which model is evaluated. This quotient is then multiplied by 1000 to project the expected number of misclassified bottlenecks per 1000 bottleneck scenario in training set. The same procedure is applied to evaluate misclassified selection bases by model per 1000 selection scenario in training set. The order of accuracy for selection scenario is as follows:

```
SVM (Summary Statistics) = CNN (Windowed SFS) > Multimodal CNN > SVM (PC1 and
                    PC2) > CNN (image) > CNN (SNP matrix)
```

Conversely, for the bottleneck case, the order of accuracy is:

```
SVM (Summary Statistics) > CNN (Windowed SFS) = SVM (PC1 and PC2) > CNN (SNP
                    matrix) ~ Multimodal CNN > CNN (image)
```

Considering both misclassified selection and bottleneck cases together, the overall accuracy order is as follows:

```
SVM (Summary Statistics) > CNN (Windowed SFS) > SVM (PC1 and PC2) > Multimodal
                    CNN > CNN (SNP matrix) > CNN (image)
```

The comparison can be visualized in the figure (x). It is observed that there is more misclassified bottleneck case in each of the model compared to the selection scenario.

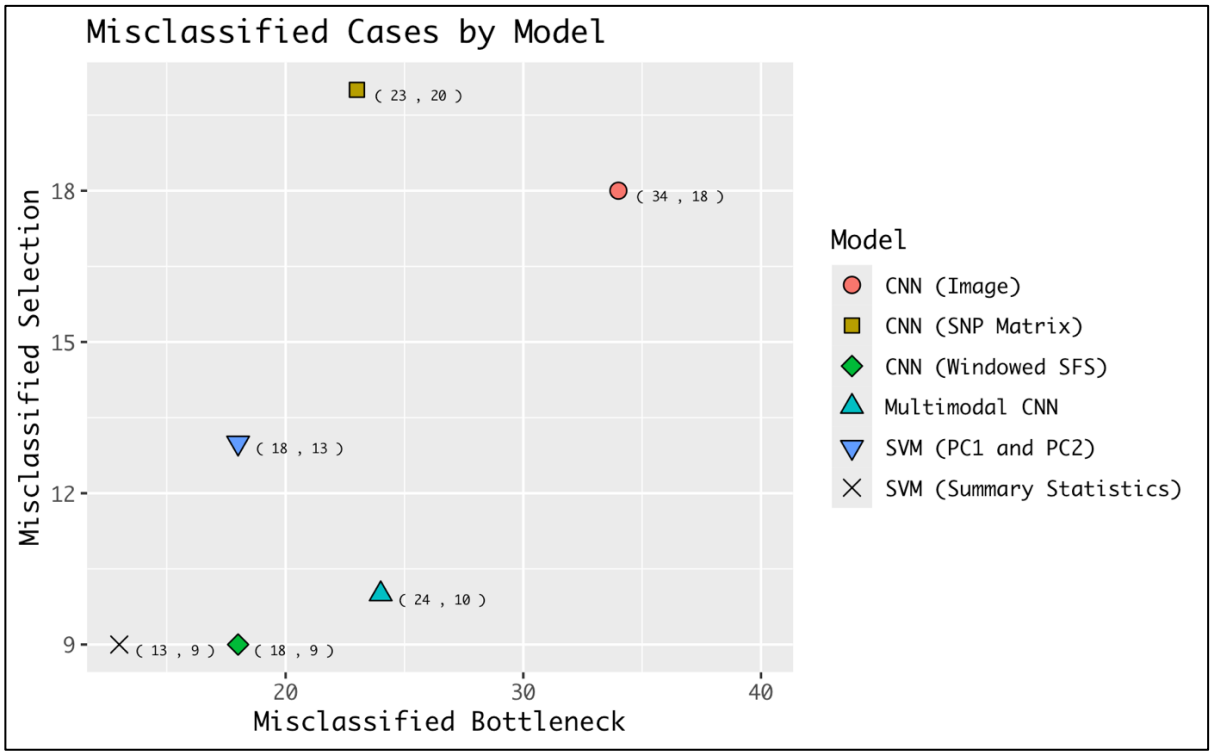

Figure: Dot plot representing misclassified bottleneck cases on the x-axis and misclassified selection cases on the y-axis. The color and shape of the icon represent the model used.

Discussion:

1. Insights from summary statistics:

The simulations exploring bottleneck and selection scenarios offer profound insights into how demographic events, such as bottlenecks and selective pressures, shape genealogies. Notably, the higher Tajima's D observed in the bottleneck scenario compared to the selection scenario signifies an excess of intermediate-frequency variants in the former. This disparity suggests that during the bottleneck scenario, certain lineages failed to coalesce at the bottleneck event, resulting in a few lingering ancestral lineages with extended coalescence times. Consequently, this prolonged coalescence not only facilitated an increased number of mutations but also led to the higher prevalence of intermediate-frequency variants within the population. Conversely, in the selection scenario, lineages exhibit more recent coalescence compared to the bottleneck scenario. This rapid coalescence can be attributed to the inability of lineages to escape the selective sweep, likely due to the absence of recombination events during the sweep. Moreover, the confinement of the selected mutation to a single locus on a solitary chromosome further amplifies this convergence, reflecting the characteristic nature of a hard selective sweep(Garrick, 2023).

Quite a few simulations were observed to have overlapped summary statistics between the selection and bottleneck scenarios. In the bottleneck scenario, these instances occur with high bottleneck intensity, where the overlap with selection scenarios suggests that most lineages coalesce around the bottleneck due to its heightened intensity, resulting in a genealogy like that observed in many cases of the selection scenario. In the selection scenario, such overlaps are observed in cases with low selection coefficients, so low that recombination events transpire during the selective sweep, elongating the time to the most recent common ancestor (TMRCA) as observed in bottleneck scenarios. Simulations reveal the parameter space that leads to similar genealogies between bottleneck and selection scenarios, making them challenging to distinguish from summary statistics alone. In summary, these findings illustrate how demographic and selective events shape population genealogies, with implications for patterns of genetic variation and coalescence dynamics(Arnab et al., 2023; Garrick, 2023; Johri et al., 2022; Koropoulis et al., 2020; Mughal & DeGiorgio, 2019).

2. *A case for CNN application in Population History Inference*:

2.1. CNNs are capable of effectively distinguishing between evolutionary scenarios, even in challenging cases such as bottleneck or selection scenarios.

Convolutional Neural Networks have effectively demonstrated their ability to learn and differentiate patterns associated with evolutionary events, such as population bottlenecks or selection, encoded in our genome. Although as shown their accuracy may be slightly lower compared to summary statistic approaches, it remains comparable. This fact is substantiated by the high accuracy achieved by CNNs trained solely on compressed images as shown in results, which misclassified only a minuscule fraction of simulations—52 out of 9970 runs. The success of CNNs can be attributed to their unique properties, such as translational invariance(Mouton et al., 2020). This means they can recognize patterns irrespective of their position in the input data. In the context of genomic analysis, where the effects of selection and bottleneck events

can occur with high positional variance across the genome, this property becomes particularly advantageous. In contrast, fully connected neural networks may struggle with such tasks due to their inability to effectively handle positional variance associated with training as well as evolutionary event. Their performance could be hindered by the need to account for the specific positions of genomic features, whereas CNNs can identify patterns across a broader range, thanks to their translational invariance.

The comparison between the full SNP matrix and the compressed image form underscores the efficiency of Convolutional Neural Networks (CNNs) in learning patterns from reduced data sizes. Despite the full SNP matrix being 2.5 times larger in data size compared to the compressed image form, the CNN trained on the full SNP matrix only exhibits a marginal increase of 0.52 percent in performance compared to the one trained on compressed images. This ability of CNNs to generalize well from compressed representations highlights their robustness to noise and redundancy in the data. By focusing on essential features and patterns, CNNs can effectively learn representations that generalize across different data modalities and compression levels. This is particularly advantageous in scenarios where computational resources or data storage capacities are limited, as CNNs inference model can operate efficiently even with reduced data sizes. Furthermore, the minimal performance gap between CNNs trained on full data and compressed representations indicates that the CNN is not overly sensitive to data size but rather prioritizes learning meaningful patterns. This aligns with the understanding that CNNs excel at extracting hierarchical representations of features, allowing them to capture essential information while filtering out irrelevant details(Crowley, 2023).

2.2. Slightly higher Accuracy of SVM model doesn't discount the CNN power in genetic analysis.

The comparison between the CNN trained on windowed side frequency spectrum (SFS) and the SVM model trained solely on summary statistics provides insights into the relative performance of these approaches in predicting evolutionary scenarios. While the CNN trained on windowed SFS achieved slightly higher accuracy compared to other CNN models, it still fell short of the SVM models trained on summary statistics although fared well than SVM model trained on summary statistics.

This discrepancy suggests that while CNNs are effective in capturing spatial patterns within the SFS data, they may not fully exploit the broader contextual information encoded in summary statistics. Summary statistics provide a condensed representation of key features across the entire dataset, offering a comprehensive overview of genetic variation and evolutionary dynamics. This summarizing capacity enables SVM models to generalize well across a wide range of cases, resulting in higher predictive accuracy.

However, it's important to note that the superior performance of the SVM model trained on summary statistics does not discount the value of CNNs in genetic analysis. CNNs excel at capturing intricate spatial relationships and patterns within data, which can be particularly useful in scenarios where the underlying structures are complex or nonlinear. Additionally, the CNN's ability to extract features directly from raw data may offer advantages in certain applications where the data distribution is not fully known or understood. Nevertheless, the slight advantage of the SVM model in this context suggests that while CNNs are adept at learning local patterns, they may struggle a bit to fully capture the global characteristics encoded in summary statistics.

While it's important to consider that the performance of a CNN is heavily influenced by both the trained data and the architecture of the network itself, it's worth noting that a CNN model trained exclusively on simulated selection and bottleneck data, which produce comparable summary statistics, may indeed slightly perform better.

2.3. Aligned Sequences complement perfectly for Convolutional Network based inference.

Summary statistics and likelihood-based inference dominate the field of population and evolutionary history inference(Johri et al., 2022; Pool et al., 2010). However, these methods sometimes encounter challenges, such as loss of information with summary statistics or complex likelihood calculations, especially for higher-dimensional data, resulting in computational intensity for likelihood-based inference methods(Flagel et al., 2019). The potential of neural network-based population inference as the next-generation solution for population genetics inference is significant. Not only are they easier to implement, but aligned sequences also serve as perfect complementary data inputs for Convolutional Neural Network-based approaches. The aligned sequences or downsized SNP matrix, as utilized in our methods, serve as a natural data type to be represented in the form of images, for which convolutional neural networks excel.

2.4. Non-Requirement of Domain Specific knowledge for Inference:

Deep learning, a product of 5th Industrial Revolution, is known for its independence from domain-specific knowledge, particularly in fields abundant with data(Raja Santhi & Muthuswamy, 2023). For instance, in IT systems, prior to the advent of deep learning-based models, malware recognition necessitated intensive domain-specific knowledge for theoretically based models(Hemalatha et al., 2021). The empirically oriented approach of deep learning has already found applications in various genetics fields, notably in Functional Genomics. Here, vast amounts of genomic data of diverse modalities are utilized to infer genomic expression(Xie et al., 2016), regulation (Sokolova et al., 2024), and disease-associated variants (Jo et al., 2022). Additionally, deep learning has been employed for detecting noncoding RNA (Baek et al., 2018) and regulatory elements(Yang et al., 2017). Pre-trained models intended for other purposes can be repurposed for new problems through transfer learning (Iman et al., 2023). In the realm of Evolutionary population genetics, training data often needs to be simulated, making domain-specific knowledge imperative. However, simulation models are generally less complex and more straightforward than those required for theoretical inference, such as for inference in the case of organisms with multiple ploidies(Flagel et al., 2019). Additionally, instead of heavily relying on precise simulations have traditionally been heavily relied upon, alternatives such as Transformers or GAEs (Generative Adversarial Networks) can be utilized to generate more extensive and diverse training data(Achuthan et al., 2022).

3. Insights into misclassified cases:

The numerical disparity in the number of misclassified bottleneck cases being higher for the selection scenario across all models, regardless of whether they were trained on summary statistics or raw SNP data, stands outs significantly. This discrepancy is stark across all models: it is 1.8 times higher for the CNN model trained on PNG images, 1.21 times higher for the CNN model trained on SNP Matrix, 2.3 times higher for the multimodal CNN model, and 2 times higher for the CNN model trained on Windowed Site Frequency Spectrum. Even with the

SVM model trained on Principal Component, there is a 1.38 times higher rate of misclassified bottleneck cases. Similarly, when trained on summary statistics, it is 2 times higher than the misclassified selection scenario.
This difference could stem from a potential bias in the simulated data, possibly due to a much higher representation of selection cases leading to similar alignments close to those of the misclassified bottleneck cases. Further investigation or possible cross-validation error analysis (Schaffer, 1993) might shed more light on this. The bias appears to be most pronounced in the multimodal CNN model, followed by the SVM trained on summary statistics and the CNN trained on windowed site frequency spectrum. As the bias exist in both CNN and SVM model, it might be associated more to data rather than model itself.

In scenarios where SVM models trained on Principal Components misclassify selection, selection coefficients typically range from 0.0092 to 0.8840. Conversely, misclassified bottleneck cases exhibit bottleneck intensities ranging from 0.8502 to 0.9984. The increased variance observed in misclassified simulation cases may stem from the additional variance introduced by selection. Sampling more from selection scenarios compared to bottleneck scenarios could prove beneficial, although it's crucial to avoid excessively high data bias which might lead to erroneous learning and results.

4. *Effect of Convolutional Kernel Shape on Accuracy:*

The CNN model trained directly on the uncompressed SNP matrix provided insights into the effect of convolutional kernel shape on classification accuracy. Capturing genomic variation is a complex task, and our results demonstrate that rectangular kernels (15x9 and 9x15) can capture it much more effectively compared to square kernels (7x7 and 9x9). A possible explanation for this observation could be the nature of genomic variation and the structure of the SNP matrix.

Genomic variation often exhibits non-uniform patterns and relationships across different regions of the genome (Shriver et al., 2004). Our results indicate that Rectangular convolutional kernels may be better suited to capture these varied patterns compared to square kernels. The elongated shape of rectangular kernels provides a larger receptive field along one dimension, allowing them to capture more diverse and elongated patterns present in the SNP matrix. This flexibility enables the model to better extract relevant features that represent the complex genomic variations inherent in the data.

On the other hand, square kernels might not efficiently capture the full extent of variation (evident by results), as they are constrained to a fixed size in both dimensions. This limitation could result in the model overlooking important patterns or relationships in the data, leading to lower classification accuracy.

Overall, the choice of convolutional kernel shape significantly impacts the model's ability to capture complex genomic variations. In this case, rectangular kernels appear to offer a more effective representation compared to square kernels. Future models aimed at analyzing genomic variation data can benefit from this insight to develop more accurate and robust architectures.

5. Relationship between training data size and model accuracy:

The results of the CNN model trained on compressed SNP data demonstrate a positive linear correlation between the training dataset size and the model's accuracy. This finding aligns with theoretical and empirical studies across various domains, which have consistently shown that increasing the amount of training data is one of the most effective ways to improve the performance of machine learning models (Cho et al., 2016).

However, the relationship between training data size and model accuracy is not unbounded. As the training dataset becomes increasingly large, the incremental gains in accuracy begin to diminish. At a certain point, further increases in training data require disproportionately large amounts of additional data to achieve marginal improvements in model performance.

Moreover, the relationship between training data size and model robustness must also be considered. While increased training data can lead to higher accuracy, it may also result in a decrease in the model's overall robustness. This trade-off highlights the importance of carefully balancing the size of the training dataset with other model design and optimization considerations.

The amount of training data can be increased through the use of data augmentation techniques(van Dyk & Meng, 2001), which have been successfully applied in various other domains to improve model performance. These techniques involve generating additional synthetic training examples by applying transformations to the existing data, thereby expanding the diversity of the training dataset without the need for manual data collection and labeling.

In conclusion, there is no universal optimal size for the training dataset of a CNN-based deep learning model. The appropriate training data size depends on the complexity of the problem, the model architecture, and the number of input and output variables. Determining the optimal training data size often requires an iterative, empirical approach, involving experimentation and evaluation of the model's performance and robustness.

6. Time as a resource:

In scientific research, the optimization of time as a critical resource is imperative. Our study involved simulations wherein each run demanded approximately 11 minutes, with 10 minutes for the execution of the forward simulator (SLiM) and an additional minute for the coalescence based burnin (pySLiM). Moreover, the training of various data models exhibited significant time discrepancies. Notably, the convolutional neural network (CNN) trained on an uncompressed single nucleotide polymorphism (SNP) matrix exhibited the lengthiest training duration, with each epoch requiring approximately 60 minutes. Conversely, the CNN model trained on the compressed SNP matrix showcased a substantially reduced training time, averaging around 10 minutes per epoch. Additionally, the support vector machine (SVM) model-based analysis demonstrated exceptional efficiency, boasting the shortest processing time among the investigated methods. While the disparity in accuracy among the models was marginal, it is anticipated that with more complex problem formulations and advancements in CNN architecture research, CNN models might attain superior accuracy.

7. Common Pitfalls and Improvement Methods for CNN Models in Population Genetics:

7.1. Order of Aligned Sequence:

Aligned sequences serve as natural inputs for convolutional neural network (CNN) based models but still the order of individuals in those aligned sequences does not convey any information relevant for inferring demographic history or population parameters. The model can learn the order of individuals during training which can lead to a less robust model. Such a model would be highly dependent on the individual order in the data. There are two possible approaches to resolve this bias:

1) Generating multiple permutations of the data with different individual orders during training.

2) Aligning individuals based on sequence similarity, the method which is employed in our model training.

The first approach introduces variability in the individual order during training which prevents the model from learning the order of individual as a relevant feature. The second approach groups individuals based on their genetic similarity, which leads to similar ordering for new cases thus providing useful insights.

the trained CNN model becomes more robust to the order of individuals in SNP array and generalizable by implementing either of the given method. These approaches force the network to learn underlying pattern rather than

7.2. Reproducibility of the Model:

Reproducibility of CNN models are crucial to implement standardized practices across different deep learning platforms. Here are some key consideration and ways with which we can have reproducible and reliable models: **Setting Random Seeds**: Ensuring reproducibility requires setting random seeds for all processes that involve randomness, such as initial parameter initialization and stochastic gradient descent during training. This practice ensures that the model's initial state and the training process are consistent across different runs, enabling reproducible results. **Platform-Independent Model Implementation:** Deep learning models implemented using different libraries, PyTorch and TensorFlow, are often not transferable which leads to platform dependent learning biases. To solve this issue, it is essential to establish common benchmarks and standardized model architectures that are platform independent. **Standardized Evaluation Metrics:** Standardized evaluation metrics should be adopted as it is crucial for comparing model performance across different studies and platforms. The performance indicative metrics, such as accuracy, precision, recall, and F1-score, should be calculated and reported. **Comprehensive Documentation:** There should be a comprehensive documentation containing model architecture, hyperparameters, training procedures, and evaluation methods as it is essential for enhancing reproducibility. This documentation should be detailed and accessible, allowing researchers to replicate the study accurately. **Open-Source Code and Data Sharing:** Open-source code and data sharing practices should be promoted as they can significantly enhance reproducibility within the scientific community. Accessible codebase and datasets can help researchers to verify research and build upon the research findings (for code used in this study see data availability section in methods).**Collaborations and Benchmarking Initiatives**: Benchmarking practices should be

established within the scientific community to facilitate the development of platform-independent models and standardized evaluation protocols. These initiatives can increase the exchange of ideas meanwhile promoting best practices. This also drive the field towards more reproducible and comparable research outcomes.

7.3. Robustness of Model:

The prominent factor which can lead to non-robust models is the quality and quantity of the training data. This can be understood through No Free Lunch (NFL) theorem, first proposed by David Wolpert and William Macready in 1997 (Adam et al., 2019).The NFL theorem states that, averaged over all possible data-generating distributions, the performance of any two optimization algorithms is equivalent when searching for the extreme of a cost function. Simply, no single optimization algorithm can be universally superior to all others across all possible problem space. NFL theorem has significant implications for machine learning models, as it suggests that the performance of a model is tied to the features and type of the data it is trained on. A model that performs exceptionally well on one dataset may not generalize effectively to another dataset with different statistical properties.

Data Quality:

The accuracy of a machine learning model is heavily affected by the quality of its training data. The training data with features like noisy, incomplete, biased and/or inconsistent might lead to the model learning these pseudo features while failing to capture the genuine underlying patterns. This might result in the lower model performance when applied to previously unknown fresh data, which differs from the training distribution.

Data Quantity:

The amount of training data is also an important component in determining model resilience. In general, larger training datasets improve model performance by providing a more complete picture of the underlying data generation process. However, if the training data is insufficient or does not appropriately represent the diversity of the issue domain, the model may be overfitting to the current data and fail to generalize effectively to new scenarios.

Ways to Improve CNN Models in Population Genetics:

- Data Augmentation: Creating synthetic samples from existing data to improve model robustness.
- To prevent bias, provide equal representation of different classes in training data.
- Regularization Techniques: Use dropout to reduce overfitting and increase generalization.
- Transfer Learning: Adapting pre-trained models for population genetics, especially when training data is sparse.

7.4. Theoretical Population Genetics informed deep learning model:

Incorporating domain-specific knowledge into neural networks, as seen in physics-informed neural networks (PINNs) (Cai et al., 2021), is a promising approach that could be valuable for population genetics inference. By creating a loss function that penalizes outputs against theoretical genetics principles, deep learning models in this domain could potentially yield more accurate and biologically meaningful results, ensuring that the inferred genetic parameters align with known genetic principles and constraints. One of the key challenges in applying this

concept to population genetics is defining the specific genetic principles and constraints to be incorporated into the loss function. Genetic models often involve complex factors such as mutation rates, genetic drift, selection pressures, and population dynamics. Designing a loss function that accurately captures these aspects while remaining computationally efficient would require careful consideration and domain expertise.

Incorporating domain knowledge through techniques like domain adaptation can help address issues of simulation misspecification, which is a common challenge in population genetics inference. By encouraging the model to learn domain-invariant features, such approaches can improve the robustness and accuracy of inference, particularly in scenarios where the simulated data may not fully capture the complexities of the real-world data. Overall, the integration of domain-specific knowledge into deep learning models for population genetics inference would be challenging as well as an exciting area of and the development of more reliable and interpretable.

Summary:

The emergence of convolutional neural network (CNN) based methodologies and other deep learning approaches in population genetics marks a new beginning in the field of evolutionary biology. Notably, recent applications of Graph Neural Networks (GNNs) have facilitated the inference of ancestral recombination graphs (ARG) from genomic data, enabling the capture of nonlinear and long-term dependencies, inherent in genetic sequences (Hejase et al., 2022). Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, have been used to extract representations of genomic sequences, providing valuable input features for downstream population genetic tasks such as selection scans or demographic inference (Alharbi & Rashid, 2022). There also exists significant potential in integrating attention-based transformer neural networks (Vaswani et al., 2023) with CNNs in population genetics research. CNNs excel in feature extraction, while transformers can enhance the focus on important genetic regions in the input, thereby synergistically improving inference accuracy. As the volume of available data continues to burgeon and neural architecture undergoes constant evolution, the possibilities within this domain remain boundless.

It is important to recognize that while deep learning-based inference holds great promise, it should not serve as an alternative of theoretical population genetics. It should be viewed as a complementary tool, enriching our understanding of evolutionary dynamics. Our study shows the promising future of CNN-based deep learning models, showcasing their comparative accuracy to traditional summary statistic-based methods. Its wouldn't be wrong to say that CNNs and deep learning methodologies are forging a new path in evolutionary history inference and parameter estimation, catalyzing paradigm shifts in population genetics research.

Meanwhile, Theoretical Physicists engage in lively debate among themselves about the nature and existence of time and its linear nature (Arntzenius & Maudlin, 2002), while the field of evolutionary history and population genetics persist and continue to enrich our toolkit for population history inference with innovative methodologies like CNN.

References:

Achuthan, S., Chatterjee, R., Kotnala, S., Mohanty, A., Bhattacharya, S., Salgia, R., & Kulkarni, P. (2022). Leveraging deep learning algorithms for synthetic data generation to design and analyze biological networks. *Journal of Biosciences*, *47*(3), 43. https://doi.org/10.1007/s12038-022-00278-3

Adam, S. P., Alexandropoulos, S.-A. N., Pardalos, P. M., & Vrahatis, M. N. (2019). No Free Lunch Theorem: A Review. In I. C. Demetriou & P. M. Pardalos (Eds.), *Approximation and Optimization: Algorithms, Complexity and Applications* (pp. 57–82). Springer International Publishing. https://doi.org/10.1007/978-3-030-12767-1_5

Agarap, A. F. (2019). *Deep Learning using Rectified Linear Units (ReLU)* (arXiv:1803.08375). arXiv. https://doi.org/10.48550/arXiv.1803.08375

Agha, R., Gross, A., Rohrlack, T., & Wolinska, J. (2018). Adaptation of a Chytrid Parasite to Its Cyanobacterial Host Is Hampered by Host Intraspecific Diversity. *Frontiers in Microbiology*, *9*. https://doi.org/10.3389/fmicb.2018.00921

Alharbi, W. S., & Rashid, M. (2022). A review of deep learning applications in human genomics using next-generation sequencing data. *Human Genomics*, *16*(1), 26. https://doi.org/10.1186/s40246-022-00396-x

Anderson, C. N. K., Ramakrishnan, U., Chan, Y. L., & Hadly, E. A. (2005). Serial SimCoal: A population genetics model for data from multiple populations and points in time. *Bioinformatics*, *21*(8), 1733–1734. https://doi.org/10.1093/bioinformatics/bti154

Ansari, S. (2023). Deep Learning and Artificial Neural Networks. In S. Ansari (Ed.), *Building Computer Vision Applications Using Artificial Neural Networks: With Examples in OpenCV and TensorFlow with Python* (pp. 169–260). Apress. https://doi.org/10.1007/978-1-4842-9866-4_5

Arnab, S. P., Amin, M. R., & DeGiorgio, M. (2023). Uncovering Footprints of Natural Selection Through Spectral Analysis of Genomic Summary Statistics. *Molecular Biology and Evolution*, *40*(7), msad157. https://doi.org/10.1093/molbev/msad157

Arntzenius, F., & Maudlin, T. (2002). Time Travel and Modern Physics. *Royal Institute of Philosophy Supplements*, *50*, 169–200. https://doi.org/10.1017/S1358246100010560

Aronne, G. (2017). Identification of Bottlenecks in the Plant Life Cycle for Sustainable Conservation of Rare and Endangered Species. *Frontiers in Ecology and Evolution*, *5*. https://doi.org/10.3389/fevo.2017.00076

Aurélien Géron. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition* (2nd ed.). O'Reilly Media, Inc.

Baek, J., Lee, B., Kwon, S., & Yoon, S. (2018). LncRNAnet: Long non-coding RNA identification using deep learning. *Bioinformatics*, *34*(22), 3889–3897. https://doi.org/10.1093/bioinformatics/bty418

Baldi, P., & Sadowski, P. J. (2013). Understanding Dropout. *Advances in Neural Information Processing Systems*, *26*. https://proceedings.neurips.cc/paper/2013/hash/71f6278d140af599e06ad9bf1ba03cb0-Abstract.html

Beerli, P. (2006). Comparison of Bayesian and maximum-likelihood inference of population genetic parameters. *Bioinformatics*, *22*(3), 341–345. https://doi.org/10.1093/bioinformatics/bti803

Beerli, P., & Felsenstein, J. (1999). Maximum-Likelihood Estimation of Migration Rates and Effective Population Numbers in Two Populations Using a Coalescent Approach. *Genetics*, *152*(2), 763–773. https://doi.org/10.1093/genetics/152.2.763

Benjamin C. Haller. (2016). *Eidos: A Simple Scripting Language*. http://benhaller.com/slim/Eidos_Manual.pdf

Benjamin C. Haller & Philipp W. Messer. (2016). *SLiM: An Evolutionary Simulation Framework*. http://benhaller.com/slim/SLiM_Manual.pdf

Ben-Kiki, O., & Evans, C. (2009). *YAML Ain't Markup Language (YAML™) Version 1.2*.

Besag, J. (1974). Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, *36*(2), 192–225. https://doi.org/10.1111/j.2517-6161.1974.tb00999.x

Bunnefeld, L., Frantz, L. A. F., & Lohse, K. (2015). Inferring Bottlenecks from Genome-Wide Samples of Short Sequence Blocks. *Genetics*, *201*(3), 1157–1169. https://doi.org/10.1534/genetics.115.179861

Butlin, R. K. (2010). Population genomics and speciation. *Genetica*, *138*(4), 409–418. https://doi.org/10.1007/s10709-008-9321-3

Cai, S., Mao, Z., Wang, Z., Yin, M., & Karniadakis, G. E. (2021). Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, *37*(12), 1727–1738. https://doi.org/10.1007/s10409-021-01148-1

Caspermeyer, J. (2019). Sea Otters Shown to Have Low Genetic Diversity. *Molecular Biology and Evolution*, *36*(12), 2926–2927. https://doi.org/10.1093/molbev/msz165

Cavalli-Sforza, L. L., & Edwards, A. W. F. (1967). Phylogenetic analysis. Models and estimation procedures. *American Journal of Human Genetics*, *19*(3 Pt 1), 233–257.

Chen, M., Pan, D., Ren, H., Fu, J., Li, J., Su, G., Wang, A., Jiang, L., Zhang, Q., & Liu, J.-F. (2016). Identification of selective sweeps reveals divergent selection between Chinese Holstein and Simmental cattle populations. *Genetics Selection Evolution*, *48*(1), 76. https://doi.org/10.1186/s12711-016-0254-5

Cho, J., Lee, K., Shin, E., Choy, G., & Do, S. (2016). *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?* (arXiv:1511.06348). arXiv. https://doi.org/10.48550/arXiv.1511.06348

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., & de Hoon, M. J. L. (2009). Biopython: Freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, *25*(11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163

Cortés, A. J. (2017). Local Scale Genetic Diversity and its Role in Coping with Changing Climate. In *Genetic Diversity*. IntechOpen. https://doi.org/10.5772/67166

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Crowley, J. L. (2023). Convolutional Neural Networks. In M. Chetouani, V. Dignum, P. Lukowicz, & C. Sierra (Eds.), *Human-Centered Artificial Intelligence: Advanced Lectures* (pp. 67–80). Springer International Publishing. https://doi.org/10.1007/978-3-031-24349-3_5

Deep Learning: History and State-of-the-Arts. (2020). In *The Development of Deep Learning Technologies: Research on the Development of Electronic Information Engineering Technology in China* (pp. 1–11). Springer. https://doi.org/10.1007/978-981-15-4584-9_1

Felsenstein, J. (2001). Taking Variation of Evolutionary Rates Between Sites into Account in Inferring Phylogenies. *Journal of Molecular Evolution*, *53*(4), 447–455. https://doi.org/10.1007/s002390010234

Ferdosi, M. H., Henshall, J., & Tier, B. (2016). Study of the optimum haplotype length to build genomic relationship matrices. *Genetics Selection Evolution*, *48*(1), 75. https://doi.org/10.1186/s12711-016-0253-6

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, *222*(594–604), 309–368. https://doi.org/10.1098/rsta.1922.0009

Flagel, L., Brandvain, Y., & Schrider, D. R. (2019). The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference. *Molecular Biology and Evolution*, *36*(2), 220–238. https://doi.org/10.1093/molbev/msy224

Galtier, N., Depaulis, F., & Barton, N. H. (2000). Detecting Bottlenecks and Selective Sweeps From DNA Sequence Polymorphism. *Genetics*, *155*(2), 981–987. https://doi.org/10.1093/genetics/155.2.981

Garrick, R. C. (2023). Genetic signatures of lineage fusion closely resemble population decline. *Ecology and Evolution*, *13*(11), e10725. https://doi.org/10.1002/ece3.10725

Garrigan, D. (2009). Composite likelihood estimation of demographic parameters. *BMC Genetics*, *10*, 72. https://doi.org/10.1186/1471-2156-10-72

Gholamalinezhad, H., & Khosravi, H. (2020). *Pooling Methods in Deep Neural Networks, a Review* (arXiv:2009.07485). arXiv. https://doi.org/10.48550/arXiv.2009.07485

Goldstein, T., Studer, C., & Baraniuk, R. (2016). *A Field Guide to Forward-Backward Splitting with a FASTA Implementation* (arXiv:1411.3406). arXiv. https://doi.org/10.48550/arXiv.1411.3406

Griffiths, R. C., & Tavaré, S. (1994). Sampling theory for neutral alleles in a varying environment. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *344*(1310), 403–410. https://doi.org/10.1098/rstb.1994.0079

Grinberg, M. (2018). *flask web development: Developing web applications with python*. O'Reilly Media, Inc.

Gutenkunst, R. N., Hernandez, R. D., Williamson, S. H., & Bustamante, C. D. (2009). Inferring the Joint Demographic History of Multiple Populations from Multidimensional SNP Frequency Data. *PLOS Genetics*, *5*(10), e1000695. https://doi.org/10.1371/journal.pgen.1000695

Hagenblad, J., Hülskötter, J., Acharya, K. P., Brunet, J., Chabrerie, O., Cousins, S. A. O., Dar, P. A., Diekmann, M., De Frenne, P., Hermy, M., Jamoneau, A., Kolb, A., Lemke, I., Plue, J., Reshi, Z. A., & Graae, B. J. (2015). Low genetic diversity despite multiple introductions of the invasive plant species Impatiens glandulifera in Europe. *BMC Genetics*, *16*(1), 103. https://doi.org/10.1186/s12863-015-0242-8

Haller, B. C., Galloway, J., Kelleher, J., Messer, P. W., & Ralph, P. L. (2019). Tree-sequence recording in SLiM opens new horizons for forward-time simulation of whole genomes. *Molecular Ecology Resources*, *19*(2), 552–566. https://doi.org/10.1111/1755-0998.12968

Haller, B. C., & Messer, P. W. (2019). Evolutionary Modeling in SLiM 3 for Beginners. *Molecular Biology and Evolution*, *36*(5), 1101–1109. https://doi.org/10.1093/molbev/msy237

Haller, B. C., & Messer, P. W. (2023). SLiM 4: Multispecies Eco-Evolutionary Modeling. *The American Naturalist*, *201*(5), E127–E139. https://doi.org/10.1086/723601

Hanin, B. (2019). Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations. *Mathematics*, *7*(10), Article 10. https://doi.org/10.3390/math7100992

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Harris, R. B., & Jensen, J. D. (2020). Considering Genomic Scans for Selection as Coalescent Model Choice. *Genome Biology and Evolution*, *12*(6), 871–877. https://doi.org/10.1093/gbe/evaa093

Hejase, H. A., Dukler, N., & Siepel, A. (2020). From summary statistics to gene trees: Methods for inferring positive selection. *Trends in Genetics : TIG*, *36*(4), 243–258. https://doi.org/10.1016/j.tig.2019.12.008

Hejase, H. A., Mo, Z., Campagna, L., & Siepel, A. (2022). A Deep-Learning Approach for Inference of Selective Sweeps from the Ancestral Recombination Graph. *Molecular Biology and Evolution*, *39*(1), msab332. https://doi.org/10.1093/molbev/msab332

Hemalatha, J., Roseline, S. A., Geetha, S., Kadry, S., & Damaševičius, R. (2021). An Efficient DenseNet-Based Deep Learning Model for Malware Detection. *Entropy*, *23*(3), 344. https://doi.org/10.3390/e23030344

Hickerson, M. J., Stahl, E., & Takebayashi, N. (2007). msBayes: Pipeline for testing comparative phylogeographic histories using hierarchical approximate Bayesian computation. *BMC Bioinformatics*, *8*(1), 268. https://doi.org/10.1186/1471-2105-8-268

Hobolth, A., & Wiuf, C. (2009). The genealogy, site frequency spectrum and ages of two nested mutant alleles. *Theoretical Population Biology*, *75*(4), 260–265. https://doi.org/10.1016/j.tpb.2009.02.001

Hogg, C. J. (2024). Translating genomic advances into biodiversity conservation. *Nature Reviews Genetics*, *25*(5), 362–373. https://doi.org/10.1038/s41576-023-00671-0

Hohenlohe, P. A., Funk, W. C., & Rajora, O. P. (2021). Population genomics for wildlife conservation and management. *Molecular Ecology*, *30*(1), 62–82. https://doi.org/10.1111/mec.15720

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574–591.

Hudson, R. R. (2002). Generating samples under a Wright–Fisher neutral model of genetic variation. *Bioinformatics*, *18*(2), 337–338. https://doi.org/10.1093/bioinformatics/18.2.337

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Iman, M., Arabnia, H. R., & Rasheed, K. (2023). A Review of Deep Transfer Learning and Recent Advancements. *Technologies*, *11*(2), Article 2. https://doi.org/10.3390/technologies11020040

Ingman, M., Kaessmann, H., Pääbo, S., & Gyllensten, U. (2000). Mitochondrial genome variation and the origin of modern humans. *Nature*, *408*(6813), 708–713. https://doi.org/10.1038/35047064

Ivana Cvijović, Michael M Desai, & Benjamin H Good. (n.d.). *The Effect of Strong Purifying Selection on Genetic Diversity—PubMed*. Retrieved April 22, 2024, from https://pubmed.ncbi.nlm.nih.gov/29844134/

Jeffrey A. Clark. (2024). *Pillow (PIL Fork) Documentation*. https://readthedocs.org/projects/pillow/downloads/pdf/latest/

Jensen, J. D., Kim, Y., DuMont, V. B., Aquadro, C. F., & Bustamante, C. D. (2005). Distinguishing Between Selective Sweeps and Demography Using DNA Polymorphism Data. *Genetics*, *170*(3), 1401–1410. https://doi.org/10.1534/genetics.104.038224

Jo, T., Nho, K., Bice, P., Saykin, A. J., & For The Alzheimer's Disease Neuroimaging Initiative. (2022). Deep learning-based identification of genetic variants: Application to Alzheimer's disease classification. *Briefings in Bioinformatics*, *23*(2), bbac022. https://doi.org/10.1093/bib/bbac022

Johri, P., Aquadro, C. F., Beaumont, M., Charlesworth, B., Excoffier, L., Eyre-Walker, A., Keightley, P. D., Lynch, M., McVean, G., Payseur, B. A., Pfeifer, S. P., Stephan, W., & Jensen, J. D. (2022). Recommendations for improving statistical inference in population genomics. *PLOS Biology*, *20*(5), e3001669. https://doi.org/10.1371/journal.pbio.3001669

Jones, F. C., Grabherr, M. G., Chan, Y. F., Russell, P., Mauceli, E., Johnson, J., Swofford, R., Pirun, M., Zody, M. C., White, S., Birney, E., Searle, S., Schmutz, J., Grimwood, J., Dickson, M. C., Myers, R. M., Miller, C. T., Summers, B. R., Knecht, A. K., … Kingsley, D. M. (2012). The genomic basis of adaptive evolution in threespine sticklebacks. *Nature*, *484*(7392), 55–61. https://doi.org/10.1038/nature10944

Kelleher, J., Thornton, K. R., Ashander, J., & Ralph, P. L. (2018). Efficient pedigree recording for fast population genetics simulation. *PLOS Computational Biology*, *14*(11), e1006581. https://doi.org/10.1371/journal.pcbi.1006581

Kim, Y., & Stephan, W. (2002). Detecting a local signature of genetic hitchhiking along a recombining chromosome. *Genetics*, *160*(2), 765–777. https://doi.org/10.1093/genetics/160.2.765

Kingma, D. P., & Ba, J. (2017). *Adam: A Method for Stochastic Optimization* (arXiv:1412.6980). arXiv. https://doi.org/10.48550/arXiv.1412.6980

Kingman, J. F. C. (1982). The coalescent. *Stochastic Processes and Their Applications*, *13*(3), 235–248. https://doi.org/10.1016/0304-4149(82)90011-4

Koropoulis, A., Alachiotis, N., & Pavlidis, P. (2020). Detecting Positive Selection in Populations Using Genetic Data. In J. Y. Dutheil (Ed.), *Statistical Population Genomics* (pp. 87–123). Springer US. https://doi.org/10.1007/978-1-0716-0199-0_5

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, *25*. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Kuhner, M. K. (2006). LAMARC 2.0: Maximum likelihood and Bayesian estimation of population parameters. *Bioinformatics*, *22*(6), 768–770. https://doi.org/10.1093/bioinformatics/btk051

Lai, Y.-T., Yeung, C. K. L., Omland, K. E., Pang, E.-L., Hao, Y., Liao, B.-Y., Cao, H.-F., Zhang, B.-W., Yeh, C.-F., Hung, C.-M., Hung, H.-Y., Yang, M.-Y., Liang, W., Hsu, Y.-C., Yao, C.-T., Dong, L., Lin, K., & Li, S.-H. (2019). Standing genetic variation as the predominant source for adaptation of a songbird. *Proceedings of the National Academy of Sciences*, *116*(6), 2152–2157. https://doi.org/10.1073/pnas.1813597116

Laval, G., & Excoffier, L. (2004). SIMCOAL 2.0: A program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history. *Bioinformatics (Oxford, England)*, *20*(15), 2485–2487. https://doi.org/10.1093/bioinformatics/bth264

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Lewontin, R. C., & Hubby, J. L. (1966). A Molecular Approach to the Study of Genic Heterozygosity in Natural Populations. II. Amount of Variation and Degree of Heterozygosity in Natural Populations of DROSOPHILA PSEUDOOBSCURA. *Genetics*, *54*(2), 595–609.

Li, J. Z., Absher, D. M., Tang, H., Southwick, A. M., Casto, A. M., Ramachandran, S., Cann, H. M., Barsh, G. S., Feldman, M., Cavalli-Sforza, L. L., & Myers, R. M. (2008). Worldwide Human Relationships Inferred from Genome-Wide Patterns of Variation. *Science*, *319*(5866), 1100–1104. https://doi.org/10.1126/science.1153717

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., & Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, *21*(6), 335–346. https://doi.org/10.1038/s41583-020-0277-3

Lozada-Soto, E. A., Maltecca, C., Lu, D., Miller, S., Cole, J. B., & Tiezzi, F. (2021). Trends in genetic diversity and the effect of inbreeding in American Angus cattle under genomic selection. *Genetics, Selection, Evolution : GSE*, *53*, 50. https://doi.org/10.1186/s12711-021-00644-z

Luikart, G., Allendorf, F., Cornuet, J.-M., & Sherwin, W. (1998). Distortion of allele frequency distributions provides a test for recent population bottlenecks. *Journal of Heredity*, *89*(3), 238–247. https://doi.org/10.1093/jhered/89.3.238

MacLeod, I. M., Hayes, B. J., & Goddard, M. E. (2014). The Effects of Demography and Long-Term Selection on the Accuracy of Genomic Prediction with Sequence Data. *Genetics*, *198*(4), 1671–1684. https://doi.org/10.1534/genetics.114.168344

Mao, A., Mohri, M., & Zhong, Y. (2023). Cross-Entropy Loss Functions: Theoretical Analysis and Applications. *Proceedings of the 40th International Conference on Machine Learning*, 23803–23828. https://proceedings.mlr.press/v202/mao23b.html

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/

Metzler, D. (2011). Jaatha: A fast composite-likelihood approach to estimate demographic parameters. *Molecular Ecology*, *20*(13), 2709–2723. https://doi.org/10.1111/j.1365-294X.2011.05131.x

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., & Leisch, F. (2020, October 14). *Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien [R package e1071 version 1.7-4]*. https://www.semanticscholar.org/paper/Misc-Functions-of-the-Department-of-Statistics%2C-TU-Meyer-Dimitriadou/571964e71aa822b950c6c6dfed52fa63694c05a9

Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-Tinch, C. H., Sochat, V., Forster, J., Lee, S., Twardziok, S. O., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., & Köster, J. (2021). *Sustainable data analysis with Snakemake* (10:33). F1000Research. https://doi.org/10.12688/f1000research.29032.2

Moore, G. (1965). *Cramming More Components onto Integrated Circuits (1965)*. https://doi.org/10.7551/mitpress/12274.003.0027

Mouton, C., Myburgh, J. C., & Davel, M. H. (2020). Stride and Translation Invariance in CNNs. In A. Gerber (Ed.), *Artificial Intelligence Research* (pp. 267–281). Springer International Publishing. https://doi.org/10.1007/978-3-030-66151-9_17

Mughal, M. R., & DeGiorgio, M. (2019). Localizing and Classifying Adaptive Targets with Trend Filtered Regression. *Molecular Biology and Evolution*, *36*(2), 252–270. https://doi.org/10.1093/molbev/msy205

Mühlenbein, H. (2009). Computational Intelligence: The Legacy of Alan Turing and John von Neumann. In C. L. Mumford & L. C. Jain (Eds.), *Computational Intelligence: Collaboration, Fusion and Emergence* (pp. 23–43). Springer. https://doi.org/10.1007/978-3-642-01799-5_2

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, *2*(1), 1. https://doi.org/10.1186/s40537-014-0007-7

Nei, M. (1986). Definition and Estimation of Fixation Indices. *Evolution*, *40*(3), 643–645. https://doi.org/10.2307/2408586

Newton, D., Yousefian, F., & Pasupathy, R. (2018). Stochastic Gradient Descent: Recent Trends. In *Recent Advances in Optimization and Modeling of Contemporary Problems* (pp. 193–220). INFORMS. https://doi.org/10.1287/educ.2018.0191

Neyman, J. (1971). MOLECULAR STUDIES OF EVOLUTION: A SOURCE OF NOVEL STATISTICAL PROBLEMS*. In S. S. Gupta & J. Yackel (Eds.), *Statistical Decision Theory and Related Topics* (pp. 1–27). Academic Press. https://doi.org/10.1016/B978-0-12-307550-5.50005-8

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable Parallel Programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*, *6*(2), 40–53. https://doi.org/10.1145/1365490.1365500

Nielsen, R. (2000). Estimation of Population Parameters and Recombination Rates From Single Nucleotide Polymorphisms. *Genetics*, *154*(2), 931–942. https://doi.org/10.1093/genetics/154.2.931

Noor, M. A. F., & Feder, J. L. (2006). Speciation genetics: Evolving approaches. *Nature Reviews Genetics*, *7*(11), 851–861. https://doi.org/10.1038/nrg1968

Nordborg, M., & Krone, S. M. (2002). Separation of Time Scales and Convergence to the Coalescent in Structured Populations. In M. Slatkin & M. Veuille (Eds.), *Modern developments in theoretical population genetics: The legacy of Gustave Malecot* (p. 0). Oxford University Press. https://doi.org/10.1093/oso/9780198599623.003.0012

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). *Automatic differentiation in PyTorch*. https://openreview.net/forum?id=BJJsrmfCZ

Pearson, K. (1901). *LIII. On lines and planes of closest fit to systems of points in space*. https://doi.org/10.1080/14786440109462720

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., & Liao, Q. (2017). Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, *14*(5), 503–519. https://doi.org/10.1007/s11633-017-1054-2

Poggio, T., Torre, V., & Koch, C. (1987). Computational vision and regularization theory. In M. A. Fischler & O. Firschein (Eds.), *Readings in Computer Vision* (pp. 638–643). Morgan Kaufmann. https://doi.org/10.1016/B978-0-08-051581-6.50061-1

Pool, J. E., Hellmann, I., Jensen, J. D., & Nielsen, R. (2010). Population genetic inference from genomic sequence variation. *Genome Research*, *20*(3), 291–300. https://doi.org/10.1101/gr.079509.108

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., & Feldman, M. W. (1999). Population growth of human Y chromosomes: A study of Y chromosome microsatellites. *Molecular Biology and Evolution*, *16*(12), 1791–1798. https://doi.org/10.1093/oxfordjournals.molbev.a026091

Radiuk, P. M. (2017). Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. *Information Technology and Management Science*, *20*(1), Article 1.

Ragsdale, A. P., Weaver, T. D., Atkinson, E. G., Hoal, E. G., Möller, M., Henn, B. M., & Gravel, S. (2023). A weakly structured stem for human origins in Africa. *Nature*, *617*(7962), 755–763. https://doi.org/10.1038/s41586-023-06055-y

Raja Santhi, A., & Muthuswamy, P. (2023). Industry 5.0 or industry 4.0S? Introduction to industry 4.0 and a peek into the prospective industry 5.0 technologies. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, *17*(2), 947–979. https://doi.org/10.1007/s12008-023-01217-8

Reid, N. M., Proestou, D. A., Clark, B. W., Warren, W. C., Colbourne, J. K., Shaw, J. R., Karchner, S. I., Hahn, M. E., Nacci, D., Oleksiak, M. F., Crawford, D. L., & Whitehead, A. (2016). The genomic landscape of rapid repeated evolutionary adaptation to toxic pollution in wild fish. *Science*, *354*(6317), 1305–1308. https://doi.org/10.1126/science.aah4993

Ripplinger, J., & Sullivan, J. (2008). Does Choice in Model Selection Affect Maximum Likelihood Analysis? *Systematic Biology*, *57*(1), 76–85.

Rodrigues, M. F., Kern, A. D., & Ralph, P. L. (2023). Shared evolutionary processes shape landscapes of genomic variation in the great apes. *bioRxiv*, 2023.02.07.527547. https://doi.org/10.1101/2023.02.07.527547

Rojas, R. (1996). The Backpropagation Algorithm. In R. Rojas (Ed.), *Neural Networks: A Systematic Introduction* (pp. 149–182). Springer. https://doi.org/10.1007/978-3-642-61068-4_7

Rutherford, S., Rossetto, M., Bragg, J. G., McPherson, H., Benson, D., Bonser, S. P., & Wilson, P. G. (2018). Speciation in the presence of gene flow: Population genomics of closely related and diverging Eucalyptus species. *Heredity*, *121*(2), 126–141. https://doi.org/10.1038/s41437-018-0073-2

Sadegh, M., & Vrugt, J. A. (2014). Approximate Bayesian Computation using Markov Chain Monte Carlo simulation: DREAM(ABC). *Water Resources Research*, *50*(8), 6767–6787. https://doi.org/10.1002/2014WR015386

Saeb, A. T. M., & Al-Naqeb, D. (2016). The Impact of Evolutionary Driving Forces on Human Complex Diseases: A Population Genetics Approach. *Scientifica*, *2016*, 2079704. https://doi.org/10.1155/2016/2079704

Sánchez, L., & Woolliams, J. A. (2004). Impact of Nonrandom Mating on Genetic Variance and Gene Flow in Populations With Mass Selection. *Genetics*, *166*(1), 527–535. https://doi.org/10.1534/genetics.166.1.527

Sanger, F., Nicklen, S., & Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, *74*(12), 5463–5467. https://doi.org/10.1073/pnas.74.12.5463

Schaffer, C. (1993). Selecting a classification method by cross-validation. *Machine Learning*, *13*(1), 135–143. https://doi.org/10.1007/BF00993106

Schlamp, F., van der Made, J., Stambler, R., Chesebrough, L., Boyko, A. R., & Messer, P. W. (2016). Evaluating the performance of selection scans to detect selective sweeps in domestic dogs. *Molecular Ecology*, *25*(1), 342–356. https://doi.org/10.1111/mec.13485

Schrider, D. R., & Kern, A. D. (2016). S/HIC: Robust Identification of Soft and Hard Sweeps Using Machine Learning. *PLoS Genetics*, *12*(3), e1005928. https://doi.org/10.1371/journal.pgen.1005928

Shriver, M. D., Kennedy, G. C., Parra, E. J., Lawson, H. A., Sonpar, V., Huang, J., Akey, J. M., & Jones, K. W. (2004). The genomic distribution of population substructure in four populations using 8,525 autosomal SNPs. *Human Genomics*, *1*(4), 274. https://doi.org/10.1186/1479-7364-1-4-274

Sokolova, K., Chen, K. M., Hao, Y., Zhou, J., & Troyanskaya, O. G. (2024). *Deep Learning Sequence Models for Transcriptional Regulation*. https://doi.org/10.1146/annurev-genom-021623-024727

Sonsthagen, S. A., Wilson, R. E., & Underwood, J. G. (2017). Genetic implications of bottleneck effects of differing severities on genetic diversity in naturally recovering populations: An example from Hawaiian coot and Hawaiian gallinule. *Ecology and Evolution*, *7*(23), 9925–9934. https://doi.org/10.1002/ece3.3530

Stephan, W. (2019). Selective Sweeps. *Genetics*, *211*(1), 5–13. https://doi.org/10.1534/genetics.118.301319

Stigler, S. M. (1982). Thomas Bayes's Bayesian Inference. *Journal of the Royal Statistical Society: Series A (General)*, *145*(2), 250–258. https://doi.org/10.2307/2981538

Tajima, F. (1989). Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics*, *123*(3), 585–595. https://doi.org/10.1093/genetics/123.3.585

Team, R. (2014). R: A language and environment for statistical computing. *MSOR Connections*. https://www.semanticscholar.org/paper/R%3A-A-language-and-environment-for-statistical-Team/659408b243cec55de8d0a3bc51b81173007aa89b

Theissinger, K., Fernandes, C., Formenti, G., Bista, I., Berg, P. R., Bleidorn, C., Bombarely, A., Crottini, A., Gallo, G. R., Godoy, J. A., Jentoft, S., Malukiewicz, J., Mouton, A., Oomen, R. A., Paez, S., Palsbøll, P. J., Pampoulie, C., Ruiz-López, M. J., Secomandi, S., … Zammit, G. (2023). How genomics can help biodiversity conservation. *Trends in Genetics*, *39*(7), 545–559. https://doi.org/10.1016/j.tig.2023.01.005

Ünal, H. T., & Başçiftçi, F. (2022). Evolutionary design of neural network architectures: A review of three decades of research. *Artificial Intelligence Review*, *55*(3), 1723–1802. https://doi.org/10.1007/s10462-021-10049-5

van Dyk, D. A., & Meng, X.-L. (2001). The Art of Data Augmentation. *Journal of Computational and Graphical Statistics*, *10*(1), 1–50. https://doi.org/10.1198/10618600152418584

VanLiere, J. M., & Rosenberg, N. A. (2008). Mathematical properties of the r2 measure of linkage disequilibrium. *Theoretical Population Biology*, *74*(1), 130–137. https://doi.org/10.1016/j.tpb.2008.05.006

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (arXiv:1706.03762). arXiv. https://doi.org/10.48550/arXiv.1706.03762

Watterson, G. A. (1975). On the number of segregating sites in genetical models without recombination. *Theoretical Population Biology*, *7*(2), 256–276. https://doi.org/10.1016/0040-5809(75)90020-9

Wein, T., & Dagan, T. (2019). The Effect of Population Bottleneck Size and Selective Regime on Genetic Diversity and Evolvability in Bacteria. *Genome Biology and Evolution*, *11*(11), 3283–3290. https://doi.org/10.1093/gbe/evz243

Willi, Y., Fracassetti, M., Bachmann, O., & Van Buskirk, J. (2020). Demographic Processes Linked to Genetic Diversity and Positive Selection across a Species' Range. *Plant Communications*, *1*(6), 100111. https://doi.org/10.1016/j.xplc.2020.100111

Wong, P. C. (2021). Unsupervised Machine Learning. In A. A. von Davier, R. J. Mislevy, & J. Hao (Eds.), *Computational Psychometrics: New Methodologies for a New Generation of Digital Learning and Assessment: With Examples in R and Python* (pp. 173–193). Springer International Publishing. https://doi.org/10.1007/978-3-030-74394-9_10

Xie, R., Quitadamo, A., Cheng, J., & Shi, X. (2016). A predictive model of gene expression using a deep learning framework. *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 676–681. https://doi.org/10.1109/BIBM.2016.7822599

Xu, C., & Jackson, S. A. (2019). Machine learning and complex biological data. *Genome Biology*, *20*(1), 76. https://doi.org/10.1186/s13059-019-1689-0

Yang, B., Liu, F., Ren, C., Ouyang, Z., Xie, Z., Bo, X., & Shu, W. (2017). BiRen: Predicting enhancers with a deep-learning-based model using the DNA sequence alone. *Bioinformatics*, *33*(13), 1930–1936. https://doi.org/10.1093/bioinformatics/btx105

Mölder F, Jablonski KP, Letcher B *et al*. Sustainable data analysis with Snakemake [version 2; peer review: 2 approved]. F1000Research 2021, 10:33 (https://doi.org/10.12688/f1000research.29032.2)

Grinberg, M. (2018). Flask web development: developing web applications with python. " O'Reilly Media, Inc."

Acknowledgements

I am deeply thankful to my supervisor, Dr. Dirk Metzler, for his unwavering support and patience, always willing to assist me with even the silliest of questions. His guidance, time, and optimism have been invaluable throughout my journey. I am also grateful to my friends, especially Uthara and Pratik, for their support in times of need, whether it was finding housing or providing comfort during moments of self-doubt. My heartfelt appreciation goes to my parents, not only for their genetic contribution but also for instilling in me strong moral values. Special thanks to my sister, whose daily dose of Instagram reels kept my spirits high. I am also thankful to Dr. Philomena for her exceptional administrative and personal support. Lastly, I express my gratitude to the MEME administration, fellow members, and the Erasmus Program of the European Commission, as well as the European taxpayers, whose funding made my studies possible and dreams a reality.