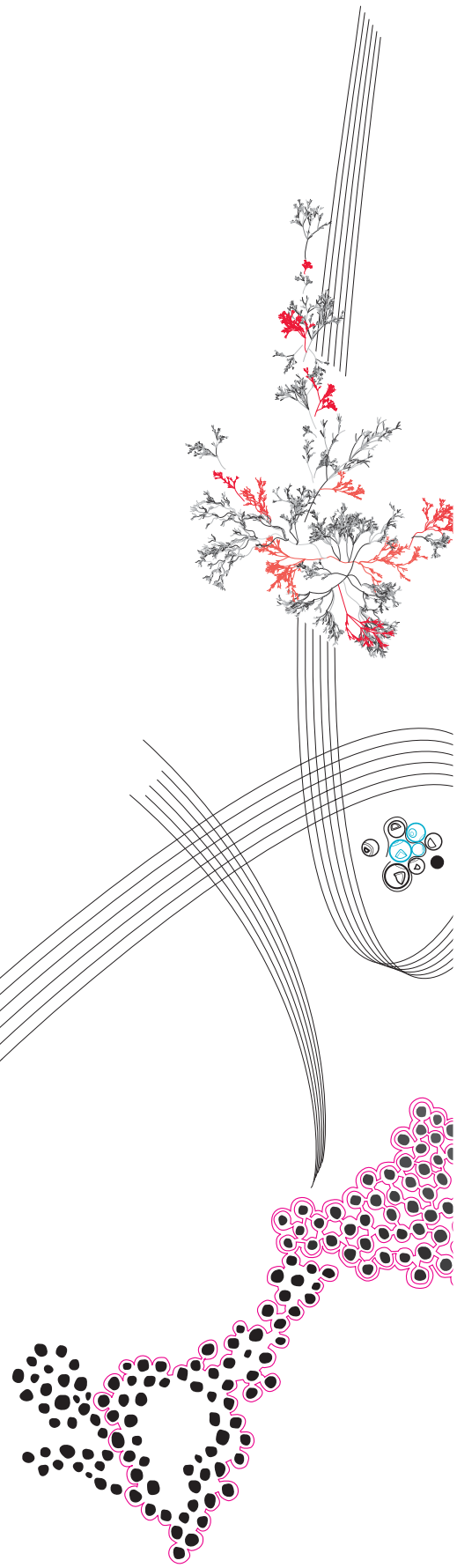


MSc Electrical Engineering
Dependable Integrated Systems

Robust and Scalable Selective Sweep Detection using Convolutional Neural Networks

Sjoerd van den Belt

Supervising committee:
dr. ir. N. Alachiotis
dr. N. Strisciuglio
dr.ing. K.H. Chen

June, 2024

Computer Architecture for Embedded Systems group
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Abstract

Localizing DNA mutations that have led to positive natural selection is important for understanding viruses and diseases, the development of drugs, and many other applications. Traditionally, statistical models processing single nucleotide polymorphisms (SNPs) have been applied to detect a distinct pattern in SNP data indicative of positive natural selection, called a selective sweep. These models can effectively and efficiently detect selective sweeps under simple evolutionary models but are prone to high false positive rates when confronted with data where confounding factors are present. Recently, machine learning-based methods have been demonstrated to be more robust to confounding factors. Using convolutional neural networks (CNNs) the classification accuracy of selective sweeps in the presence of confounding factors is improved, however, CNNs are computationally expensive compared to statistical methods, limiting the usefulness of these methods when applied to large datasets. In this thesis, FAST-NN is presented, which uses 1D convolutions to process allele frequencies and pairwise SNP distances. FAST-NN achieves a selective sweep classification accuracy on all tested datasets that outperforms or performs on par with the state-of-the-art, while decreasing execution time on both CPU and GPU. Previous work that focuses on using CNNs for selective sweep detection primarily evaluates classification performance and does not explicitly evaluate the performance and precision of models when applied for detection. In addition, CNN-based selective sweep detection methods apply a sliding window or grid over a genome to sample windows for classification. When performing fine-grained detection, the sampled windows overlap, and if each window is processed separately, this leads to repeated, redundant, computations. The FAST-NN model was designed considering the classification of separate genomic segments. In this thesis, the FASTER-NN model extends the FAST-NN model and takes advantage of wide input windows through a larger receptive field. FASTER-NN is specifically designed to optimize detection performance, and has an improved detection sensitivity compared to state-of-the-art models, while only processing allele frequencies and pairwise SNP distances. Moreover, by using dilated convolutions and optimizing data reuse, the execution time is nearly invariant to input width. FASTER-NN enables whole-genome scans at a considerably reduced execution time compared to other CNN-based detection methods, paving the way for accessible CNN-based selective sweep detection, without requiring expensive hardware such as a GPU. FASTER-NN has been extended to classify selective sweeps and recombination hotspots through a single model. This extended model demonstrates the effect of partially retaining information on linkage disequilibrium on recombination hotspot classification. By using grouped allele frequencies, the execution time of recombination hotspot classification through a CNN can be reduced, with a negligible effect on classification accuracy. Selective sweep detection through CNNs can be accelerated by using reconfigurable hardware. FPGAs are constrained by limited I/O bandwidth and hardware resources but can implement customized on-chip parallel pipelines that sustain high data rates. Through the compact data format of allele frequencies, and by using only 1D convolutions, the FAST-NN model requires limited I/O bandwidth and hardware resources. To this end, an 8-bit quantized version of the model is deployed on an FPGA in a fully-pipelined architecture, running at 90 MHz with an initiation interval of one clock cycle. This solution can densely scan all 22 human autosomes in 135 milliseconds, classifying one window of 128 SNP positions per clock cycle.

Contents

Introduction

Positive natural selection is an evolutionary phenomenon that drives the adaptive power of a species to its environment. Positive natural selection, or any non-random selection process, exists due to genetic diversity throughout a population [50]. Neutral genetic diversity does not affect any physical characteristics or traits of a species. On the other hand, adaptive genetic diversity leads to changes in the attributes of a species, which can improve its fitness. This, in turn can lead to an increased chance of survival and reproduction for the carriers of the mutated gene. Positive selection is a purely statistical consequence inherent to populations where adaptive genetic diversity is present. When a subgroup of a population has an elevated reproduction rate, due to genetic diversity, the part of the population carrying the favourable genes has an increased chance of spreading this gene to new generations [23]. One can imply that nature *selects* the favourable genes to be propagated to consecutive generations, hence the term positive natural selection.

Determining which genetic mutations lead to positive natural selection is important for understanding the evolution of a species. This can lead to understanding mutations that have enabled resilience to disease in animals, which can in turn infer the effect of a virus on humans [16]. Detecting positive selection can improve designing effective drug treatment [7]. Moreover, natural selection is also relevant in research towards viruses, such as SARS-CoV-2 which was responsible for the COVID-19 pandemic, in this virus positive selection is used to identify the mutations that led to the adaption of the virus to human transmission [29].

Over the course of multiple generations, a favourable gene that leads to positive natural selection is spread to an increasing proportion of the population. Given sufficient time, and under certain conditions, the entire population will adopt the favourable gene [23]. When genomes are sampled from present-day populations where positive natural selection has driven its evolution over past generations, the positive selection leaves specific patterns in the genetic encoding [53]. The pattern left by positive selection is referred to as a selective sweep. Uncovering these patterns is the key challenge to identifying the positive natural selection that previous generations have undergone. This is not a trivial task, positive natural selection is not the only effect that leaves patterns in the genome. Other population effects, such as bottlenecks in population size, and genetic effects, such as genomic regions with increased recombination rates, can leave patterns similar to those left by positive natural selection [1].

Detecting specific patterns in genetic data requires an appropriate encoding of this data that can be processed by computer algorithms. The genome is made up of organic molecules called nucleotides. In DNA these nucleotides are arranged into long sequences of nucleotide pairs, also called basepairs. The information in the DNA is encoded in the sequences formed by the basepairs. When identifying patterns which arise because of mutations in the DNA, it is not necessary to process all basepairs, since many basepairs do not demonstrate any genetic variation throughout a population. To reduce the data to only include informative basepairs, given a certain set of samples from a population, only the basepairs where at least one sample has a mutation are kept. These basepairs are named single nucleotide polymorphisms (SNPs). Single nucleotide refers to the mutation affecting a single nucleotide basepair, and polymorphisms refer to the existence of different basepairs at this position.

Identifying specific patterns in SNP data is a common method for studying population genomics. This includes the identification of selective sweeps, and identifying regions in the genome with higher levels of recombination, called recombination hotspots [13]. Statistical tools exist which can efficiently process SNP data of multiple samples in a population to identify patterns indicative of a selective sweep. [54][49][1]. The patterns these tools use to detect regions of interest are based on our limited understanding of the effect that evolution has on the genome. Recently, machine-learning methods, and in particular neural networks, have proven to outperform statistical methods in detecting genomic regions of interest [20][56][43][68][69]. These methods do not rely

on our fundamental understanding of the underlying mechanisms driving genetic mutations, but rather these models learn the patterns embedded in these mutations from training data.

SNP data from multiple samples can be arranged into a matrix of SNPs, and it can be assumed that SNPs in close proximity, on the same genome, have a higher correlation than distant SNPs. This naturally points towards convolutional neural networks (CNNs) as the model of choice for analyzing SNPs [38]. CNNs have an inductive bias for identifying the local spatial arrangement of data, making these networks excellent models for SNP processing. CNNs have widely been employed for SNP analysis, and have proven accurate and more robust to false positives when analyzing patterns in the presence of confounding effects [68][69]. Using machine learning for this task does come with some disadvantages, the execution time that CNNs require is greater than the execution time of summary statistics. Genetic datasets are growing in size due to advances in DNA alignment [51], increasing the input size to these models. The processing power required by CNN models to perform inference over genetic data increases as these datasets expand. These limitations in the scalability of CNNs reduce the effectiveness and adaption of CNNs as a practical tool to analyze SNP data.

When detecting selective sweeps on a genome, the SNPs on the genome are scanned for patterns. Scanning an entire genome can be done by sliding a window across the genome and analyzing each window separately. Optimized tools that employ summary statistics to detect positive selection perform such sliding window approaches [1][48]. When increasing the width of the window used to slide over the genome, the processing time for each sampled window increases. Due to this, using wide windows for fine-grained scans is inefficient, even when using summary statistics to analyze SNP data. When CNN models are implemented for this task, the efficiency is even lower due to the higher execution time per window. Most research towards using CNNs to analyze SNP data focuses on classifying SNP data, rather than scanning extensive sequences of SNPs [20][56][43][68]. If the execution time to scan genomes using CNN models is impractical, these tools will not be widely adopted, despite their effectiveness.

The field of bio-informatics often deals with the need to process large amounts of data. To reduce the extensive processing times imposed by this, hardware acceleration using programmable hardware has been explored in various domains, including phylogenetics and population genetics [4][42][64][11][6]. In the context of detecting positive natural selection, processing SNPs through statistical models has been accelerated using Field Programmable Gate Arrays (FPGAs) [5][6][15]. Using FPGAs, dedicated hardware architectures are implemented, which outperform the throughput of general-purpose hardware. Whilst FPGA-based hardware acceleration for CNNs is a popular domain of research [57][63] [28], no previous work has implemented a CNN model for SNP processing on an FPGA.

This thesis tackles the challenges accompanying the practical use of CNNs for detecting positive natural selection. Solving these challenges will allow future research to efficiently and reliably detect regions of interest on genomes, which is applicable in various research domains, such as drug and virus research. The research throughout this thesis attempts to answer the following research question: *How can CNNs be deployed to efficiently, accurately and robustly detect genomic regions affected by positive natural selection?* To answer this question, four sub-questions are posed.

1. Genomic datasets are extensive and CNN-based models do not scale as well to the large datasets as summary statistics-based methods, making CNNs less efficient. This raises the following question. How can CNN-based classification models for positive selection be scalable to a large sample size without compromising effectiveness?

2. Often, classification accuracy is used as the sole performance metric of CNN-based methods. However, the practical application of these models is detection when scanning a genome. To address this discrepancy, the following question is addressed. How can CNN-based models be optimized, in terms of efficiency and effectiveness, for detecting regions affected by positive selection when scanning a genome?

3. Identifying the presence of other genomic effects at the position of a potential selective sweep can be used to tune the parameters of the detection framework. Currently, models that perform classification using SNP data focus on classifying a single genomic effect using a dedicated model. This raises the question, can CNN-based models be extended to detect multiple genomic effects, without repeating inference, and does detecting confounding factors improve the ability to detect selective sweeps?

4. The high parallelism of an FPGA implementation can increase the throughput of a selective sweep detection algorithm, but requires quantization which can affect the model performance.

By increasing throughput through hardware acceleration, the execution time to process large datasets can be reduced, allowing for fine-grained detection without the disadvantage of long execution times. To explore this, the following question is posed. Can FPGAs be used to accelerate a quantized CNN model to detect positive selection on a genome without reducing the effectiveness of the model?

Question 1 is explored in chapter 2, where SNP data is compressed in order to reduce the processing time of a CNN model, reducing the execution time required to classify SNP data. In chapter 3 the second question is addressed, by analyzing the effect of different CNN models on the precision and accuracy of detection, and demonstrating an efficient CNN-based detection framework. Chapter 4 evaluates the effectiveness of the models proposed in the previous chapter on another genomic effect, and adapts the model for the simultaneous classification of two distinct effects. Finally, in chapter 5 the model from chapter 2, using the efficient implementation demonstrated in chapter 3 is accelerated using an FPGA.

Selective sweep classification

2.1 Introduction

Positive natural selection is a driving factor for the adaption and evolution of a species. The genome of a species that has been affected by positive natural selection features an identifiable region called a selective sweep. A complete hard selective sweep is a selective sweep that occurs as a consequence of a beneficial genetic mutation that is ultimately adopted by the entire population. Any mention of a selective sweep in this work refers to a complete hard selective sweep. In order to analyze the genomic mutations within a population, the genome data can be represented by its single nucleotide polymorphisms (SNPs). Each SNP represents a position (locus) along the genome where a mutation has been observed. These SNPs can be represented as binary states indicating either an ancestral (original) or derived (mutated) state at a polymorphic locus. The SNP data of a population is represented as a 2-dimensional matrix featuring the consecutive SNPs of a unique sample from the population as its rows, where each column corresponds to a specific SNP locus. Through modern DNA sequencing methods, there has been a surge in the available amount of genetic data, leading to datasets of increasingly large sample sizes [51], which can lead to more effective selective sweep detection [46].

The presence of a selective sweep leaves three distinct signatures that are detectable through the analysis of SNPs. The three signatures are: a localized reduction in polymorphism [53], a shift in the site frequency spectrum (SFS) towards higher and lower frequencies of derived variants [12], and a pattern of linkage disequilibrium (LD) with increased LD on both sides of the sweep and reduced LD across the sweep [33]. These signatures are primarily used by selective sweep detection methods through summary statistics [1] and likelihood-based methods [17]. More recently, machine learning methods have proven to be highly successful in identifying selective sweeps [20][56][43][68] [69], outperforming statistical and likelihood-based methods. Machine learning methods use convolutional neural networks (CNNs) [38] to learn the difference between genomes affected and unaffected by a selective sweep. Despite the advances in selective sweep recognition by virtue of machine learning-based methods, practical and robust detection of selective sweeps remains challenging.

Under realistic evolutionary conditions, the selective sweep is often not the only genomic effect at play. Other effects than selective sweeps may simultaneously and similarly introduce signatures in the genome and confound the effect purely introduced by a selective sweep [62], misleading likelihood and summary-statistics-based methods. CNN-based methods that process raw SNP data are more robust to these underlying effects [69] [68], but do not scale well to large sample sizes limiting their practicality. Modern DNA sequencing allows for the efficient accumulation of genetic data samples [51], which can improve the effectiveness of selective sweep detection [46]. As the sample sizes of datasets increase, more SNPs are discovered [59], further growing the amount of raw data to be processed. CNN-based methods generally focus on processing whole SNP matrices [20][56][43][68] [69], which is computationally expensive since the number of computations performed by a CNN scales linearly with both the sample size and the number of SNPs per sample. Other CNN-based methods compute summary statistics in a preprocessing step, and apply a CNN classifier to these statistics [32], but this requires extensive preprocessing, which is also inefficient. Existing CNN-based methods fail to address the prohibitive computational complexity required to process large SNP datasets, and require expensive resources or accelerators, such as GPUs, to run efficiently.

Selective sweep detection is often framed as a classification problem and focuses on optimizing the classification accuracy on narrow genomic segments [20][56][43][68]. Practical applications of selective sweep classification, however, most often require localizing the regions affected by a selective sweep within a larger genomic sequence, making it a detection problem rather than a classification problem. Selective sweep detection is similar to the more general problem of object

detection using CNNs, which is an extensive field of research [71]. To precisely localize the regions affected by selective sweeps, fine-grained detection is required. A classification model can trivially be used for detection by sliding a window over the input data, where a smaller window stride enables increased detection granularity. Increased granularity increases the detection precision and reduces the chance of missing a target since less data is being skipped between windows, but comes at the cost of greater computational expense. Using a fine-grained sliding window, often the stride of the window is less than half the window width, in which case consecutive windows will overlap. Processing the overlapping data for every single window is computationally expensive. Convolutional layers exhibit translational equivariance, which implies that identical input at different positions in the window leads to identical output at different positions in the window. This makes it possible to reuse the CNN output of the overlapping region [52], significantly optimizing fine-grained sliding window detection. Many CNN-based methods for selective sweep classification suggest per-window reordering of SNP data as a pre-processing step before classification [68][20][56]. Reordering data impairs the reuse of data between overlapping windows. When reordering data based on the content of a window, the overlapping components between different windows are not consistent, prohibiting data reuse, and requiring processing the whole window at each position.

The SNP matrix encodes the LD and the SFS but does not include the locations of the SNPs on the genome. The locations of the SNPs must be known to derive the degree of polymorphism within a region, and a reduction of localized polymorphism is one of the signatures of a selective sweep. The SNP matrix and the SNP positions can both be passed to the same CNN model in a process called data fusion. Previously, SNP positions have been fused by separately processing the positions through a fully connected layer, and appending the output to the final layer of a CNN-based classifier [20]. This is an example of late fusion, as the position data is fused with the SNP data at the output stage of the model. One problem with this implementation is that by processing the SNP positions through a fully connected layer the spatial arrangement of the SNP positions is not explicitly passed to the model. In a fully connected layer, each value is combined with each other value, not taking into account the fact that neighbouring position values are more closely related than distant position values. Moreover, late fusion limits the ability of the model to combine the information of the SNP positions with the SNP matrix. Another general framework for analyzing SNP data proposes early fusion, by fusing the distances between SNP positions at the input stage of the model, as an additional input channel to the CNN [13]. In this implementation, the spatial arrangement of the position data is explicitly passed to the model, since the convolutional layers take this information into account. This improves the ability of the model to interpret the position data. When the relative arrangement of the SNP positions is considered, the SNP density, and thus the local polymorphism, at any location within a window can easily be computed. By fusing the SNP positions and SNP matrix at the input of the model, the model can account for any patterns that arise when combining the SNP matrix and the local SNP positions, this is an example of early fusion. This has been applied for the classification of recombination hotspots [13].

In this chapter, FAST-NN (reversed abbreviation of Neural Network for Tracing Sweeps with Allele Frequencies) is introduced. This novel CNN method provides scalable CNN-based selective sweep classification. FAST-NN addresses the scalability problem faced by existing CNN-based methods whilst retaining robustness to confounding factors. FAST-NN does not require any pre-processing to compute summary statistics. It exploits two fingerprints that are informative of selective sweeps, the density of polymorphisms [53] and the SFS [12]. These metrics are fully encoded by a vector of derived allele frequencies, and a vector of SNP distances. Both vectors are one-dimensional and thus are size-invariant to the sample size. This makes FAST-NN scalable to datasets featuring large sample sizes. FAST-NN does not require reordering any data, allowing the model to exploit data reuse when implemented in a detection context for efficient fine-grained selective sweep detection.

By employing early fusion of the derived allele frequency vector with the SNP distance vector, and by optimizing the model architecture through an extensive neural architecture search (NAS), FAST-NN exhibits improved classification accuracy when classifying selective sweeps that are in the presence of various confounding factors. The design of FAST-NN features the following contributions:

- By evaluating the effectiveness of fusing SNP distances with raw SNP data or derived allele frequency data, it is demonstrated that early fusion leads to the greatest performance gain.
- It is demonstrated that derived allele frequencies are an effective method for classifying selective sweeps, especially when used in combination with SNP distances. This allows for

scalable CNN-based selective sweep detection that is practical without requiring any hardware acceleration.

- FAST-NN outperforms state-of-the-art CNN-based selective sweep classification methods, without any window-based preprocessing or data reordering step, enabling integration into an efficient detection framework.

The performance of FAST-NN is evaluated for selective sweeps in the presence of recombination hotspots, population bottlenecks, and migration. Despite its compact 1D data representation, FAST-NN does not compromise performance. Under all evaluated conditions, FAST-NN outperforms existing CNN-based classification methods whilst requiring significantly less execution time. The scientific results of this chapter have been presented in a peer-reviewed conference paper accepted for publication [58].

2.2 Method

2.2.1 Data representation

The SNP matrix is a two-dimensional arrangement of SNPs where the rows represent the samples, or individuals, from the population and the columns represent loci at which a SNP occurs in any of the samples. Each value in the matrix represents the state of the allele at a locus for a sample in the population. A binary '0' represents an ancestral state of the allele, and a '1' represents a derived (or mutated) state. The positions of the loci in the SNP matrix are given in terms of basepairs (bp) where one thousand bp is expressed as a kilobase (kb or kbp). From the SNP matrix and the SNP positions, features indicative of a selective sweep can be computed. The site frequency spectrum (SFS) is the frequency spectrum of polymorphisms over all SNPs. A shift in the SFS, where high and low frequencies become more prominent, is a signature of a selective sweep. This signature is locally encoded in the vector of derived allele frequencies (DAFs) of a SNP matrix. The DAF at a locus is determined by the number of derived SNPs divided by the total number of samples. The DAF at locus i of a SNP matrix is computed as follows:

$$DAF_i = \frac{\sum_{j=0}^{N} \mathrm{a}_{ij}}{N}, \tag{2.1}$$

where N is the sample size and a_{ij} is the SNP value at locus i for sample j. Figure 2.1 illustrates how a SNP matrix looks in its binary format before computing the derived allele frequencies and as a DAF vector. The brightness of a pixel on the DAF vector scales linearly with the derived allele frequency, where a frequency of 1.0 translates to a white pixel and a DAF of 0.0 translates to a black pixel. The size of the DAF vector is independent of the number of samples. Moreover, the DAF vector is robust against missing data, since the derived allele frequency at a certain locus can be computed from any number of samples at a certain locus, by reducing N by the number of missing values.

Complementary to the DAF vector, another signature of selective sweeps is a local reduction in polymorphisms. The degree of polymorphism along the genome is encoded in the density of the SNPs per basepair. This density cannot be computed from the SNP matrix itself, for this metric the basepair positions of the SNPs must be known. The positions of the SNPs can be described by a position vector P. In a previously proposed classification model, the raw position vector has been used as input to a CNN-based classification model [20]. This work focuses on classifying SNP matrices of full simulations. In the case of classifying entire simulations, passing the raw SNP positions to the model does not impose a problem. However, when the SNP matrix used as input to a CNN-based model is a windowed subset of SNPs sampled from a larger sequence of SNPs, using this position vector P as raw data to train and test a classification model can lead to issues. When the model is only trained with data at a specific absolute position, then the model may not generalize to new data where the absolute positions differ from the training data. If the selective sweep in the simulation always occurs at the same absolute position on the genome, then the model may not be able to classify a selective sweep at any other absolute position. To ensure that the model is not biased to an absolute position, the position data must be made independent of the absolute position of the sampled SNP matrix on the genome. One method to remove this bias is by encoding the SNP positions within the matrix as positions relative to a particular column on the matrix. This method, however, leads to another problem. Calculating the SNP positions relative to a particular column is not a translation equivariant operation, and is therefore problematic when implementing the classification model in an efficient sliding window-based detection framework.

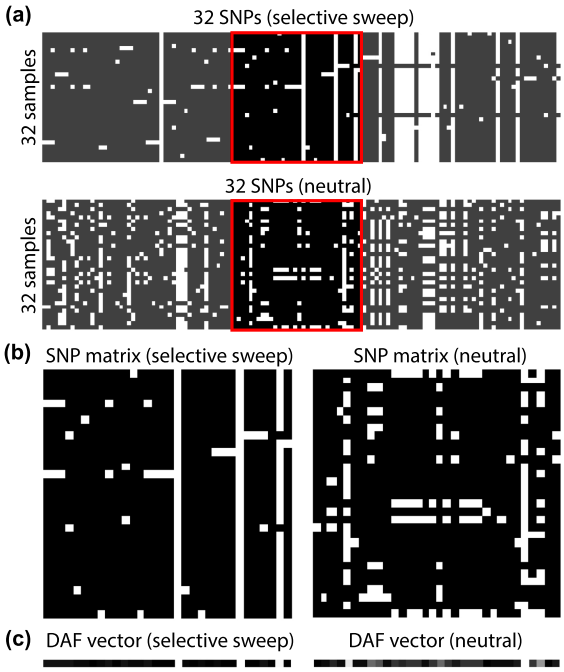


Figure 2.1: Figure (a) shows two 32-SNP windows sampled from larger selective-sweep (top) and neutral (bottom) datasets. Figures (b) and (c) depict the resulting SNP matrices and their visual representation as derived allele frequency vectors, respectively.

Another method to remove the positional bias of the SNP data is by using the pairwise distances between SNPs. This transformation is independent of the absolute position of the matrix and is translation invariant. This method is inspired by the work of [13], where this has been applied for recombination hotspot classification. The pairwise distances between the SNPs are computed as follows:

$$D_i = P_{i+1} - P_i. \tag{2.2}$$

The distance value of the final SNP in a sequence is defined as 0. This distance vector is of the same length as the DAF vector. This has the added benefit that these vectors can be processed as a single matrix. From the DAF vector and the distance vector two of the metrics that are classically used to identify selective sweeps are represented, the SFS and the degree of local polymorphism. When considering a CNN as a classification model, this data representation has a significant impact on the computational efficiency. The complexity of 2D convolution on an $N \times N$ matrix with a $K \times K$ kernel is $\sim O(N^2K^2)$. The complexity of a 1D convolution on a 1D vector of length N with kernel length K is only $\sim O(NK)$ [35]. This decrease in complexity makes 1D CNN models practical for use by researchers and users who do not have access to GPUs or other high-performance computing equipment, especially when large datasets need to be processed. Moreover, the training and inference time of these models is significantly reduced compared to the 2D counterpart. Finally, the amount of data that needs to be stored and passed to the model is reduced, making it more scalable to large datasets. Especially in the scope of genetics, scalability is a major concern. Through advances in DNA sequencing methods, the amount of data that is available for analysis is increasing rapidly [51]. As the number of samples for a genome increases, so does the number of known SNPs along this genome [59]. Therefore the increase in the amount of data to be processed when the sample size increases exceeds a linear relation.

2.2.2 Model architecture and selection

For convolutional neural networks (CNNs), and other types of artificial neural networks (ANNs) there is at present no analytical method to determine an optimal architecture for the problem at hand. Often, architectures are empirically determined based on the complexity of the problem, the expected patterns to be found in the data, and previously confirmed effective architectures. A more rigorous approach to model selection is to perform a neural architecture search (NAS). A NAS attempts to find an optimal neural network architecture within a given search space. Previously, a NAS was used to determine the optimal model for selective sweep classification [68]. In this instance, a number of hyperparameters determining the architecture and training method of the

model were defined. For each hyperparameter a number of options are considered. The selection was then done through an iterative process, where the hyperparameters are optimized sequentially, reducing the number of candidate architectures throughout the process. This method requires less computation than a full grid search. In a grid search, each position in the hyperparameter space is tested. The combination of hyperparameters leading to the best performance is selected.

The NAS that was performed in [68] considered a model that uses the full SNP matrix as input. The rows and columns of the SNP matrix are reordered before it is passed to the CNN. Using this method a lightweight model was selected which exhibited state-of-the-art performance on selective sweep classification in the presence of confounding factors. Because of the different data representations in this work, another NAS is performed to identify the optimal architecture of a model processing DAF data. Considering the 1D data representation that is used, the computational power required to train a single model is significantly reduced. This makes performing a full grid search over a selection of parameters a practical approach. The base model used for the NAS is shown in figure 2.2. The base model is a simplified version of the SweepNet model,

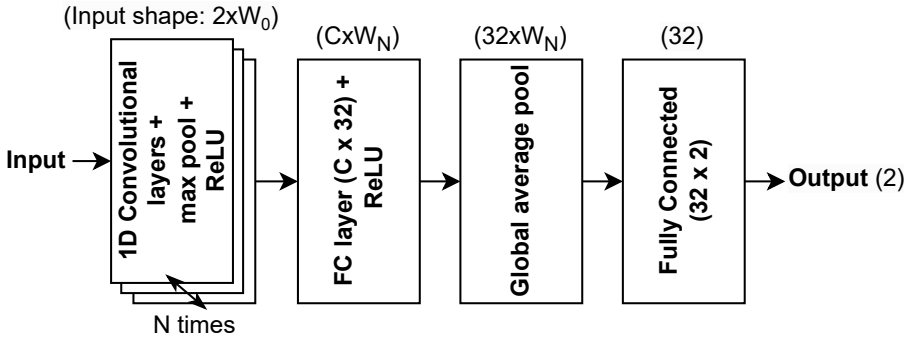


Figure 2.2: Base model for model architecture grid search. The data shape is indicated beside the dataflow arrows. The first step is repeated N times depending on the specific architecture.

omitting the Squeeze-and-Excitation layer for simplicity, and replacing the 2D convolutional layers for 1D convolutional layers. Each convolutional layer is followed by a max pooling layer. The max pooling layer uses a kernel with a width and stride of 2, which was empirically found to yield the best results over the search space. Using a pooling layer, the width of the data reduces after each convolution, decreasing the number of computations and increasing the receptive field of the model. The hyperparameters in the NAS search space, and the considered values for each hyperparameter, are shown in table 2.1. A limited number of options for each hyperparameter

Table 2.1: List of hyperparameters and selected options for model grid search, resulting in a total of 240 candidate models.

Hyperparameter	Options	Candidate models
Conv layers	2, 3, 4, 5	4
Conv channels	8, 16, 32, 64, 80	20
Kernel width	2, 3, 4	60
Kernel stride	1, 2, 3	180

are selected, to reduce the evaluation time. The candidate options are selected based on what has been observed to perform well in other works, and considering the intention for the model to be lightweight. As such the options for the number of layers and channels are not explored beyond these proposed options, as this could result in a model that is larger than intended, in terms of parameters and computational expense.

2.2.3 Memory-efficient data formatting

It has been mentioned that the allelic state of each SNP considered is either ancestral, represented by a binary '0', or derived, represented by a binary '1'. In previous methods focusing on the use of CNNs for selective sweep analysis, the SNP matrix is interpreted as an image [20][68][56][43]. Each pixel in the image represents a single state of an allele. To store these pixels, a single colour channel is used, which can store a pixel value between '0' (white) and '255' (black). This is not memory efficient, each pixel requires one byte of memory, whilst the pixel actually only encodes a single binary state. An additional issue that arises when using an image format to store the

SNP data occurs when implementing the SNP distance data in addition to the SNP matrix. The bandwidth of SNP distances that one would like to store can well exceed the 256 values that an image channel can store. Finally, when storing the derived allele frequency data instead of the SNP matrix, the 8-bit channel limits the precision at which this data can be stored. Considering the memory inefficiency of storing the SNP matrix in an image format, and the shortcomings of an image format to store the distance and DAF data, using an alternative format to store the data is preferable.

In order to store the SNP matrix and the distance data efficiently, the binary SNP states can be packed into bytes, each byte yielding 8 SNP states. The SNP distance data can be stored as floating point values. Each floating point requires 4 bytes to store. The number of bytes required to store the SNP matrix and distance data is calculated in equation 2.3:

$$B_{raw} = W \times \left(4 + \left\lceil \frac{N}{8} \right\rceil\right), \tag{2.3}$$

Where B_{raw} is the number of bytes to store the matrix, W is the width of the matrix, and N is the number of samples in the matrix. To store the DAF vector, each frequency can be stored as a floating point value. In this case, the number of bytes to store the DAF vector and distance data is calculated through equation 2.4:

$$B_{daf} = W \times (4 + 4). \tag{2.4}$$

Equation 2.3 and 2.4 indicate that when the sample size is greater than 32, saving the DAF vector becomes a more memory-efficient method for storing the raw SNP data, and can consequently reduce data loading time. The 1000 Genomes Project [18] can be taken as an example. In this real-world study, the sample size of the gathered data is 2504. In this case, storing the data as DAF vector instead of SNP matrix reduces the amount of data to be saved by 97.5%.

2.3 Evaluation

2.3.1 Experimental setup

To generate neutral SNP data, SNP data containing a selective sweep and data containing several different confounding factors, simulation software ms [26], mssel (kindly provided by R. R. Hudson), mbs [55] and msHOT [25] is used. In continuation of the work done in [68], six datasets containing several different confounding effects are generated. Table 2.2 shows the datasets, along with the commands used to generate these corresponding dataset. The datasets for training consist of 1000

Table 2.2: Datasets used for evaluation. For each confounding factor, two different genomic scenarios were tested. Parameters show the simulation parameters along with the command line options in the corresponding simulation software.

Dataset	**Software**	**Parameters**	**Values**
Mild bottleneck (D1) Severe bottleneck (D2)	Neutral: ms Selective: mssel	Severity (-eN) Duration (-eN) Beginning (-eN) Selection coefficient (-s) Sweep start time (-t)	0.5 (D1), 0.005 (D2) 0.001 (D1), 0.002 (D2) 0.1 (D1), 0.01 (D2) 0.02 0.016
Recent migration (D3) Old migration (D4)	Neutral: ms Selective: mssel	Population join time (-ej) Selection coefficient (-s) Sweep start time (-t)	0.003 (D3), 3 (D4) 0.02 0.005
Recombination hotspot Low intensity (D5) High intensity (D6)	Neutral: msHOT Selective: mbs	Hotspot intensity (-v) Hotspot region size (-v) Selection coefficient (-s) Sweep start time (-t) Mutation rate (-t) Recombination rate (-r)	2 (D5), 20 (D6) 5 kb 0.02 0.005 2000 2000

neutral and 1000 selective simulations, of which a split of 15% is used for validation. The datasets for testing consist of 1000 neutral and 1000 selective simulations. Each simulation consists of 128

samples, and from each simulation, a window of 128 SNPs is extracted, resulting in a 128×128 SNP matrix. For the selective simulations, the origin of the selective sweep is at the center of the window.

The neural networks are realized in Pytorch [45]. The networks are trained for 100 epochs at a learning rate of $0.5 \cdot 10^{-3}$, which empirically showed reliable training of the models. The models are trained with mini-batches of batch size 8 and the Adam optimizer is used [34]. After each epoch, the validation accuracy is computed. The model with the highest validation accuracy is saved for testing. When training using the full SNP matrix as input, the rows of the matrix are shuffled to randomize the training data, reducing the effects of overfitting. Unless mentioned otherwise, the data is saved and loaded as full SNP matrix data.

To evaluate the models, the testing accuracy of the models is analysed. In addition to this, the training and inference time of the models is measured. Execution times were measured on a single CPU core (Intel Xeon at 2.1GHz running Ubuntu 20.04) and on a GPU (Nvidia A40).

2.3.2 Effectiveness of including SNP positions

To evaluate the effectiveness of including the SNP positions, three different model architectures are compared. Firstly, the SweepNet [68] architecture is used as-is, without including SNP position data. Secondly, the SNP distances are processed in a fully connected layer with an input width of 128 and an output width of 64, the output of which is appended to the input of the final fully connected layer of the SweepNet model. This approach to fusing the position data to the SNP data is similar to the method used in [20]. Finally, the SweepNet model is used with an additional input channel added to the first convolutional layer, to include the SNP distances. Each model is trained a total of 10 times for each dataset, no data reordering is applied for any of the models. Figure 2.3

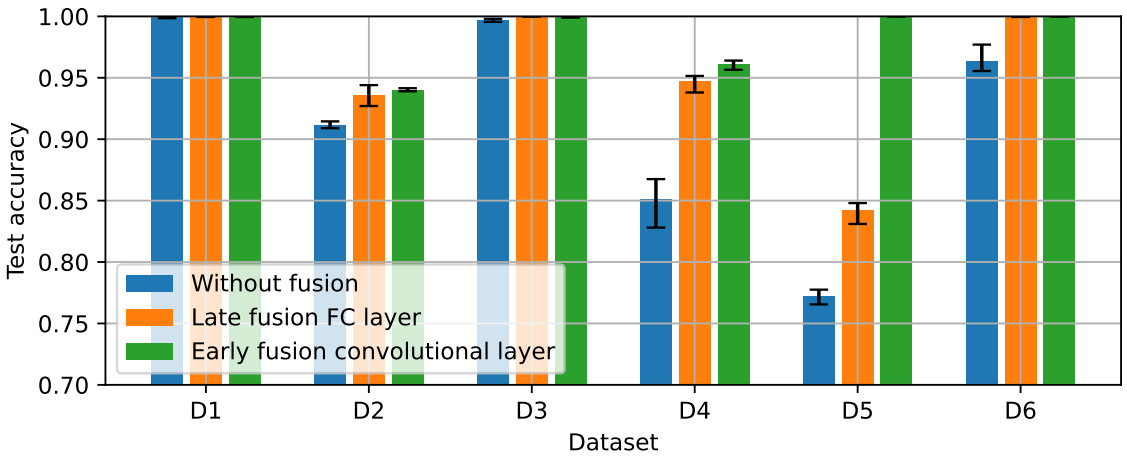


Figure 2.3: Effect of introducing SNP position data to the SweepNet model. Late fusion introduces the SNP distances after the convolutional layers through a fully connected layer. Early fusion introduces the SNP distances as an additional channel next to the SNP matrix. Average accuracy is displayed, and error bars indicate the highest and lowest-scoring models out of 10 training runs.

shows the accuracy of the three architectures for each dataset. The bar chart indicates the mean accuracy of the 10 models, and the error bars indicate the least and best performing model. The figure clearly shows that including the SNP position data can improve the model's accuracy. The extent of this benefit depends on the confounding factor. Moreover, the architecture with early fusion through the convolutional layers shows the greatest improvements, most notably in dataset 5. The improvements from including the SNP position data using convolutional layers as opposed to a separate FC layer can be explained by the fact that this method of including the positions highlights the spatial arrangement of the positions. It makes it more feasible for the model to learn local SNP densities and other spatial patterns. Moreover, the model is able to combine the SNP distance data with the SNP matrix at an earlier stage, potentially leading to useful patterns for recognition.

2.3.3 Model architecture search

The model architecture search as described in section 2.2.2 is performed. The models in the architecture search are evaluated by training and validating the models on dataset D4, which

is the dataset featuring old migration. As demonstrated in figure 2.3, this is one of the more challenging confounding factors to perform classification in. By optimising the model for this dataset, a model with optimal performance, even under challenging conditions, is selected. Figure 2.4 shows the maximum validation accuracy during training for each of the candidate models.

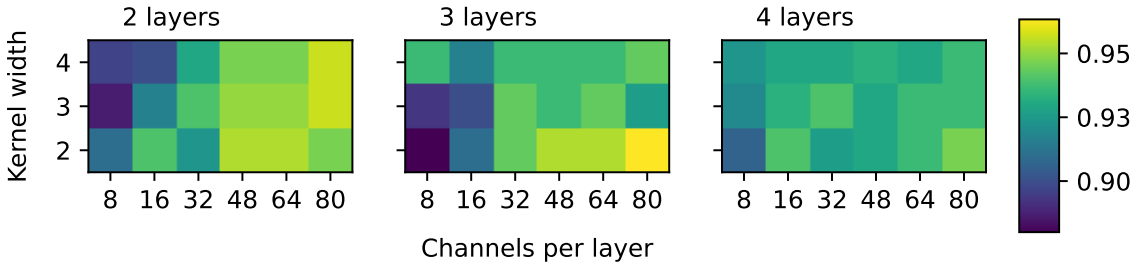


Figure 2.4: Validation accuracy of candidate 1D convolutional models. Trained on dataset 4 (old migration).

The model with 3 convolutional layers, each consisting of 80 channels and a kernel width of 2 has the highest validation accuracy at 0.963. The models with a lower number of channels per convolutional layer consistently show a lower validation accuracy. These models likely do not have sufficient parameters to capture the complex patterns indicating a selective sweep. The models consisting of 4 convolutional layers overall do not perform as well as the models with fewer layers. This is due to these models being more prone to overfitting. Figure 2.5 Shows the training and validation accuracy of the best model for each number of convolutional layers.

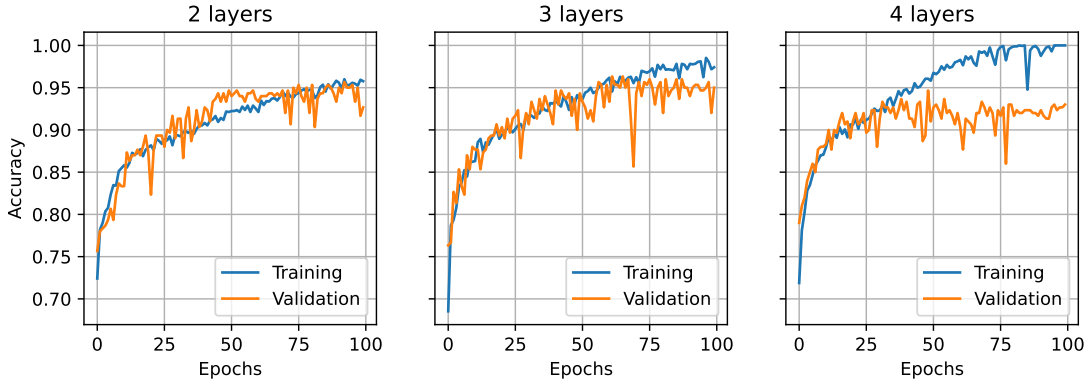


Figure 2.5: Training and validation accuracies of best models from model architecture for each number of convolutional layers. The models with more layers tend to overfit.

Figure 2.5 clearly illustrates that the model with 4 layers overfits, causing a lower validation accuracy. The model with 3 layers slightly overfits as well, but maintains a greater validation accuracy than the best model with 2 layers.

2.3.4 Effectiveness of allele frequency as input

Having selected a candidate model, the performance of this model is evaluated and compared to the SweepNet model which uses early SNP distance fusion, unless specified otherwise. All references to SweepNet in this comparison refer to SweepNet using early distance fusion. Table 2.3 shows the inference and training time for regular SweepNet using the full SNP matrix and for FAST-NN processing only the allele frequencies.

The models reported in table 2.3 are trained and tested on a shared resource, and can therefore be prone to slight variations in execution time due to resource allocation by the server. To minimize the influence of this on the execution time of the models, the load of the server was monitored, and out of 10 runs the one where the lowest interfering server load was observed is reported in table 2.3. On the CPU, FAST-NN trains between 22.19x and 624.77x faster while performing inference between 29.69x and 763.73x faster, when the sample size ranges from 64 to 1000 samples. On the GPU, the speedups range between 1.60x and 19.60x for training, and between 5.67x and 11.22x for testing, both increasing with the sample size. The reduced GPU speedup can be attributed to the data transfers to/from the GPU. Due to the short execution time of FAST-NN, the communication

Table 2.3: Training time per epoch and inference time of SweepNet and FAST-NN for SNP matrices of width 128 and height 64, 128, and 1000 using various data loading methods.

Model	Input	64 samples				128 samples				1000 samples			
		CPU		GPU		CPU		GPU		CPU		GPU	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
SweepNet	images	51.25	33.85	2.09	6.12	106.27	63.81	3.26	5.9	1387	779	30.57	13.13
	binary	45.33	25.91	2.19	1.11	95.5	54.1	4.27	1.88	1339	739.37	44.5	16.59
FAST-NN	images	2.89	1.65	1.7	6.5	3.56	2	2.14	4.73	9.95	7.05	8.31	7.73
	binary	2.31	1.14	1.31	1.08	3.19	1.35	2.16	1.64	5.94	4.25	5.4	3.41
	binary 1D	2.35	1.17	1.74	1.31	2.5	1.01	1.82	1.32	2.22	1.02	1.56	1.17

Table 2.4: Speedup of training and inference compared to SweepNet using images as input for SNP matrices of width 128 and height 64, 128, and 1000 using various data loading methods.

Model	Input	64 samples				128 samples				1000 samples			
		CPU		GPU		CPU		GPU		CPU		GPU	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
SweepNet	images	1	1	1	1	1	1	1	1	1	1	1	1
	binary	1.13	1.31	0.95	5.51	1.11	1.18	0.76	3.14	1.04	1.05	0.69	0.79
FAST-NN	images	17.73	20.52	1.23	0.94	29.85	31.91	1.52	1.25	139.40	110.50	3.68	1.70
	binary	**22.19**	**29.69**	**1.60**	**5.67**	33.31	47.27	1.51	3.60	233.50	183.29	5.66	3.85
	binary 1D	21.81	28.93	1.20	4.67	**42.51**	**63.18**	**1.79**	**4.47**	**624.77**	**763.73**	**19.60**	**11.22**

overhead becomes a bottleneck. This unfavourable computation-to-communication ratio generally occurs when performing small computational tasks using hardware accelerators with dedicated memory space. Moving data in larger batches reduces the number of transfers and can increase the GPU speedup.

Figure 2.6 shows the execution time for training and inference of SweepNet and FAST-NN. Clearly, the FAST-NN model scales better with larger windows. This is in particular the case when using a CPU, but for large window sizes, and especially when training, FAST-NN has a significant benefit on the GPU as well.

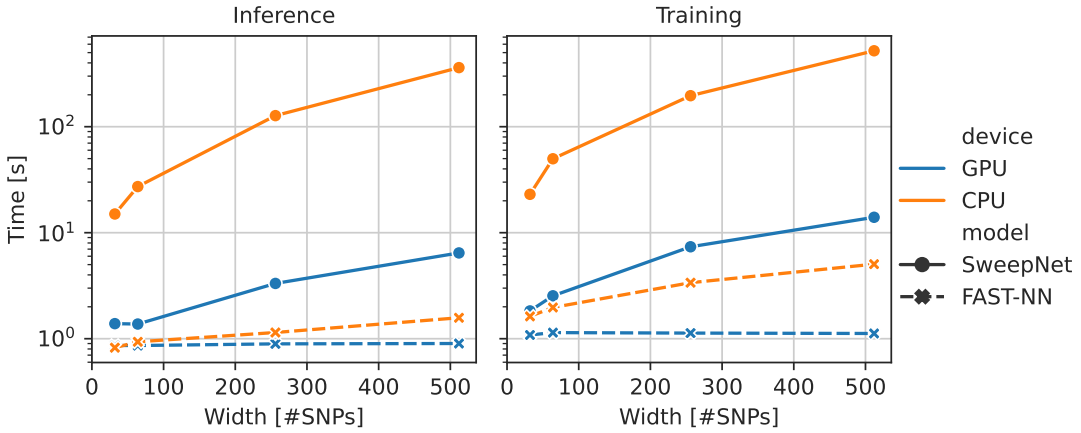


Figure 2.6: Training and inference times for increasing window width. Time (y-axis) is displayed on a logarithmic scale.

Figure 2.7 shows the test accuracy of the 1D model compared to SweepNet. For datasets D1, D3, D5, and D6, both models achieve test accuracies between 0.998 and 1.0. For the challenging confounding factors of a severe bottleneck (D2) and old migration (D4), both models have comparable performance: SweepNet achieves maximum accuracies of 0.942 and 0.964, respectively, while FAST-NN achieves 0.944 on both.

2.3.5 Comparing to summary statistics

Section 2.3.4 demonstrates that FAST-NN has a computational advantage over SweepNet, and is almost as accurate as SweepNet (when SweepNet uses early distance fusion), despite only processing allele frequencies instead of raw SNP data. Both of these models demonstrate the performance of a CNN-based approach for selective sweep classification. To put the relative performance of these

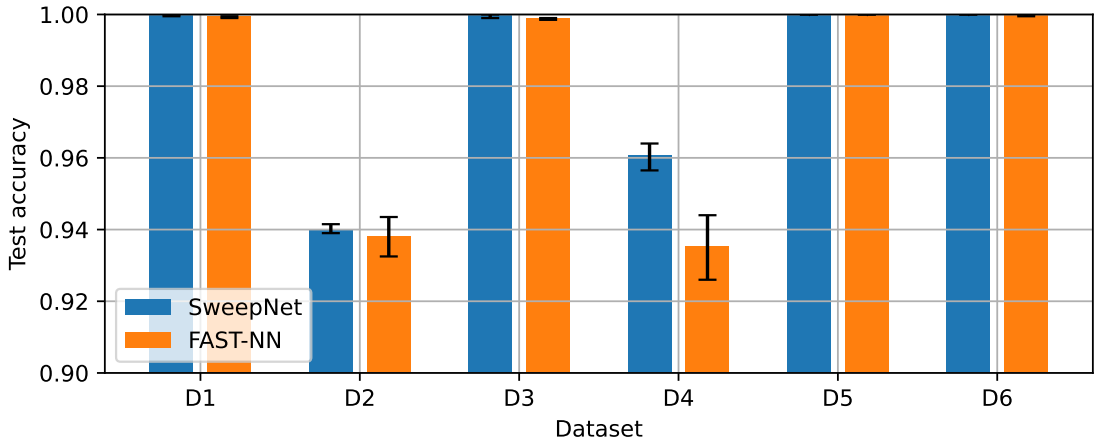


Figure 2.7: Mean accuracy of SweepNet compared to the 1D model using only allele frequency data over 10 training instances. Error bars indicate the best and worst models out of 10 runs.

models into perspective with classical (non-machine-learning) method, several summary statistics for neutrality have been used to classify the datasets evaluated in this chapter. We compared the performance of FAST-NN with the classification performance of summary statistics used for sweep detection. Tajima's D [54], Fu and Li's D and F [21] and Rozas' R_2 [49] are computed using PopGenome [48]. PopGenome is a versatile tool for statistical analysis of population genomics, written in R. The μ statistic [1] is a summary statistic that combines each of the three signatures of selective sweeps: localized reduction of polymorphisms, a shift in the site frequency spectrum, and a specific pattern in linkage disequilibrium. The μ statistic is efficiently computed by the open-source software RAiSD, which is implemented in C.

Table 2.5 shows the maximum classification accuracy attained using each of the summary statistics, and using FAST-NN trained on 1,700 simulations. The μ statistic outperforms the other statistics for each of the datasets, except for the severe population bottleneck, where Fu and Li's D and the μ statistic achieve an accuracy of 0.9215 and 0.92, respectively. FAST-NN has an accuracy that is greater or equal to each of the summary statistics for all evaluated datasets except D1, where FAST-NN has a single misclassification and the μ statistic has a perfect score. Table 2.6 shows the execution time of PopGenome and RAiSD, along with the inference time of FAST-NN. When evaluating PopGenome, only the processing time of the function that computes the summary statistic is included in the execution time. Running PopGenome computes all supported summary statistics for testing neutrality in one run. PopGenome can run in fast mode, which speeds up the computation significantly, but it does not support computing Rozas' R_2. RAiSD outperforms PopGenome in terms of execution time, due to its efficient C implementation. The execution time of RAiSD is greater than the inference time of FAST-NN on a CPU, taking an average of 1.87 seconds and 1.24 seconds, respectively. However, using FAST-NN requires training the model, which takes an average of 3.19 seconds per epoch when training on 1,700 simulations.

Table 2.5: Classification accuracy of neutrality tests using various summary statistics, and using FAST-NN. Each dataset consists of 1,000 neutral simulations and 1,000 simulations featuring a selective sweep. Each simulation is classified by a window of 128 SNPs, where the window is centered around the selective sweep, when one is present.

Dataset	Tajima's D	Fu and Li's F	Fu and Li's D	Rozas' R_2	μ statistic	FAST-NN
Mild population bottleneck (D1)	0.987	0.9845	0.954	0.987	1.0	0.9995
Severe population bottleneck(D2)	0.8565	0.9165	0.9215	0.861	0.92	0.9435
Recent migration (D3)	0.894	0.848	0.663	0.894	0.978	0.999
Old migration (D4)	0.706	0.7435	0.706	0.7135	0.854	0.944
Low intensity recombination hotspot (D5)	0.609	0.613	0.599	0.608	0.708	1.0
High intensity recombination hotspot (D6)	0.513	0.508	0.513	0.5135	0.9035	1.0

2.3.6 Comparing to other CNN-based methods

The FAST-NN CNN model has been benchmarked against other methods CNN-based methods, the methods that FAST-NN is compared to are ImaGene [56], diploS/HIC [32], SweepNet [68] (as is, without SNP distances), and the model of Nguembang Fadja et al. [43]. DiploS/HIC uses 12

Table 2.6: Execution time (seconds) of the tools used to compute the summary statistics in table 2.5, and execution time to perform inference using FAST-NN. Running PopGenome in fast mode computes Tajima's D, Fu and Li's F, and Fu and Li's D. Running PopGenome in normal mode also computes these statistics, with the addition of Rozas' $\mathbf{R_2}$. RAiSD computes the μ statistic.

Dataset	**PopGenome**	**PopGenome (fast)**	**RAiSD**	**FAST-NN**
Mild population bottleneck (D1)	10.44	4.00	1.87	1.22
Severe population bottleneck (D2)	10.68	4.17	2.17	1.22
Recent migration (D3)	10.23	4.01	1.79	1.19
Old migration (D4)	10.61	3.98	1.79	1.34
Low intensity recombination hotspot (D5)	10.49	4.13	1.75	1.23
High intensity recombination hotspot (D6)	10.76	4.11	1.82	1.25

summary statistics to classify selective sweeps, FAST-NN uses derived allele frequencies, and all other methods use raw SNP data. To prevent any of the models from overfitting, each model has been trained for 10 epochs using an extended dataset composed of 50,000 neutral simulations and 50,000 simulations containing a selective sweep.

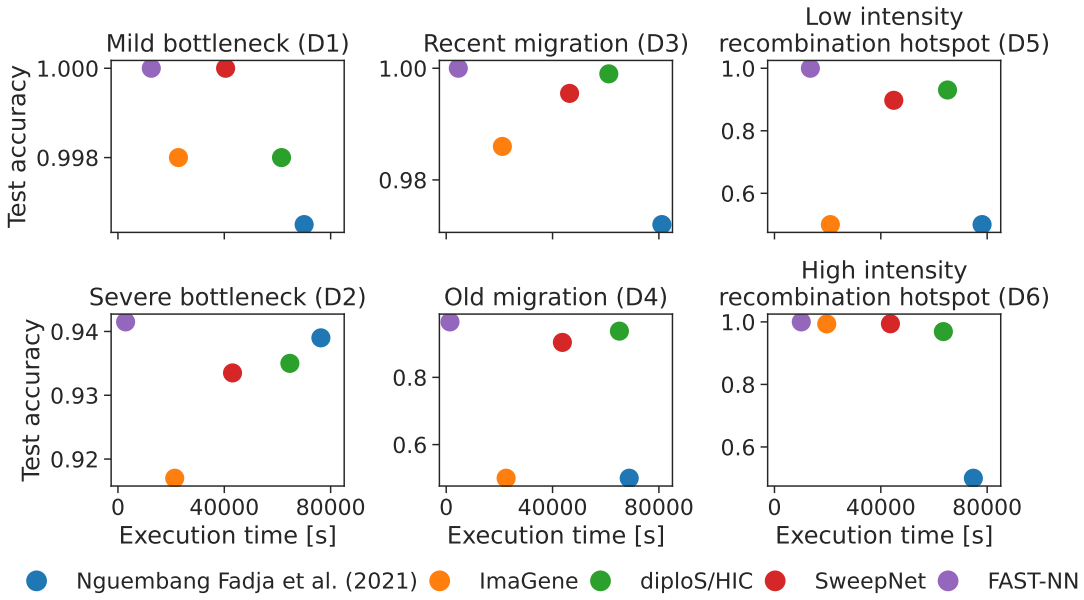


Figure 2.8: Total time (preprocessing, training, and inference) and test accuracy for various CNN-based sweep classification methods.

Figure 2.8 plots the total CPU execution time, including preprocessing, training (until the best epoch), and inference, against accuracy. It is observed that FAST-NN has a lower execution time than all other methods for each dataset. FAST-NN also has a higher or equal accuracy to the other methods that were tested. The only method based on summary statistics, diploS/HIC, has an accuracy of at least 0.935 for any dataset containing population bottlenecks (D1 and D2), an accuracy of at least 0.938 for datasets featuring migration (D3 and D4), and has an accuracy of at least 0.931 for datasets that feature recombination hotspots. While FAST-NN respectively scores an accuracy of at least 0.942, 0.965, and 1.0 under these confounding factors. FAST-NN is between 4.9 and 45.9 times faster than diploS/HIC, depending on the dataset. Across all datasets, ImaGene achieved the second shortest execution time, but it fails to train with the datasets simulating old migration (D4) and a low-intensity recombination hotspot (D5).

2.4 Discussion and conclusion

In this chapter, the effectiveness of using allele frequencies and single nucleotide polymorphism (SNP) pairwise distances as input to a CNN for complete hard selective sweep classification has been demonstrated. By evaluating various datasets featuring various confounding effects, it is shown that FAST-NN exhibits lower execution times than a number of other CNN-based classification methods, whilst scoring higher classification accuracies. This performance gain can be

attributed to careful model selection using a neural architecture search, and effective fusion of derived allele frequencies and pairwise SNP distances. Allele frequency-based classification scales better with datasets composed of a large number of samples since the amount of data is independent of the sample size. This reduction in data allows FAST-NN to use wide input windows whilst maintaining low execution times. Because FAST-NN does not require reordering data, no preprocessing is required, which further increases the efficiency of the implementation. Moreover, because the data is not rearranged in a preprocessing step, the (intermediate) results of the CNN are translationally equivariant. This allows the integration of FAST-NN in an efficient CNN-based selective sweep detection framework. Chapter 3 will further improve the FAST-NN model, specifically for selective sweep detection, and will demonstrate the efficiency of the network for scanning extensive genomic regions.

Selective sweep detection

3.1 Introduction

Identifying which mutations in the genome lead to positive natural selection can be insightful and is a resourceful tool in the analysis of viruses and drug treatments [7] [16]. A genetic mutation that leads to positive natural selection can be detected by analyzing single nucleotide polymorphisms (SNPs). Positive natural selection leads to a distinct SNP pattern, called a selective sweep. When the mutations driving these selective sweeps are adapted by the entire population of the sampled species, this is referred to as a complete hard selective sweep. This work focuses exclusively on complete hard selective sweeps and any reference to selective sweeps refers to this type of selective sweep. Ample work has been done towards improving classification models for selective sweeps, initially through classical statistical analysis of the SNP data [53] [12] [33] [1], and more recently by means of convolutional neural networks (CNNs) [20][56][43][68] [69]. Most of these works lay their main focus on the classification of narrow genomic segments, labeling these segments as a selective sweep (positive) or neutral segment (negative) [20][56][43][68]. Optimizing a model that is able to classify a set of SNPs as selective or neutral is an important step in finding regions on real genomes that are affected by selective sweeps, hence the need to design effective classification models. However, often in a practical application, the goal is not to classify a single genomic segment, but to detect where, along a real genome, a selective sweep is present. In previous methods, preprocessing steps that may hinder this goal are suggested, such as data reordering are suggested [20] [56] [68]. Whilst it may help a learning model classify data, reordering the data can hinder the efficient implementation of a CNN-based method for detecting selective sweeps. Reordering data can introduce redundant computations, especially when aiming for fine-grained detection, which makes the method scale poorly when applied to extensive large datasets. Moreover, classification performance is often used as the sole metric to determine the quality of a model [20, 56, 43, 68], however, the most effective training method for models that robustly detect selective sweeps can differ from the optimal method for classifying SNP segments.

In the field of computer vision, detection and classification are two distinct applications. When applying classification, the goal is to categorize targets by a set of classes. The output of a classifier is a categorical distribution indicating the probability of the target belonging to each class. In the context of selective sweep classification, the classification model returns the probability that a genomic segment is affected by a selective sweep. When performing detection, the goal is to locate the position of a target, if any target is to be found in the input. The detection algorithm aims to return the coordinates of one or multiple target(s), along with the probability distribution associated with each target. In the context of selective sweep detection, the detection model predicts at which SNP positions the selective sweep can be found, along with the probability that the SNP belongs to the selective sweep. A classification model can be applied in a detection algorithm in several ways, one of which is by processing the input in segments by means of a sliding window [71]. When implementing this for selective sweep detection, the efficiency of the algorithm is of major importance to the practicality of the method, since the genomes of real species are extensive, and can contain a large number of SNPs. If the detection algorithm is inefficient, the processing time required for fine-grained detection becomes infeasible. To illustrate, the 1000 Genomes project gathered over 80 million SNPs from the human genome across all sampled populations [18]. When applying inefficient fine-grained detection over this dataset, the processing time becomes impractical. Fine-grained scanning of only a million SNP locations from a population of 1000 individuals, using a fine-grained sliding window without any optimization, even when accelerated using a GPU, would take over 4 days. Clearly, a more efficient method is required to make this approach practical.

Because of the variability in the evolutionary processes of genomes, the number of SNPs in

a genomic region is inconsistent. FAST-NN is designed to support data of variable input width, which has the advantage that the model does not need to be altered when classifying data of different input widths. One can train a model FAST-NN for an input of 128 SNPs, and use the same model on data consisting of 512 SNPs. This is possible due to the global average pooling layer in FAST-NN, which averages over the spatial dimensions after the final convolutional layer of the model. In this architecture, the input shape to the fully connected output layer is constant, and independent of the input shape. This design choice is also used in the SweepNet model [68]. When using a CNN-based model for scanning genomes, the advantage of this flexibility to different input widths is absent, since, in this case, it is assumed that the total length of the genomic sequence far exceeds the width of one window. Hence, it is possible to choose any number of SNPs per window, making it possible to ensure that each window has an identical number of SNPs. Using a global average pooling layer can implicate a trade-off to a potential gain in classification performance. By averaging the output of the final convolutional layer over the spatial dimensions, spatial information is discarded between the final convolutional layer and the fully connected output layer. This may restrict the ability of the model to learn global spatial patterns in the data. The receptive field in a CNN is the range of inputs that are combined to compute a single output in a convolutional layer [41]. Convolutional layers deeper in a network have an increasingly wide receptive field, enabling these layers to learn increasingly global patterns. The final layer in most CNN models is fully connected, combining data derived from all its input values. The fully connected layer can learn global patterns in the data. By averaging over the spatial dimensions before applying the fully connected layer, the model only learns patterns that are detectable within the receptive field spanned by the final convolutional layer. In the context of selective sweep classification, a wide receptive field enables the model to learn global patterns in the site-frequency-spectrum and polymorphism density, possibly enhancing the ability to identify selective sweeps.

This work focuses on designing a CNN-based model that is optimized for implementation in a detection framework, which can perform efficient fine-grained detection of selective sweeps. The effect of using a model featuring a wide receptive field is explored, evaluating the performance for various input widths. In this chapter the FAST-NN model introduced in chapter 2 is redesigned, focusing specifically on detection performance metrics. This work features the following contributions:

- The design of a CNN architecture that outperforms the selective sweep detection performance and precision compared to the state-of-the-art. The proposed implementation enables fine-grained detection without significant computational overhead, allowing for practical application of the model on real data.
- By exploring the effect of input width on different CNN-based models, it is shown that wide inputs can improve detection true-positive-rate and precision, especially when used with a model with a wide receptive field. The models are implemented in an efficient fine-grained detection framework by altering the models to generate dense output, without recomputing data from overlapping windows [66] [52].
- Through these optimizations a scan of all 22 human autosomes is performed in just over 1 minute, demonstrating the practicality of this method within the context of a real genomic dataset.

The detection performance is evaluated for various evolutionary scenarios, demonstrating that the model performs detection with improved robustness to genomic effects that can confound the selective sweep. Employing the proposed efficient detection framework enables researchers to use this method for the practical analysis of realistic genetic data.

3.2 Method

3.2.1 Efficient sliding window

To perform efficient and effective selective sweep detection along a genomic sequence, wide genomic segments need to be classified, without significantly reducing computational efficiency. A sliding window is a trivial method to apply a classification model for detection. However, a sliding window that divides a genomic sequence into non-overlapping segments may not produce optimal results. A classification model trained to classify selective sweeps is expected to perform better when more SNPs affected by the sweep lie within the window. This is most likely to occur when the center of the sweep aligns with the center of a window. However, if a sliding window without any overlap

between windows is used for detection, the center of the sweep might not lie in the center of any of the windows. This reduces the ability of the detection method to detect the selective sweep successfully. Instead, a sliding window with overlapping windows can be used, where increasing the overlap increases the probability that the center of the selective sweep lies at the center of any of the windows. This method can cause significant computational overhead when implemented without any optimizations. In a naive implementation, each window must be processed by the CNN model separately. This implies that, when windows with overlap are used, the classifier needs to process the overlapping data multiple times. Moreover, when using wider inputs, the computational complexity increases since the amount of overlap between windows increases. Detection using a sliding window over the entire genomic sequence can be viewed as imposing a grid over the sequence and evaluating a window centered around each gridpoint. When performing detection by imposing a grid over the genome, processing a window centered at each gridpoint, the number of SNPs that need to be processed is $W_w \cdot G$, where W_w is the width of the window and G is the number of grid points. For the finest level of detection, a window can be sampled at each SNP, which requires computing $W_w \cdot (W_s - W_w)$ SNPs, where W_s is the number of SNPs in the entire genomic sequence where detection is performed. This implementation for detection becomes impractical when using a wide window on a lengthy genomic sequence. Fortunately, convolutional layers in a CNN are translation equivariant, this implies that identical input data at different positions leads to identical output at different positions. Because of this property, the overlapping data that is shared between different windows only needs to be processed once by the convolutional model, since the output data of convolutional layers can be reused [52]. Figure 3.1 illustrates how windows

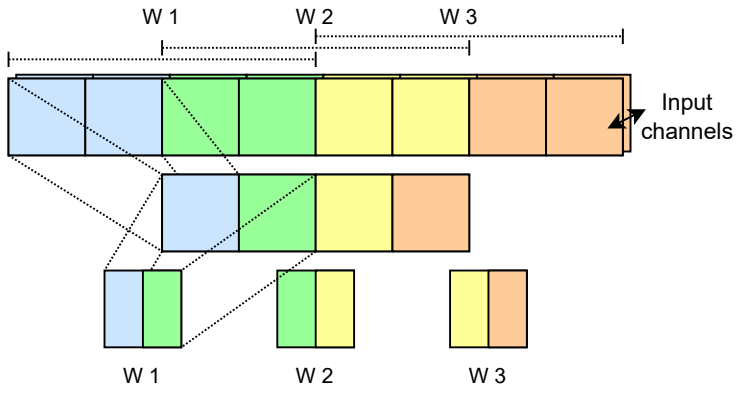


Figure 3.1: When using a CNN classification model on input with overlapping data, the intermediate output of one window can be reused to compute the output of the next window. This example illustrates this for three windows of width 4, for two convolutional layers. The first layer has a kernel size of 2 and stride of 2, the second layer has a kernel size of 2 and stride of 1.

with overlapping input data can reuse intermediate output values. One constraint to this property arises when any of the convolutional layers or pooling layers apply a stride greater than 1, which is the case in the illustration. In this case, the overlapping data between windows only produces identical output if the window stride satisfies equation 3.1.

$$S_w \cdot = k \cdot \prod_{n=0}^{N} S_n \tag{3.1}$$

Here N is the number of convolutional and pooling layers, S_n is the stride of layer n, S_w is the sliding windows stride and k is any integer number. The limitation imposed by equation 3.1 reduces the ability to perform fine-grained detection efficiently, since a larger window stride decreases the potential precision of the algorithm, and running the model multiple times to compensate for this reduces the efficiency of detection and may lead to recomputing identical intermediate outputs. Fortunately, it is possible to adjust the model architecture to allow for a dense output, meaning one output for each input, by using dilated convolutions [66]. Dilation implies that the convolutional kernel is not contiguous, but samples separated inputs. The second layer in the network shown in figure 3.2 illustrates this kind of kernel. By choosing the amount of dilation based on the stride in the original model, a dense model using dilation can be designed that is functionally identical to a model using a stride greater than 1. Figure 3.2 shows the dense model equivalent to the model shown in figure 3.1. The dense implementation of the model needs to perform additional

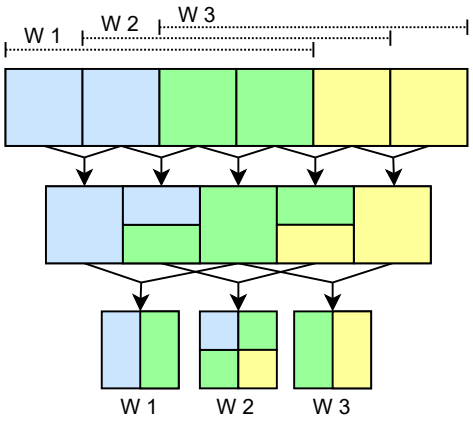


Figure 3.2: Dense outputs are generated By removing the stride from the first network layer. Through dilated convolutions in the subsequent layer, outputs are generated for a sliding window with a stride of 1. Subsampling the outputs from odd windows will give outputs identical to the example in figure 3.1.

computations to generate the additional outputs at each layer. The number of computations that the convolutional layers and pooling layers in a CNN need to make can be expressed using equation 3.2. In this equation, computations from a convolutional layer and a pooling layer are treated equally.

$$C = K_0 \cdot \left\lfloor \frac{W_s - K_0}{S_0} + 1 \right\rfloor + K_1 \cdot \left\lfloor \frac{W_s - K_1}{S_0 S_1} + 1 \right\rfloor, ..., K_{N-1} \cdot \left\lfloor \frac{W_s - K_0}{\prod_{n=0}^{N-1} S_n} + 1 \right\rfloor \tag{3.2}$$

C is the number of computations, K_n is the kernel at layer n, W_s is width of the entire genomic sequence, S_n is the stride at layer n and N is the number of layers in the model. Equation 3.2 is simplified by assuming that for all n, $K_n << W_s$. In the case of a realistic genomic sequence, where up to millions of SNPs are expected, this is a reasonable assumption. Doing so, the amount of computations can be simplified to equation 3.3.

$$C = K_0 \cdot \frac{W_s}{S_0} + K_1 \cdot \frac{W_s}{S_0 S_1}, ..., K_{N-1} \cdot \frac{W_s}{\prod_{n=0}^{N-1} S_n} = W_s \cdot \sum_{i=0}^{N-1} \frac{K_i}{\prod_{n=0}^{i} Sn} \tag{3.3}$$

When adapting the architecture of the model to generate dense output, the stride of each layer is set to 1, which increases the number of computations in the network. The increase in computations when converting the model to a dense model is described by equation 3.4.

$$\frac{C_{dense}}{C} = \frac{\sum_{i=0}^{N-1} K_i}{\sum_{i=0}^{N-1} \frac{K_i}{\prod_{n=0}^{i} Sn}} \tag{3.4}$$

Applying equation 3.4 to the FAST-NN model from chapter 1 yields the following.

$$\frac{C_{dense}}{C} = \frac{3+2+3+2+3+2}{3+\frac{2}{2}+\frac{3}{2}+\frac{2}{2^2}+\frac{3}{2^2}+\frac{2}{2^3}} = \frac{15}{7}$$

As stated previously, the number of redundant computations performed when the efficient method is not used depends on the number of grid points used for detection. A dense grid implies more overlap between windows and will favour the dense output implementation. For the number of computations to be reduced by the dense implementation, equation 3.5 needs to hold.

$$W_s \cdot \frac{C_{dense}}{C} < W_w \cdot G \tag{3.5}$$

When performing detection over a genomic sequence of 10k SNPs, using a window width of 128 (the width used for classification in chapter 2) and using the FAST-NN model, the minimum number of gridpoints for the dense method to be more efficient is computed as follows.

$$G > \frac{W_s \cdot \frac{C_{dense}}{C}}{W_w} = \frac{10000 \cdot \frac{15}{7}}{128} = 167$$

When the naive sliding window algorithm is applied for dense detection without reusing overlapping data, the model needs to process $W_w \cdot (W_s - W_w)$ SNPs. When using the efficient sliding window, each SNP position only needs to be processed once, which is a total of W_s positions. By assuming that, for genomic data, $W_s >> W_w$, the efficient approach reduces the computational complexity by the following.

$$\frac{W_s}{W_w \cdot (W_s - W_w)} \simeq \frac{W_s}{W_w \cdot W_s} = \frac{1}{W_w} \tag{3.6}$$

Note that the computational overhead of the efficient sliding window is constant with respect to window width, therefore, using this method, a wide input window will not impact the computational complexity of the convolutional and pooling layers. To continue the example given the FAST-NN model, again considering a window width of 128, the net reduction in computations using the efficient fine-grained sliding window can be computed by combining equations 3.4 and 3.6.

$$\frac{C_{dense}}{C} \cdot \frac{1}{W_w} = \frac{15}{7} \cdot \frac{1}{128} = 0.0167$$

Using a window width of 128 with the FAST-NN model, the computational overhead of the convolutional and pooling layers is reduced by 98.3%.

3.2.2 Model selection

By virtue of the sliding window method described in section 3.2.1, when performing detection with overlapping windows, increasing the input width does not increase the computational complexity of the convolutional layers. To this end, it is important to consider how a model can be designed that is able to benefit from wide inputs. In a CNN, the receptive field of a convolutional layer describes the spatial range of input values used to compute a single output value. A convolutional layer can only learn patterns that are detectable within the range of the receptive field. If the receptive field of all layers within a CNN is narrow, this may limit the model's ability to detect the wide patterns in any of the selective sweep signatures. This can impose a bottleneck for effective selective sweep classification. The receptive field of a convolutional layer in a CNN is described by equation 3.7 [9].

$$r_0 = \sum_{n=0}^{N} \left((K_n - 1) \sum_{i=0}^{n-1} S_i \right) + 1 \tag{3.7}$$

Where r_0 is the receptive field size at the final convolutional layer of the CNN, N is the number of convolutional and pooling layers in the model architecture, K_n is the kernel at layer n and S_i is the stride at layer i. Applying this equation to compute the width of the receptive field for the SweepNet model [68], the ImaGene model [56] and the model by Nguembang Fadja et al. [43], it is found that the respective receptive fields of the final layers with respect to the input are 7, 29 and 232. Depending on the density of the SNPs, in terms of basepairs, the receptive fields of the former two models may not cover a large part of the region affected by the selective sweep, limiting the model's ability to learn and detect global patterns in the data. This is also the case for the FAST-NN model which has a receptive field of 15. Moreover, FAST-NN and SweepNet take the average over the spatial dimensions at the output of the convolutional layers, removing spatial information after the convolutional layers. The ImaGene model and the model by Nguembang Fadja et al. process the output of the convolutional layers through a fully connected layer, enabling the models to learn spatial patterns in the outputs of the final convolutional layer. An advantage of averaging the output of the convolutional layers over the spatial dimensions is that the architecture of the model is independent of the input shape. The model can be used on inputs of any width, which is convenient when classifying genomic segments of varying widths. This flexibility is unnecessary when applying the classification models for detection since the input width is sampled from a wider genomic sequence, which allows sampling at a fixed width. Thus, in order to prevent the potential limitations that global average pooling and a narrow receptive field may induce in the FAST-NN model, the receptive field of the model is increased, by increasing the kernel sizes of the convolutional layers to 3, and by appending three additional convolutional layers to the model, each with a kernel size of 6 and a stride of 2. The global average pooling layer is omitted, and the output of the convolutional layers is directly passed to a fully connected layer. To limit the number of parameters introduced by the additional convolutional layers and the larger fully connected layer, the number of channels in each convolutional layer is reduced to 32. In order for this model to accept narrow input widths down to 32 SNPs, the appended CNN layers apply zero-padding to the input of the layers. The receptive field of this new model is 309. In order for this model to learn the global patterns of the selective sweep, it is necessary to use a

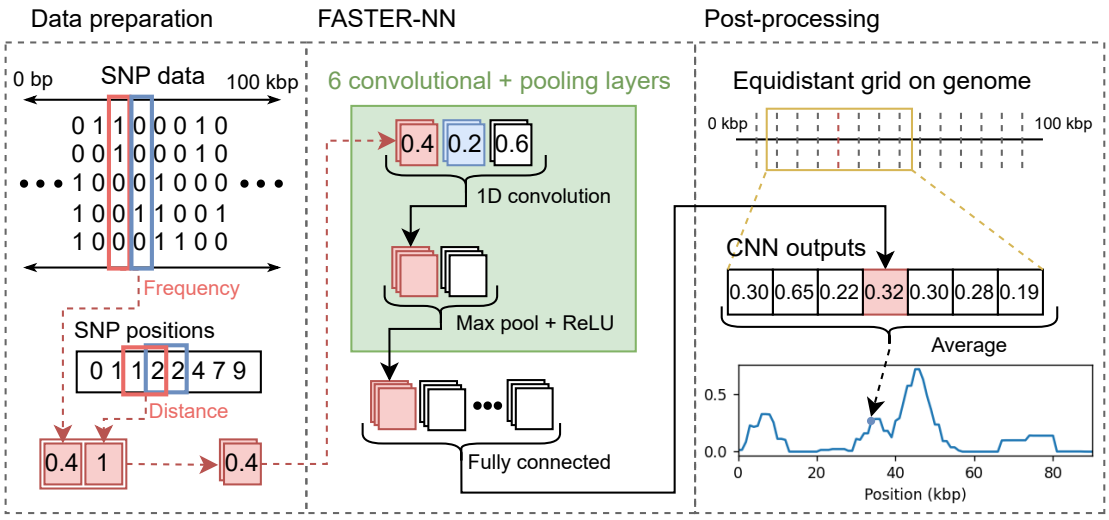


Figure 3.3: Overview of the detection framework. In the data preparation stage, the derived allele frequency and the SNP distances are computed. In the second stage, FASTER-NN is used to perform classification, producing an array of classification scores corresponding to the SNP positions. In the final stage, the dense output scores are subsampled using an equidistant grid over the genome, and an averaging sliding window is applied to this grid to determine a final classification score at a particular position.

wide window as input, that covers a significant genomic region. To evaluate the effectiveness of classifying a wider input region, the detection performance is tested for various window widths. The new model, due to its increased complexity, is more prone to overfitting on the training data. To reduce the effects of overfitting, the models are trained with a larger training dataset, evaluating the implications that this has on the classification and detection performance of the trained model.

3.2.3 Post-processing

When applying a sliding window with overlap to detect a selective sweep along a genome, multiple windows likely contain SNPs lying within the region affected by the selective sweep. As such, it can be expected that multiple neighbouring windows classify positively due to the selective sweep. By combining the neighboring positively classified windows, it is possible to perform selective sweep detection with increased robustness. When this method is applied, false positive detections with erroneous positive scores, which do not have neighbouring positive classifications, will be filtered out. Combining the classifications is done by means of an averaging sliding window over classification outputs. The optimal width of the averaging sliding window depends on how many consecutive classifications contain SNPs that are affected by the selective sweep. This, in turn, depends on the width of the selective sweep (in basepairs), the input width to the classifier in SNPs, and the density of the SNPs in terms of basepairs. Because the SNP density is affected by confounding effects, the width of the averaging sliding window is kept constant in terms of basepairs, not in terms of SNPs. this implies that the range of SNPs covered by the averaging window depends on the local SNP density. The averaging sliding window width for optimal detection performance is empirically determined. Potentially this width can be derived analytically by identifying the effect of population parameters and confounding factors, such as recombination hotspots, on selective sweep width.

Figure 3.3 illustrates the framework used for efficient selective sweep detection. The three-stage approach uses a model with dilated convolutions to perform fine-grained detection. In data preparation, the derived allele frequency and SNP distances are computed, requiring only these two vectors to be passed to the model, decreasing the amount of data to be transferred and processed. After obtaining the dense inference output from the model, outputs from the dense model are sampled based on a grid on the genome that is equidistant in terms of basepairs. This implies that the distance between each point on the grid is equal in terms of basepairs, not in terms of the number of SNPs between them. An averaging sliding window is applied over the grid, generating an output at each grid position. The final output is used to determine the presence of a selective sweep within the scanned genomic region.

3.2.4 Detection performance metrics

There is no strict definition on how to define the precision of a selective sweep detection algorithm since the width of the selective sweep is unknown. Previously, the success rate has been used as a metric to evaluate detection precision in this context. This metric takes the greatest classification score within a genomic sequence as the location where the sweep is detected. It then evaluates how many detected selective sweeps lie within a fixed radius from the center of the selective sweep [1][69]. One problem with this approach is that the result of this metric is highly sensitive to the radius that is used to consider the detection a success. Since the width of the sweep is unknown, defining a success radius that certainly corresponds to evaluating the precision of a model is not possible. In the same works that apply the success rate, the average distance of detections to the center of the selective sweep is used as a metric. This metric is not sensitive to a particular threshold, however, the metric can be misleading since an erroneous detection that is located at an outer edge of the genome has a significant negative effect on the total distance score. For this reason, it is more insightful to compare the precision of different models by means of a density plot. This plot uses the positions with the highest score for each detection, to calculate the density of detections over the simulation length. The density indicates the number of highest scores per basepair at a particular position. The density plot demonstrates how frequently selective sweeps are detected far from the center of the sweep, and at what distance these erroneous detections are made. Furthermore this plot gives insight into the width of the genomic region where the selective sweep is frequently detected.

Detection performance is evaluated by scoring each simulation according to the maximum classification output, after post-processing, along the simulated genome. The simulations are then categorized as selective or as neutral, depending on their score. If the score is above a given threshold, the simulation is considered to feature a selective sweep. The number of false positives and false negatives, when applying this method, is highly sensitive to the selected threshold, and the accuracy is likewise sensitive to this parameter. For this reason, the accuracy of a model, using only one specific threshold, is not a good indication of the performance of a model. The optimal threshold to evaluate the performance of the detection model will depend on the requirements of the application. For this reason, the receiver operating characteristic (ROC) curve is used instead. This curve evaluates the performance of the model in terms of true positive rate (TPR) and false positive rate (FPR), at each possible threshold. From this curve, another metric, named the area under curve (AUC), can be computed. The AUC integrates the ROC curve, providing a single statistic indicative of the model's ability to discriminate neutral and selective simulations, without specifying a particular threshold.

3.3 Evaluation

3.3.1 Experimental setup

Neutral SNP data, SNP data containing a selective sweep, and data containing several different confounding factors is generated using the simulation software ms [26], mssel (kindly provided by R. R. Hudson), mbs [55] and msHOT [25]. The same simulation parameters used to generate the data for chapter 2 are used. The genomes are 100 kbp in length, and for the selective simulations, the location of the selective sweep is at 50 kbp. For training, three different datasets per evolutionary scenario are generated, datasets consisting of 1k, 10k, and 50k neutral and selective simulations. When training, a split of 15% of the training data is used for validation. The datasets for testing consist of 1k neutral and 1k selective simulations. Each simulation features 128 samples. Using these simulations, SNP matrices with a width of 32, 64, 128, 256, and 512 are extracted. For training, the SNP matrix windows are centered around the location of the selective sweep.

Pytorch [45] is used to construct and train the neural network models. Each model is trained for 50 epochs at a learning rate of $0.5 \cdot 10^{-3}$, this learning rate is empirically determined and shows reliable training of each model. The models are trained using mini-batches of batch size 8 and the Adam optimizer is used [34]. After each epoch, the validation loss is computed. The model state after the epoch resulting in the lowest validation loss is saved and used for inference. To evaluate the model classification and detection performance, the ROC curve and the AUC are used. The detection precision is evaluated by plotting the density of detected selective sweeps.

3.3.2 Detection performance

Since most other methods do not separately evaluate detection performance, but rather focus only on classification metrics, the model with the best classification performance is used as a baseline. In chapter 2 figure 2.8 shows that FAST-NN yields the best classification accuracy and has the lowest execution time compared to other CNN-based selective sweep classification methods, hence the FAST-NN model, and the FAST-NN model with additional CNN layers, dubbed FASTER-NN (a reverse abbreviation of Neural Network for Robust and Efficient Tracing of Selection using Allele Frequencies) are trained on the 6 datasets as outlined in table 2.2. Both models are trained on each dataset using the 1K, 10K, and 50K neutral and selective training samples. Table 3.1 shows the AUC scores for classification, for an input size of 128 DAFs of 128 samples. Figure 3.2 shows the AUC scores for detection using the same input size.

Table 3.1: Classification AUC for both models, trained using 1k, 10k and 50k training samples from each class

	Model	D1	D2	D3	D4	D5	D6
1k	FAST-NN	1.0	0.960	1.0	0.984	0.904	1.0
	FASTER-NN	1.0	0.971	1.0	0.989	0.858	1.0
10k	FAST-NN	1.0	0.966	1.0	0.991	1.0	1.0
	FASTER-NN	1.0	0.976	1.0	0.991	1.0	1.0
50k	FAST-NN	1.0	0.968	1.0	0.993	1.0	1.0
	FASTER-NN	1.0	0.978	1.0	0.992	1.0	1.0

Table 3.2: Detection AUC for both models, trained using 1k, 10k and 50k training samples from each class

	Model	D1	D2	D3	D4	D5	D6
1k	FAST-NN	1.0	0.872	1.0	0.966	0.979	1.0
	FASTER-NN	1.0	0.878	1.0	0.968	0.927	1.0
10k	FAST-NN	1.0	0.887	1.0	0.980	1.0	1.0
	FASTER-NN	1.0	0.899	1.0	0.975	1.0	1.0
50k	FAST-NN	1.0	0.894	1.0	0.982	1.0	1.0
	FASTER-NN	1.0	0.907	1.0	0.978	1.0	1.0

From table 3.1 and 3.2 it can be concluded that training using a greater number of training samples always leads to better or equivalent performance compared to using fewer samples. The performance on dataset D2 and dataset D4 is further evaluated in more detail, since the classification and detection performance metrics on the other datasets are optimal in the current configuration, and thus require no further optimization. The effect of varying the input width on the performance of FAST-NN and FASTER-NN is investigated. Given the wide receptive field of FASTER-NN, this model is expected to benefit more from processing wider samples.

Figure 3.4 illustrates the effect of the number of DAFs per input (equivalent to the input width) on the classification performance and AUC of both models, on datasets D2 and D4. Figure 3.4 shows that the accuracy for dataset D2 using FASTER-NN is nearly unchanged when altering the input width. For the original FAST-NN model, the accuracy drops rapidly given a wider input. On dataset D4, the accuracy for both models increases as the input width increases. By only evaluating the accuracy, it would appear that using wider input does not improve the performance of either model on dataset D2. This is misleading since the AUC of FASTER-NN does increase as the input width increases. Conclusively, the classification performance of FASTER-NN improves on all datasets given a wider input. The classification performance of FAST-NN drops on dataset D2, but improves on dataset D4.

Figure 3.5 shows the detection performance metrics for FAST-NN and FASTER-NN on various input widths. The performance follows the same trend as observed when evaluating the classification metrics. For dataset D2, the performance of the FAST-NN model drops, and the performance of FASTER-NN improves. This result is reflected in the classification AUC for dataset D2 but does not show in the classification accuracy for dataset D2. For dataset D4, performance scales positively to input width for both models. Contrary to what is observed for classification, the FAST-NN model outperforms the FASTER-NN model on dataset D4, both in terms of detection AUC and TPR. A possible explanation for the attenuated detection performance of the FASTER-NN model on dataset D4 is that this can be attributed to the fact that all models have been trained with data where the center of the selective sweep is at the center of the window. In the case of FAST-NN, this does not limit the robustness for detection, since FAST-NN implements global average pooling over the spatial dimensions. Due to this, in combination with the narrow receptive

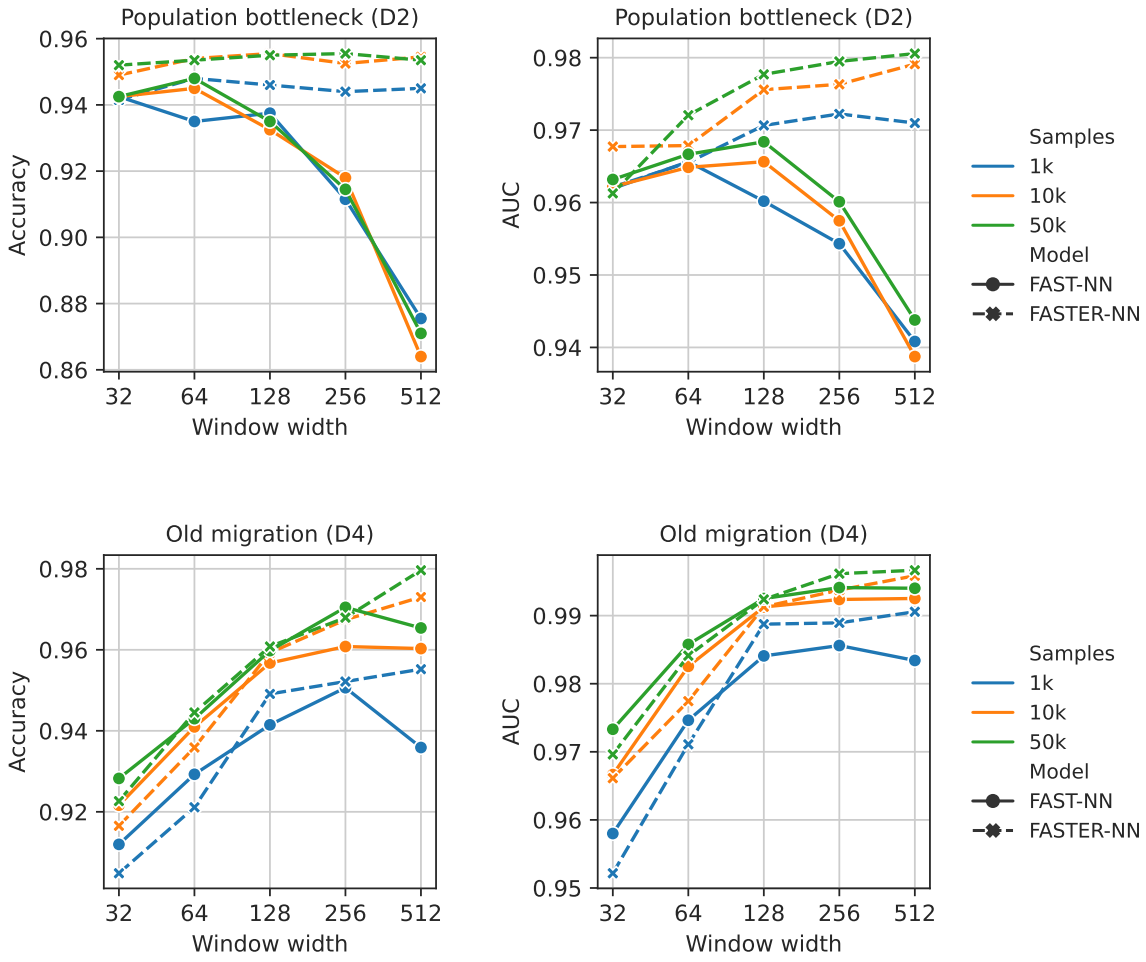


Figure 3.4: Accuracy and area under ROC curve (AUC) for classification of dataset 2 and 4 using FAST-NN and FASTER-NN, trained using 1k, 10k and 50k training samples in each class.

field before the average pooling layer, FAST-NN cannot distinguish between a selective sweep in the center or at the edge of the input. This is not the case for FASTER-NN, since this model does not use global average pooling, but passes the output of the convolutional layers directly to a fully connected layer. FASTER-NN can identify that all training data contains the selective sweep located around the exact center of the input data, and is trained to only classify these samples positively. This limits the number of inputs where the model detects the selective sweep, reducing robust detection performance. To confirm this, both models for dataset D4 have been retrained on 50k samples, but instead of centering the samples around the center of the selective sweep, the input is randomly translated by an offset between -7500 and 7500 bp. Figure 3.6 shows the ROC curve for detection of both models trained with and without random translations.

In figure 3.6, the FAST-NN model exhibits slight improvements when using randomly shifted training data, plausibly due to the increased data variation introduced by the random translations. The FASTER-NN model, however, improves significantly. As a result, the performance of both models when using random translations is nearly on par, with FASTER-NN slightly outperforming the FAST-NN model at an FPR below 0.04. This demonstrates that FASTER-NN is prone to learning the position of the selective sweep on the input data, attenuating the detection performance. Using random translations on the training data mitigates this problem. This method can also be applied to the other datasets, potentially improving the detection performance on these datasets as well.

3.3.3 Detection precision

For detection, the model performance is not measured only by evaluating the AUC and TPR. An ideal detection model is also able to specify the location of the selective sweep with fine-grained

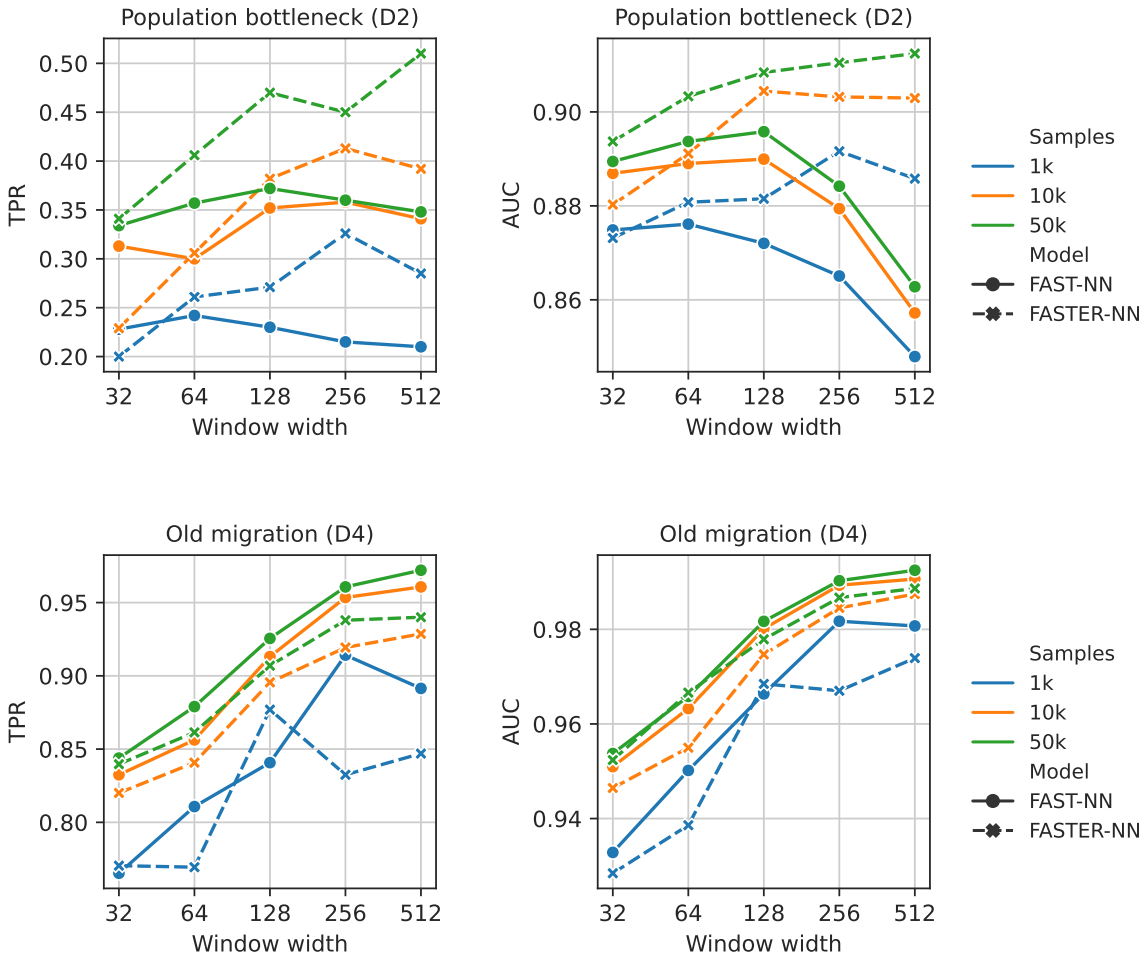


Figure 3.5: TPR at 0.05 FPR and area under ROC curve (AUC) for detection on datasets D2 and D4 using FAST-NN and FASTER-NN, trained using 1k, 10k and 50k training samples in each class.

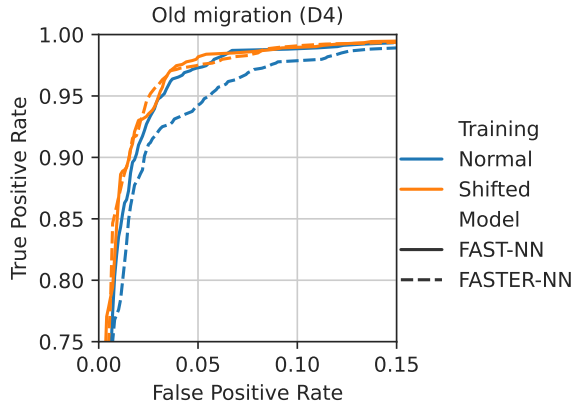
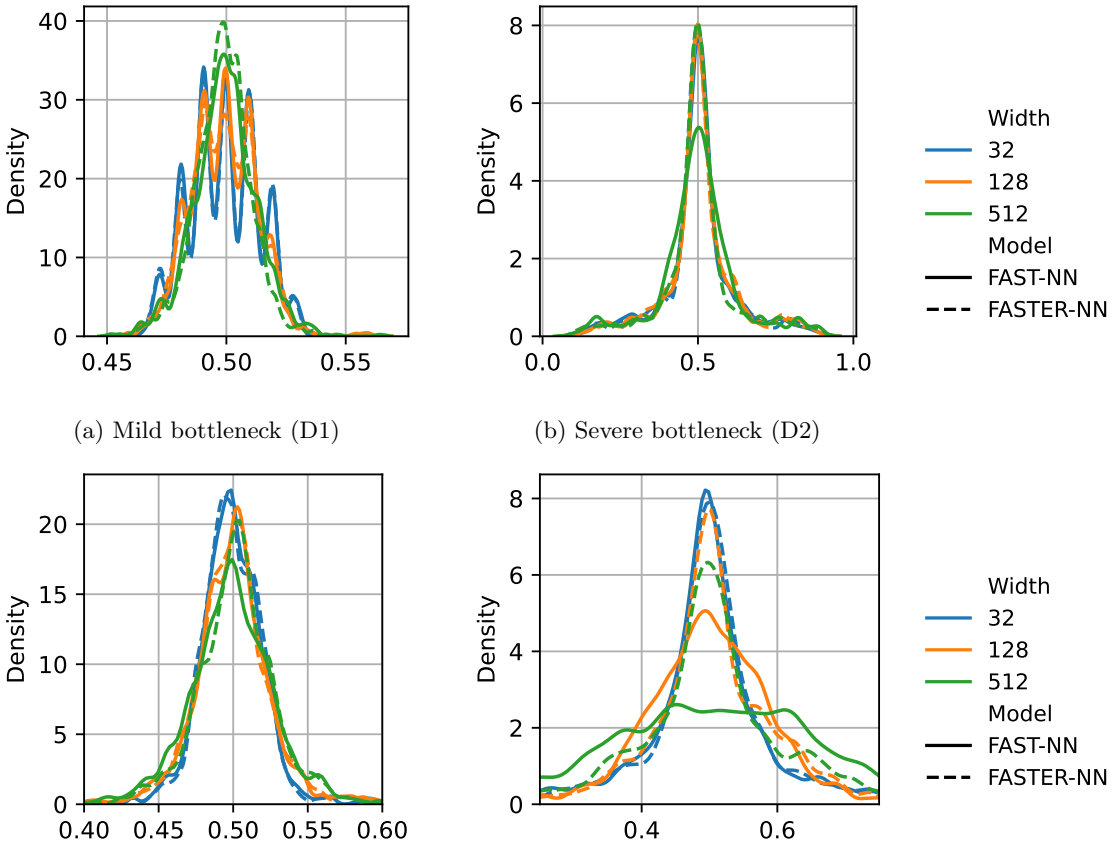


Figure 3.6: ROC curve for FAST-NN and FASTER-NN trained on dataset D4 with, and without, randomly shifting training data.

precision. For each dataset containing a selective sweep density plots are generated, indicating the density of positions where the output score of the model is greatest. High precision is indicated by a narrow peak density at the position of the selective sweep (50 kbp), low precision will result in a uniform density along the genomic positions. The precision is evaluated using the models trained with 50k neutral and selective samples since these models score the greatest detection AUC and TPR.



(a) Mild bottleneck (D1)

(b) Severe bottleneck (D2)

(c) Recent migration (D3)

(d) Old migration (D4)

Figure 3.7: Density plots of detections for datasets D1 through D4 using FAST-NN and FASTER-NN for various window widths.

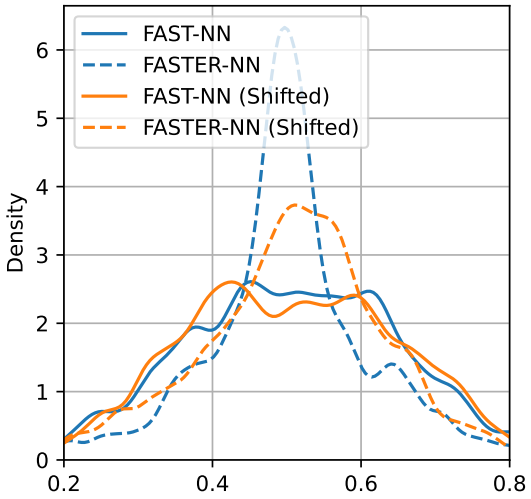
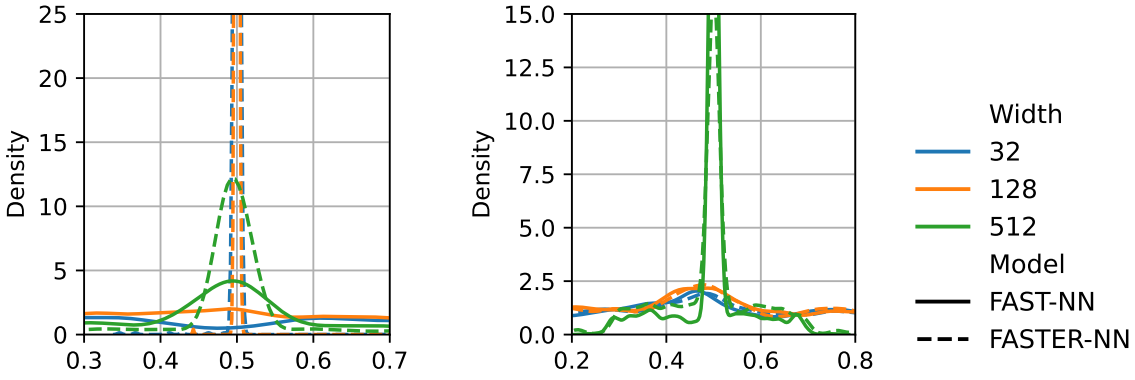


Figure 3.8: Density plot for of detections on dataset D4 (Old migration) with shifted training data.

Figure 3.7a through 3.7d show the density plots for the FAST-NN model and the FASTER-NN model, for dataset D1 through D4. For FAST-NN, on datasets D1 through D4, the precision generally drops as the input width increases. This is expected, considering that FAST-NN takes the average over the spatial dimensions, which, as previously discussed, leads to it being unable to distinguish the location of the selective sweep within the input. In this case, when using wider input data, samples further from the selective sweep will be classified positively, decreasing the precision. Similar results are observed when evaluating FASTER-N, for datasets D1, D3, and D4,

this model also has a lower precision when using wider inputs. For datasets D3 and D4 this effect is less significant compared to FAST-NN. When applying the random translation over the training samples, to ensure a high TPR, the precision of FASTER-NN drops to a score similar to that of FAST-NN, shown in figure 3.8. In contrast to the other results, the precision of FASTER-NN for dataset D2 increases when using wider windows. The results of datasets D1 through D4 show that for these datasets the precision of FASTER-NN is equal to or better than the precision of FAST-NN. Since randomly translated training samples increase the detection TPR, but decrease precision, it is possible to use a model trained with randomly shifted samples to determine whether a genomic sequence contains a selective sweep, and to consecutively use a model trained without shifted samples to pinpoint the location of the selective sweep.



(a) Low intensity recombination hotspot (Dataset D5)

(b) High intensity recombination hotspot (Dataset D6)

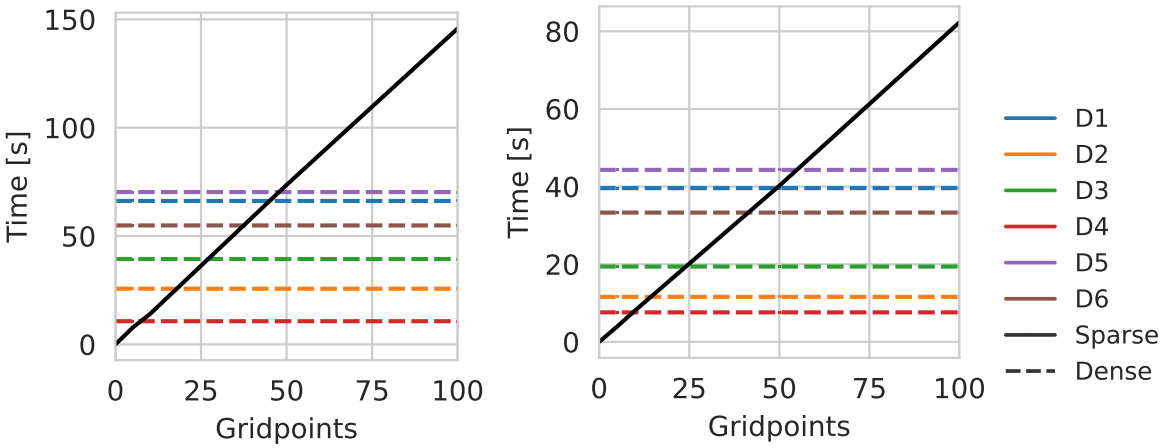
Figure 3.9: Density plots of detections on datasets D5 and D6.

The precision on datasets D5 and D6 for different input widths differs from what is observed for other datasets. For dataset D6, the density of detections does not indicate high precision for both models when using an input of 32 and 128 DAFs. When the input width increases to 512 DAFs the precision of both models becomes very high. A similar pattern is observed for dataset D5, but only when using FASTER-NN. For dataset D5, FASTER-NN has an extremely fine-grained precision when using an input width of 128 DAFs. The precision for an input of width 32 is slightly lower and for an input of width 512 is much lower, albeit still relatively precise compared to other datasets. The precision of FAST-NN remains much lower for all input widths. Datasets D5 and D6 contain simulations of a selective sweep within a recombination hotpot. The recombination hotspot of dataset D6 has a higher intensity than the recombination hotspot of dataset D5. To explain observed behaviour in detection precision, it can be hypothesized that there is an optimal input width, in terms of basepairs, for classifying a selective sweep. This optimal width depends on the simulation parameters, and would ideally capture all SNPs affected by the sweep, and no SNPs that lie outside the affected range. When detecting a selective sweep within a recombination hotspot, the SNP density is not only affected by the sweep but also by recombination. A higher recombination intensity will result in a higher SNP density, and when there is a higher SNP density, to capture the same width in terms of basepairs, a larger number of SNPs needs to be sampled. This can explain why dataset D6 (high intensity recombination) requires a wider input than dataset D5 (low intensity recombination) to achieve high precision.

3.3.4 Detection efficiency

When performing detection, an efficient sliding window approach is used to maximize data reuse between windows. A dense model is implemented to ensure that all positions along the genome are scored when using this sliding window. This model uses dilated convolutions and pooling layers instead of layers with a stride greater than 1. Using a dense model increases the number of computations for a single input, but can increase efficiency when computing overlapping windows since no data needs to be reprocessed. Whether using a dense model decreases execution time depends on the intended number of grid points. A dense model is efficient for fine-grained detection but, compared to the regular model, may be less efficient when applied for coarse-grained detection. The difference in execution time between the regular and dense models is evaluated. The execution time of the dense model is independent of the number of gridpoints, since it always yields an

output for each possible grid position, the execution time of the regular model will increase when performing inference over a greater number of gridpoints.



(a) FAST-NN (b) FASTER-NN

Figure 3.10: Inference time of dense models and sparse models for different numbers of gridpoints, using a window width of 512.

Figure 3.10b shows that after a certain number of gridpoints, the dense implementation becomes more efficient than the sparse implementation. The number of grid points depends on the width of the sampling window and the dataset since each dataset has a different average number of SNPs per simulation.

3.3.5 Human genome scan

To demonstrate the ability of FASTER-NN to perform fast whole-genome scans, the model is used to scan the 22 human autosomes (from the 1000 Genomes project [18]) for purifying (negative) selection. The FASTER-NN model is trained using simulated data that represents the human genome. This data is generated using stdpopsim [37], which is a Python library that uses the simulation engine SLiM [24] to simulate realistic populations. The first training dataset that is generated is specified only on human chromosome 1, consisting of 10k neutral and 10k selective simulations for training, and 100 simulations of each class for testing. Because stdpopsim generates VCF files, the data needs to be converted to the file format that the dataloader of FASTER-NN expects. Preprocessing this data required 679 seconds of execution time. Training FASTER-NN on this dataset for 10 epochs took 397 seconds on a CPU, and resulted in a model performing at 98% test accuracy on the simulated testing data. The model is used to scan the human chromosome 1 using a window width of 128, generating 946,100 separate scores for the 946,228 SNPs in 5.74 seconds. The post-processing of the output values slightly deviates from the process illustrated in figure 3.3: to obtain the scores at the equidistant grid points, instead of sub-sampling outputs, all scores nearest to a grid point, in basepair distance, are averaged. This way all scores are used to reduce the effect of potential outliers. The resulting basepair-equidistant grid is processed using an averaging sliding window of width 7, as illustrated in figure 3.3. The post-processed output of the scan is shown in figure 3.11a. A second FASTER-NN model is trained on 500 neutral and 500 selective simulations, from each of the 22 human autosomes, resulting in a training set of 22k simulations. This model should be better specified to scan all 22 autosomes, not just a single chromosome. The model is tested on 50 neutral and 50 selective simulations from each autosome, achieving a test accuracy of 99.4%. To evaluate the misspecification of the model trained only on chromosome 1, this model is tested on the same test set, achieving an accuracy of 99.3%. This suggests that the misspecification of the model trained on a single chromosome is negligible. Converting the 12,157,787 SNPs from the VCF files into a format accepted by FASTER-NN took 2.3 hours on a CPU. Using FASTER-NN to scan the data took 66 seconds. The model trained on all autosomes is used to scan human chromosome 1 and to scan all 22 human autosomes. It only takes 66 seconds to scan all 22 autosomes (12,157,787 SNPs). Figure 3.11b shows the output of the model trained on all autosomes when scanning only chromosome 1. The scores in figure 3.11a and 3.11b are similar, suggesting that training on all autosomes did not have a major effect on the model. Figure 3.11c shows a Manhattan plot of all 22 human autosomes.

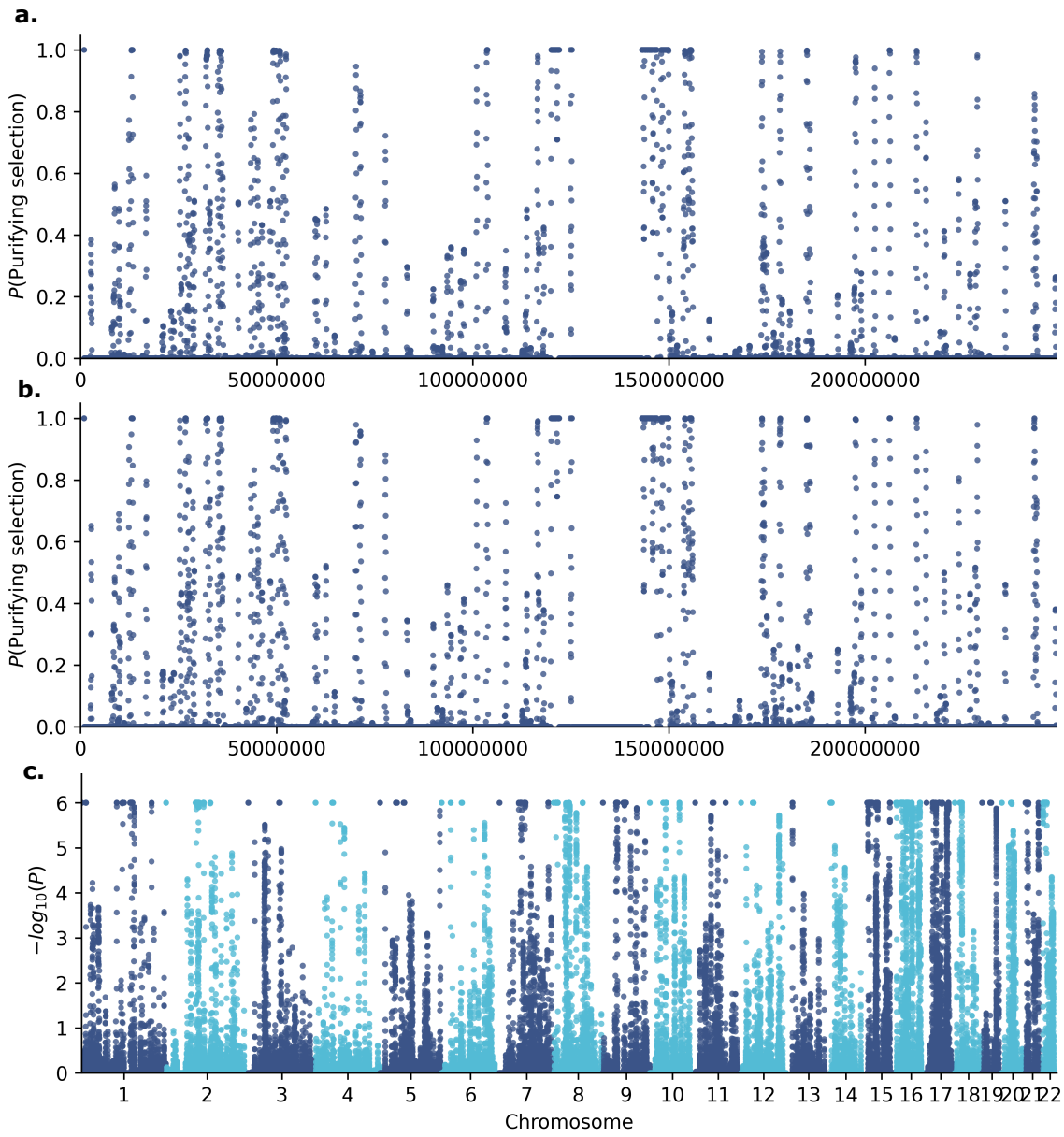


Figure 3.11: Scan of the human autosomes for purifying selection using FASTER-NN. **a-b.** Probability of purifying selection at each position along chromosome 1 using a FASTER-NN model that is trained using 10,000 neutral simulations and 10,000 simulations of purifying selection (**a.**) and using a FASTER-NN model trained on a dataset comprised of 500 neutral and 500 selective simulations from each of the 22 autosomes (**b.**). **c.** Manhattan plot of the 22 human autosomes scanned using the FASTER-NN model trained on data from all autosomes. An uncertainty of 10^{-6} is added to each score to bound the negative log probability of scores that are exactly 1.0, which occurs due to limited numerical precision.

3.4 Conclusion and discussion

This chapter introduced FASTER-NN, a CNN model designed specifically to enable efficient and effective selective sweep detection. FASTER-NN is based on the FAST-NN model, introduced in chapter 2, and uses the same principle of classification using allele frequencies and pairwise SNP distances. FASTER-NN is designed with detection using overlapping windows in mind, which, by reusing data between windows, enables the use of very wide input windows without significantly impacting execution time. FASTER-NN is designed with a wider receptive field than FAST-NN to enable learning wider patterns in data. By thoroughly evaluating the classification and detection performance of FASTER-NN and FAST-NN, it is shown that FASTER-NN mostly exhibits superior detection performance, especially when using wide input windows. Averaging detection outputs through a basepair-equidistant grid further improves the detection performance of FASTER-NN. By randomly shifting input data during training, FASTER-NN can avoid the effect of over-specification on selective sweeps in the center of an input window. It is observed that evaluating the models using detection performance metrics is more effective than solely assessing classification performance since better classification performance is not always reflected by better detection performance. By evaluating detection precision, it is found that window width can significantly impact detection precision, especially when detecting selective sweeps within recombination hotspots. FASTER-NN employs dilated convolutions to use an efficient sliding window algorithm, which enables fast whole-genome scans with wide input widths. A dense scan of 22 human autosomes can be performed in just over a minute.

In this chapter, the parameters for the framework used by FASTER-NN for reducing the effect of outliers, by averaging a basepair-equidistant grid, are determined empirically. Experimentally it is demonstrated that using wide input windows has a positive effect on detection performance. What remains to be uncovered is a rigorous approach to determine an effective input window width and averaging window size. Considering that the confounding factor and simulation parameters have a significant impact on the detection behaviour of the model, a two-step approach may be desirable. In this approach, the confounding effects and parameters of the simulation are first identified, and subsequently, these results are used to optimize the detection parameters. Considering the flexibility of CNN models, a two-step method may not be required. To improve detection performance, one could train a model to classify multiple effects in a simulation simultaneously, allowing the model to combine the information of different genomic effects. Chapter 4 explores the ability of a CNN to classify multiple effects, in particular for recombination hotspots, considering these have a profound effect on detection precision.

Recombination hotspot classification

4.1 Introduction

Genomes from various species possess localized regions of high recombination, also called recombination hotspots [47]. The increased recombination rates in these regions reduce linkage disequilibrium (LD) [10]. Classical likelihood-based methods have been developed to detect recombination hotspots based on LD [60], as well as CNN-based methods [13]. Although CNN-based methods are able to improve upon likelihood-based methods, achieving high classification accuracies, they require processing raw SNP data, which hinders scalability to large sample sizes.

In chapters 2 and 3 it is demonstrated that using derived allele frequencies (DAF), in combination with SNP distances, is effective for classifying and detecting selective sweeps. Using the DAF mitigates the scalability issues that CNNs processing raw SNP data face. DAF are informative of selective sweeps, since the metric preserves information regarding the site-frequency-spectrum and the polymorphism density. However, LD is not preserved when computing the DAF, since the linkage of SNPs can only be measured by analyzing individual samples, thus DAF DAF-based methods are expected to be less effective at classifying recombination hotspots.

CNN-based methods that have been proposed are focused on classifying a specific region-of-interest using SNP data, such as selective sweeps [20][56][43][68] [69] and recombination hotspots [13]. While the method of [13] is presented as a general framework for analyzing SNP data, it is evaluated as a method of classifying raw SNP data to identify recombination hotspots. In chapter 3 it is shown that the presence of a recombination hotspot can have a remarkable effect on the detection precision of a CNN-based classification model, affecting the window-width ideal for precise detection. In a two-step method, a recombination hotspot detection method could be applied to inform a selective sweep detection method, allowing it to adjust the window width. A two-step approach, however, seems redundant, since both the classification of selective sweeps and recombination hotspots can be done using CNN models.

Instead of a two-step approach, this chapter demonstrates that a CNN-based model can be trained to classify both selective sweeps and recombination hotspots simultaneously. Instead of using raw SNP data to preserve LD, enabling recombination hotspot classification, a hybrid approach between a DAF-based and a raw SNP-based classification method is presented. By computing the DAFs of multiple subsampled groups within the data, LD information is partially preserved, enabling hotspot classification while reducing the number of rows that need to be processed. This chapter makes the following contributions:

- It is demonstrated that selective sweeps and recombination hotspots can be classified using a single model, making a two-step approach obsolete.
- By using a hybrid approach between DAF-based classification and raw SNP-based classification, scalability is improved, preserving the ability to classify recombination hotspots.
- The effect of reordering samples before subsampling DAF groups is explored, demonstrating slight improvements in recombination hotspot classification, especially when using few DAF groups.

The combined classification performance of the new model is tested on datasets featuring recombination hotspots with low and high recombination rates (D5 and D6 from table 2.2, respectively). Using grouped DAFs substantially improves recombination hotspot classification compared to FASTER-NN, even when reducing the number of rows from 128 to 8, significantly reducing computational complexity, and improving the scalability of the method to large datasets.

4.2 Method

4.2.1 Grouped DAF model design

Linkage disequilibrium (LD) is a measure of correlation between mutations of different alleles. In a purely neutral scenario, it is assumed that allele mutation rates are fully independent. LD occurs when this independence is perturbed. Measuring the correlation between mutations of alleles requires observing multiple samples. Derived allele frequencies (DAFs) do not retain information regarding LD since all samples are combined. Recombination hotspots are identified by the measure of LD, making DAF-based models unsuitable for recombination hotspot classification. However, processing raw samples leads to scaling issues when processing whole genomes for a large number of samples. To mitigate the scalability issue, whilst retaining most of the LD information, a hybrid approach is proposed. By subsampling groups of samples from the raw SNP data, and computing separate DAFs for each subsampled group, information regarding linkage between SNPs is partially conserved. The information on LD is conserved in the difference between DAFs from different subgroups.

Given that the combination of information regarding DAFs of different subgroups yields insight into LD, and thus improves the ability to identify recombination hotspots, a CNN-based model is designed that combines these subgroups at multiple stages in the network. Considering the subsampling of rows to generate the subgroups is random, the subgroups should be processed in a manner that is invariant to the order of the groups. As such, each subgroup is processed identically by considering the subgroups as a batch dimension. Likewise, the operation to combine the subgroups should be invariant to the order of the groups, which can be done by using pooling operations. Figure 4.1 shows the Grouped Pooling Block that is designed to perform the combination of the subgroups. The block uses both average and max pooling to combine channels from different DAF groups. The averaged and max pooled channels are concatenated to the channels of each DAF group, adding additional channels to the output of the Grouped Pooling Block. To limit the number of additional channels, a fixed group of P channels is selected for pooling. For an input composed of C channels, this results in an output of $C + (2 \cdot P)$ channels. The Grouped

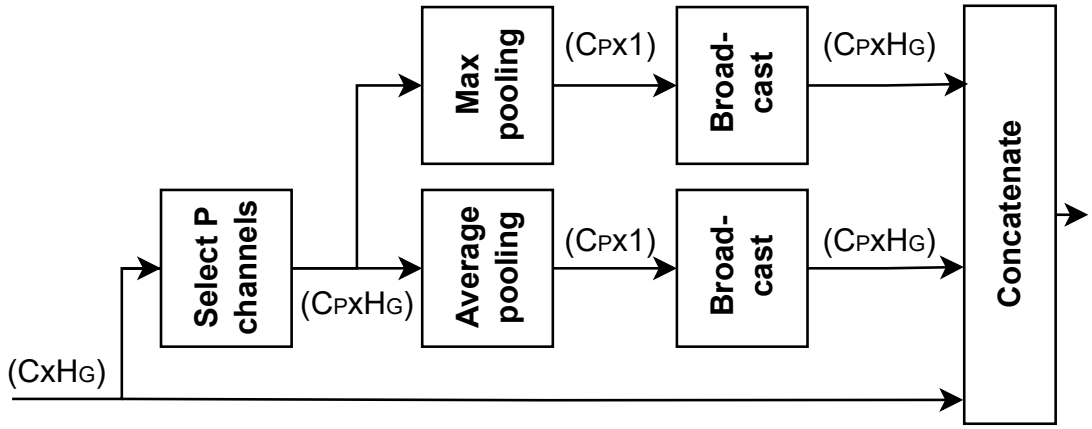


Figure 4.1: Diagram of the Grouped Pooling Block. This block uses average pooling and max pooling to combine information between groups, allowing for the model to learn patterns between different (groups of) samples. C is the number of input channels, from which P channels are selected for pooling. Dimensions between parentheses indicate the data shape between each node.

Pooling Block extends the FASTER-NN architecture, by placing one block after each convolutional layer, except the final layer. After the final convolutional layer, before the fully connected layer, the groups are combined using global average pooling over the group dimension. Setting P to 4 channels for pooling by the Grouped Pooling Block was empirically determined to give satisfactory results while limiting the additional computational complexity, as it only adds 8 additional channels. Batch normalization [27] is applied after each convolutional layer, as it is observed that this significantly reduces training time.

4.2.2 Sample reordering

Various previous works have addressed the fact that reordering raw SNP data can lead to improved performance for selective sweep classification [68][20] [56]. By reordering data, specific features of

selective sweeps can be made more pronounced. By reordering columns, information is added along the vertical axis of the SNP matrix, which is otherwise uninformative. However, as discussed in chapter 3, reordering data based on a full SNP matrix introduces a window-based preprocessing step, that hinders data reuse for efficient fine-grained detection. Nevertheless, examining the effect of reordering on the performance of a model can inform about the potential performance gain that can be achieved through reordering.

When processing SNP data using subsampled DAF groups, the effectiveness of preserving information regarding LD depends on the uniqueness of each group. If the distribution of samples in each group is a perfect representation of the distribution of all samples, the DAF of each group is identical. In this case, using DAF groups should be not more effective than using a single DAF for the entire population. Maximizing the difference between the samples in each group maximizes the difference in DAF between the groups, retaining more information on LD. To increase the difference between DAF groups, the rows in the SNP matrix can be reordered based on a matrix that determines the difference between each row. To implement this, the first row in a SNP matrix is used as a reference, and the hamming distance from the first row to each other row is computed. The rows are then sorted, placing the rows most similar to the reference row at the top of the matrix, and the most distant rows at the bottom. When sampling the subgroups to compute DAFs, the consecutive rows are grouped together, ensuring that rows with similar hamming distances are combined.

4.2.3 Multi-label classification

Even though different genomic effects, such as selective sweeps and recombination hotspots, can lead to similar patterns in SNP data, and can affect the ability of a CNN-based model to detect specific effects, the detection of these genomic effects is currently done using separate models. This is suboptimal because it requires performing inference over the same data multiple times to identify multiple genomic effects. Moreover, since the patterns that these genomic effects impose on the SNP data are related, creating a classifier that can generate both scores simultaneously could perform better in differentiating between the two genomic effects. Multi-label classification is used to enable the classification of multiple effects using a single model. To classify N genomic effects, the model has $2 \cdot N$ outputs, which are reduced to N classification scores using the softmax function on a $(2, N)$ output matrix. To train the model, backpropagation is performed once for each pair of outputs using the combined loss of both outputs. For combined selective sweep and recombination hotspot classification, each model in this chapter has 2 pairs of outputs, one for scoring the presence of a selective sweep, and one for scoring the presence of a recombination hotspot.

4.3 Evaluation

4.3.1 Experimental setup

The datasets that are generated contain raw SNP data from 128 samples, with a width of 512 SNPs, resulting in SNP matrices of 128×512 SNPs. Each dataset features four classes: neutral windows, windows featuring a recombination hotspot, windows featuring a selective sweep, and windows containing a selective sweep within a recombination hotspot. Training datasets contain 50,000 windows from each class and testing datasets contain 1,000 windows from each class. Each window has two labels to indicate the presence of a selective sweep and a recombination hotspot. The selective sweep and recombination hotspots are located in the center of each region. From the training dataset, 15% of the data is used for validation. One training and testing dataset is generated with high intensity recombination hotspots, and one training and testing dataset is generated with low intensity recombination hotspots.

Each CNN is implemented in Pytorch [45], and trained for 10 epochs at a learning rate of $0.5 \cdot 10^{-3}$. For training a batch size of 8 and the Adam optimizer is used [34]. The rows of the SNP matrices are randomly shuffled during training (unless reordering is applied). When training, a model is saved after the epoch with the highest validation accuracy.

4.3.2 Combined classification performance

SweepNet (with SNP distances), FAST-NN, FASTER-NN, and six variations of Grouped Pooling Blocks extended FASTER-NN are trained. Three variations of the extended FASTER-NN model

use 8, 32 and 128 groups without reordering, and three use the same number of groups but with reordering. Henceforth, the models without and with reordering are referred to as FASTER-NN-G-N and FASTER-NN-R-G-N, respectively, where N denotes the number of groups that the models use. Each model is trained and tested once for each dataset. Table 4.1 shows the classification accuracies of the models on both datasets, as well as the inference time of each model. When determining the accuracy of the multi-label classification models, a prediction is only counted as correct when both labels are correctly predicted. The accuracies clearly show the effect of using grouped DAFs for combined selective sweep and recombination hotspot classification. Using a larger number of groups increases the accuracy, at the cost of additional execution time. Using reordering has a positive effect on the classification accuracy when using 8 groups, but has no significant effect when using 32 or 128 groups.

Table 4.1: Classification accuracy of each model on datasets using low intensity and high intensity recombination hotspots, and CPU inference time of each model.

Model	Accuracy low recombination rate		Accuracy high recombination rate		CPU Inference time (seconds)
	No reordering	Reordering	No reordering	Reordering	
SweepNet	0.878	-	0.997	-	1065.7
FAST-NN	0.873	-	0.964	-	8.0
FASTER-NN	0.865	-	0.968	-	7.0
FASTER-NN-G-8	0.963	0.976	1.00	1.00	18.2
FASTER-NN-G-32	0.989	0.989	1.00	1.00	65.8
FASTER-NN-G-128	0.994	0.992	1.00	1.00	250.5

To further examine which genomic patterns the models fail to classify correctly, figure 4.2 plots the confusion matrix for each model on the low recombination dataset. The high recombination dataset confusion matrices are not shown since most classifiers have (near) perfect accuracy on this dataset. From the confusion matrices, it becomes clear that the models are almost exclusively confusing the neutral genomic regions and the genomic regions that contain only a recombination hotspot. The models that have a higher classification accuracy feature an improved ability to distinguish these regions. Interestingly, the presence of a selective sweep makes it easier for the models to identify the presence of a recombination hotspot.

4.4 Discussion and conclusion

The flexible nature of CNN-based models makes it possible to combine classifying multiple genomic effects through a single CNN, this makes it unnecessary to perform inference over the same data using multiple models. It has been demonstrated that, through multi-label classification, a single model can be used to effectively classify the presence of a selective sweep and a recombination hotspot. The combined effect of a recombination hotspot and a selective sweep is not fully understood from a theoretical point of view, however, in chapter 3 it is shown that the presence of a recombination hotspot can affect the optimal window width for precise selective sweep detection. By enabling a model to classify both a selective sweep and a recombination hotspot simultaneously, the presence of the recombination hotspot can potentially be taken into account by the model, enabling precise detection that is robust to the presence of a recombination hotspot.

By processing grouped DAFs consisting of various numbers of samples it is shown that information regarding LD can be partially retained while reducing the number of rows to be processed by the CNN. The accuracy of the model is improved when using a larger number of groups, but this also generates a greater amount of data to store and process, increasing execution time. Nonetheless, by reducing a SNP matrix from 128 raw samples to 32 DAFs the classification accuracy decreases by only 0.5% whereas execution is 3.8x faster when performing CPU inference. The grouped DAF method demonstrates a hybrid method between allele frequency-based classification and raw SNP-based classification. This is a step towards scalable recombination hotspot classification using CNN-based models, and can potentially improve the robustness of selective sweep detection using CNNs.

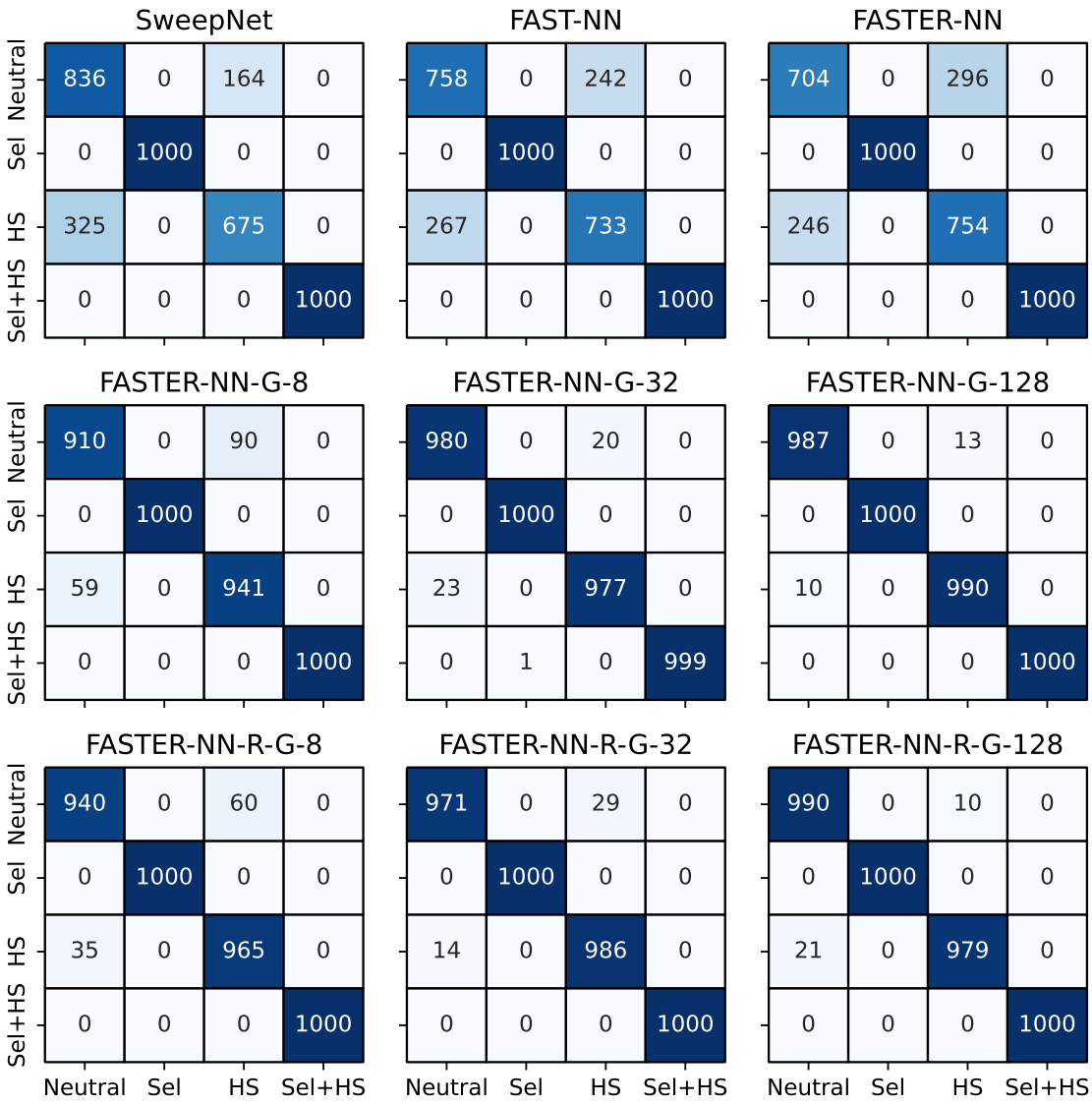


Figure 4.2: Confusion matrices for each model, where FASTER-NN-R denotes the grouped FASTER-NN model with reordering, and G-x denotes the grouped FASTER-NN model using x groups.

Hardware acceleration

5.1 Introduction

Selective sweep detection algorithms using CNNs to process raw SNP data or derived allele frequencies (DAF) outperform summary statistics-based selective sweep detection methods based on one or multiple signatures of selective sweeps [69][68]. In chapter 2 it is demonstrated that FAST-NN outperforms existing CNN-based detection algorithms, both in terms of accuracy and in terms of execution time. In chapter 3, by transforming FAST-NN to use dilated convolutions and employing data reuse to prevent redundant computations, selective sweep detection is further accelerated. This enables the use of these models for whole-genome scans at full precision within a reduced amount of time. Despite these optimizations, scanning a large number of genomic datasets can take a considerable amount of time. While selective sweep detection methods based on summary statistics have been accelerated using reconfigurable hardware [5][6][15], conventional CNN-based models designed for selective sweep detection have not been accelerated using FPGAs. By reducing the execution time of CNN-based models through hardware acceleration, the execution time to perform fine-grained detection, which increases detection precision, is more feasible.

FPGA accelerated convolutional neural network (CNN) architectures fall into one of two categories. Architectures either use large matrix-multiplication accelerators that are used as a shared resource by multiple layers within the network, or architectures pipeline all network layers, utilizing dedicated processing elements for each layer. When implementing the former approach the applications employ a systolic array to perform large matrix multiplications. The systolic array dynamically loads weights during runtime from external memory, and as such, can perform the computations required by different layers in the network architecture [67]. This type of implementation has the benefit of being flexible to almost any network architecture. However, it reduces throughput, since weights have to be loaded in from external memory, and the parallel execution of network layers is limited due to the systolic array being shared by multiple layers. A fully pipelined implementation employs dedicated accelerators for each layer in the network, each accelerator can be designed to specifically fit the requirements of a single layer in the network. This approach offers higher throughput since all layers in the CNN can process data in parallel [57]. The downside of this approach is that it lacks the flexibility of a general-purpose systolic array-based accelerator, and due to the limited number of resources on an FPGA, only small networks can be implemented in this manner.

Since floating-point operations require a considerable amount of resources on an FPGA compared to fixed-point operations, a neural network is converted to a quantized neural network (QNN) before it is implemented in an accelerator [65][22]. A quantized neural network stores weights and activations of a network as fixed-point numbers instead of floating-point numbers. Likewise, the arithmetic operations performed by the neural network are done on fixed-point numbers, decreasing the computational load and FPGA resource utilization required to perform the computations. Because of the discretization of the floating-point number in the process of quantization, there is a loss of precision that affects the output of the neural network [36]. Since no FPGA-based implementation of a CNN model for selective sweep classification exists, it is unclear what the effect of precision loss due to quantization is on the classification accuracy of the neural network.

FPGAs as hardware accelerators can offer high application-specific parallelization, but are constrained by the number of available hardware resources, and generally possess limited I/O bandwidth. FAST-NN performs selective sweep classification using DAFs and pairwise SNP distances, which is a compact representation of SNP data, with a constant data size per SNP position. Designing a model that implements a compact data format is informed by the fact that such a model is highly suitable to implement on an FPGA. Leveraging this compact data format, FAST-NN requires relatively few resources to implement its 1D CNN on an FPGA. Moreover, the use of

DAFs instead of raw SNP data decreases the amount of data that needs to be transferred to the FPGA, mitigating the I/O bandwidth as a potential bottleneck.

FAST-NN is implemented on an FPGA, achieving highly efficient selective sweep detection. The FPGA implementation of the FAST-NN model features the following contributions:

- Due to the small size of the FAST-NN model, partly realized by using a compact data format to represent SNP data, the amount of hardware resources required to realize the model on an FPGA is reduced. Hence, the FAST-NN model can fit on an FPGA at an 8-bit quantization precision. The 8-bit quantized version of the FAST-NN model is trained using QAT, which reduces the accuracy of the model by 0% for five tested datasets and 2.4% for one tested dataset.
- By using allele frequencies as a compact data format, a fine-grained detection algorithm that reuses data can be designed using only 1D convolutions. Due to the small size of this model, a fully pipelined hardware implementation of this model can be implemented on an FPGA, at an initiation interval of one at a clock frequency of 90 MHz. The hardware implementation leverages parallel processing of all layers in the CNN, and through a streaming architecture without any dataloading constraints, can load one SNP position per clock cycle. Scanning chromosome 1 of the human genome, hardware acceleration enables a theoretical speedup of 546x compared to a CPU implementation in Pytorch.

This work demonstrates that whole-genome scans using DAFs can be accelerated with FPGAs. The nature of genomic data makes FPGAs particularly suitable for this purpose. Scanning a genome requires convolutions over very wide input data (up to millions of SNPs), with a small amount of data per column. This makes a streaming FPGA architecture suitable for this application since it requires limited I/O but can benefit from high throughput and parallelism. The hardware architecture proposed in this chapter is specific to the model architecture that is implemented. In this chapter, it is only realized for the FAST-NN model architecture. The hardware design, however, can be adapted to other CNN architectures. Implementing the FASTER-NN model in this hardware architecture would require changing the number of layers, layer hyperparameters, and buffer lengths in the hardware design, but the component design and streaming architecture are identical.

5.2 Related work

In previous work, FPGA architectures have been designed that accelerate 2D CNNs in a pipelined manner for low latency inference [63] [28]. Data reuse in streaming architectures for 2D CNNs can be optimized by the use of line buffers [63]. Streaming dilated 2D convolutional neural networks have been further improved through optimizations in buffer memory arrangement [19]. Other works focus on accelerating 1D CNN models, both using a systolic array [61] and by streaming the input data [39].

Because it requires processing large datasets, FPGAs have widely been deployed in the acceleration of bioinformatics applications. In the analysis of phylogenetics FPGAs have been used to accelerate both the computation of the phylogenetic parsimony function [31][30][4] and the phylogenetics likelihood function [2][70][42]. Within the field of population genetics, multiple works have used FPGAs for accelerated detection of epistasis [64][44], and for accelerated computation of linkage disequilibrium (LD) [11].

The μ statistic, as used in the RAiSD selective sweep detection algorithm [1] combines the three known selective sweep signatures as a single metric. Through a streaming-based algorithm, an FPGA is implemented to accelerate the computation of the μ statistic [5]. Matching of SNP patterns to a pool of previously processed patterns is used to reduce processing time. Through Out-of-Core processing, memory requirements to run this detection algorithm are reduced. This implementation is further optimized by employing an optimized hashing algorithm to accelerate matching to known SNP patterns [6].

Since LD patterns as a standalone signature cannot predict the presence of a selective sweep, OmegaPlus [3] computes the ω statistic, which relates the LD patterns to the probability of detecting a selective sweep. Computing the ω statistic has been accelerated using an FPGA in a pipelined manner using floating point operations [15]. By combining this implementation with FPGA-based LD processing implementations, detecting selective sweeps using LD can be fully executed on reconfigurable hardware.

Previous work has focused on using FPGAs to accelerate the detection of selective sweeps using the classically known metrics that indicate selective sweeps. However, as mentioned in

chapter 2 and 3, convolutional neural network (CNN) based methods have demonstrated superior performance when it comes to selective sweep classification. As of now, almost no research has been done to accelerate a CNN-based model specifically for selective sweep detection on an FPGA platform. One implementation of a selective sweep classification model through a spiking CNN has been proposed [14]. Because the FAST-NN model from chapter 2 is relatively small, and only uses 1D convolution, it can be deployed to an FPGA in a fully pipelined manner using a streaming architecture.

5.3 Architecture

5.3.1 Quantization

During quantization, the weights and activations, with activations referring to the outputs of each layer in the network, are transformed from floating-point values to fixed-point values. The goal of quantization is to decrease the amount of data required to store the weights of a network and to reduce the computational resources required to perform inference, allowing for a more compact hardware design. Since quantization discretizes the weights and activations, it inevitably introduces a quantization error. In order to minimize the quantization error, it is essential that the weights and activations are scaled to an appropriate range. When performing n-bit quantization, the quantized integer can store 2^n values. Given that a group of weights or activations span the range $[a, b]$, the values must be scaled such that they range from -2^{n-1} to $2^{n-1} - 1$ in the case of a signed integer, and from 0 to $2^n - 1$ in the case of an unsigned integer. In case the values represented by a signed integer are not symmetric around 0, or in case the values represented by an unsigned integer contain negative values, a bias can be added after scaling. This bias is called the zero point and is the integer value that, when a previously quantized number is dequantized, equals 0. Following these requirements, computing the fixed-point representation of a floating-point number is done using equation 5.1.

$$x_q = \text{clip}(\text{round}(x \cdot s - z)) \tag{5.1}$$

The clip function clips a value to the range of numbers the fixed-point integer can represent (e.g. -128 to 127 for an 8-bit signed integer). Equation 5.2 denotes the scaling factor, and equations 5.3 and 5.4 denote zero point for quantization to signed and unsigned integers respectively.

$$s = \frac{2^n - 1}{b - a} \tag{5.2}$$

$$z = \text{round}(b \cdot s) + 2^{n-1} \tag{5.3}$$

$$z = \text{round}(b \cdot s) \tag{5.4}$$

The zero point z is rounded to ensure that a floating-point value of 0 is converted without any quantization error, which is important since a neural network often has an abundance of zero-valued activations. This quantization scheme is called affine quantization since it performs an affine transformation on the floating-point numbers. The special case where the z equals 0 is called symmetric quantization. Performing computations using symmetrically quantized values is more efficient because the zero point does not need to be taken into account [65].

To apply quantization to a neural network, the quantization parameters, s and z, need to be determined. The weights of the model are known before inference, so these can be quantized without additional steps. The activations of layers, however, depend on the input data. Therefore these quantization parameters cannot be based on the model itself, instead, these must be determined based on input data. Two common approaches exist to determine these values, dynamic quantization and static quantization [22]. When applying dynamic quantization, the quantization parameters s and z are dynamically adjusted to the data that is being passed through the network. To accomplish this, z and s are determined on the fly for each batch that is passed through a network layer. Dynamic quantization requires dynamically computing s and z for each batch, which, in turn, requires floating-point arithmetic. For this reason, dynamic quantization is not suitable for implementation on an FPGA. Static quantization on the other hand does not require floating-point arithmetic during inference. It calibrates the quantization parameters by observing the ranges of activations when running the model on training data and saves these parameters to use during inference. Static quantization can be improved through Quantization Aware Training (QAT). During QAT, the model trains while emulating the effect of quantization, allowing the weights of the model to become more robust to these effects. QAT typically reduces the performance degradation caused by quantization [36].

Since QAT generally results in a quantized model with the lowest loss of accuracy, the FAST-NN model is quantized using static quantization and trained using QAT. Weights in a neural network tend to be symmetrically distributed around 0, therefore symmetric quantization to signed integers is used to quantize the model weights. For the CNN layers that implement ReLU activations, the outputs are guaranteed to be distributed above 0. Thus, the activations of these layers are symmetrically quantized to unsigned integers. Finally, quantization and dequantization layers, also referred to as stubs, are added to the input and output of the network, in order to convert the floating-point inputs to quantized inputs, and the quantized outputs to floating-point outputs. The top diagram in figure 5.1 depicts the resulting system as it is implemented in Pytorch.

5.3.2 Buffer design

As mentioned previously, in 2D streaming CNN implementations on FPGAs, line buffers have been used in order to optimize data reuse [63]. These buffers are required to ensure that data in adjacent rows is simultaneously available. Because FAST-NN uses 1D convolution, it does not need to synchronize data access of input rows. To achieve an initiation interval of one clock cycle, each layer in the design must produce one output per cycle, and the input to the system needs to sustain one SNP position per cycle. One SNP position in the FAST-NN model comprises two 32-bit floating point values, the DAF and the SNP distance at that position. To enable each layer to compute one output per cycle, the input buffer to each layer should fit the kernel width of the corresponding layer. Since dilated convolutions are used, the kernel size can be computed using equation 5.5.

$$K = K_0 \cdot D - (D - 1) \tag{5.5}$$

Where D is the dilation of the kernel, and K_0 is the width of the kernel without dilation. Since the convolutional operation needs to access multiple values stored in the buffer at once, the buffers are implemented as multi-access shift registers of K depth, with a width of C quantized activations, where C is the number of input channels to the network layer. This buffer design is inspired by the approach of Liu et al. [39], which proposed a buffer design for a 1D CNN to process real-time non-concurrently available data. The bottom diagram in figure 5.1 depicts the resulting architecture, including the required buffer sizes. Through this design, one window of 128 SNP positions is classified in each cycle, where each cycle the new window is the previous window shifted by one SNP position.

5.3.3 Component design

Figure 5.2 shows the design of several of the functional components in the CNN. Since the fully connected layer can be generalized as a convolutional layer, it is not separately depicted. As the FAST-NN network only uses pooling layers with two inputs, the pooling component takes two inputs from the input buffer and propagates the greatest of the two. The quantization stub scales the input by the scaling factor, and since the input is exclusively positive, no zero point needs to be added. A value of 0.5 is added before the conversion to ensure correct rounding when converting to an integer. The dequantization stub first converts the outputs to floating-point values and then multiplies the resulting values by a scaling factor to transform the values to the expected output range.

The convolutional layers (and similarly the fully connected layers) comprise two main elements. Firstly, the convolutional operation is performed by multiplying the input values by the corresponding weights and accumulating the result. To prevent overflowing during accumulation, the accumulator has a width of 64 bits. Following the multiply-accumulation, the activations need to be requantized according to the activation scaling factor of the corresponding layer. To perform requantization, the scaling by the weights and the activation of the previous layer need to be undone, and the activation of the current layer needs to be applied. This can be done in a single operation through a requantization factor that can be computed beforehand. This factor is defined in equation 5.6.

$$s_r = \frac{s_a}{s_w \cdot s_i} \tag{5.6}$$

Here, s_a is the scaling factor of the activation of the current layer, s_w is the scaling factor of the weight used in the matrix multiplication, and s_i is the activation scaling factor of the previous layer. s_r is a floating-point value, so to ensure that the requantization component does not need to do floating-point arithmetic, s_r is quantized to a 16-bit unsigned integer. A scaling factor of 2^{16} is chosen since s^r is always less than 1, and the scaling can be undone simply by bit-shifting. The bias is then scaled by the same factor and added to the rescaled activation. After this, the scaling

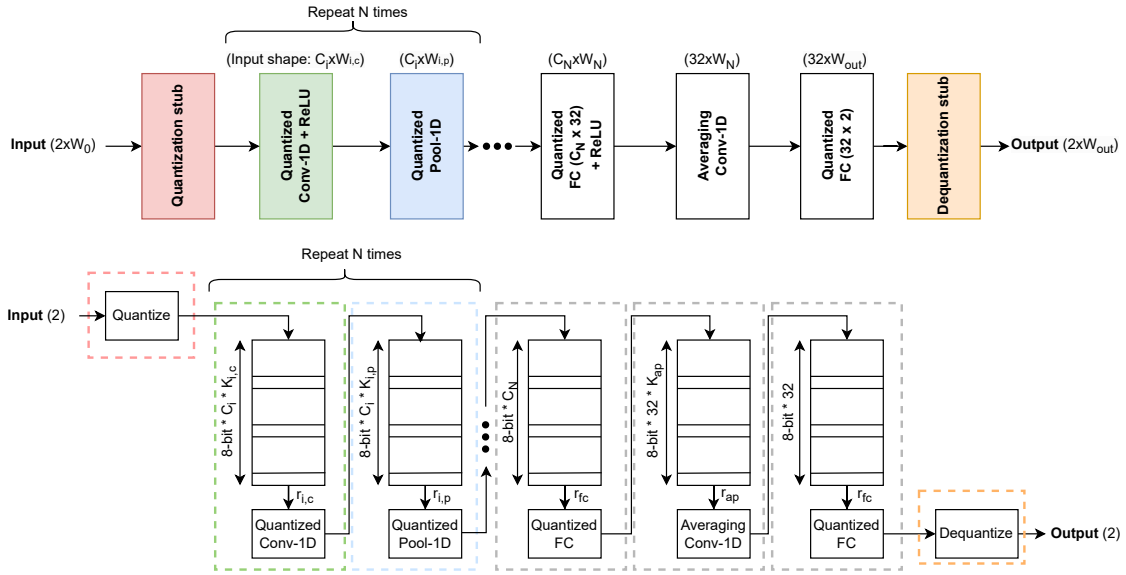


Figure 5.1: Functional block diagram of quantized CNN model in Pytorch (top) and block diagram of quantized CNN model as implemented in C++ and Vitis HLS, depicting functional blocks and memory buffers. The Python implementation processes a full input matrix at once, passing the complete output of one functional element to the next. The C++ and HLS functions compute the result of one matrix column per operation, pipelining the consecutive operations of the CNN. Multiple values are loaded from the shift-register buffers at once, denoted by the variable r_x. For FAST-NN, $r_{i,c}$ and $r_{i,p}$ are always 2, r_{fc} is 1, and r_{ap} depends on the DAF window width that the model assumes. A low-level implementation description of the (de-)quantization, convolutional, and pooling functions are shown in figure 5.2.

is undone by shifting 16 bits, and (if the activation quantization is not symmetric) the zero-point is subtracted. This is only the case for the final fully-connected layer. The ReLU activation function is implemented as a multiplexer which selects 0 if the activation is less than 0.

5.4 Implementation

Pytorch version 2.3 [45] is used to implement the quantized version of the dense FAST-NN model and to perform QAT. The FAST-NN model is trained from scratch using QAT. 8-bit precision is chosen for quantization, estimating that an 8-bit FAST-NN model will fit on the target device, whilst retaining high precision to introduce little to no performance loss.

The quantized parameters are exported to C++-style arrays, such that they can be imported by a C++ model. The CNN model is written from scratch in C++. To ensure that the HLS tool is able to optimize the hardware synthesis based on the C++ code, the layers in the network are called sequentially from the top-level function. Each layer is written such that the loops in the function can be fully unrolled, meaning that only the innermost loop contains operations. Each layer in the network only takes two parameters, which are the input stream and output stream variables. These variables communicate the input and output between layers. The buffers of the network layers are locally contained within each layer as a static variable.

To achieve an initiation interval of 1, the layers in the model use the HLS pipeline pragma, with an II of 1, this automatically unrolls all the loops to achieve the desired initiation interval. The top-level function uses the HLS dataflow pragma, enabling the top-level function to start processing one input per clock cycle. Synthesis shows that the design can run at an initiation interval of 1, with a total latency of 66 cycles. The quantization of the input introduces the most latency out of all model layers, with a latency of 15 cycles.

The Vitis HLS design targets an Ultrascale+ XCU250 FPGA on an Alveo U250 board. The design utilizes all four Super Logic Regions (SLRs). A total of 114,324 flip-flops are used and a total of 470,563 LUTs are implemented. The design utilizes 12,281 DPSs, which is nearly all DSPs available on the board. Table 5.1 gives a full overview of the resource utilization on the FPGA, aggregating the resources used on each SLR. Most DSPs are required by the 80 by 80 channel convolutional layers. Using convolutional layers with a reduced number of channels can drastically

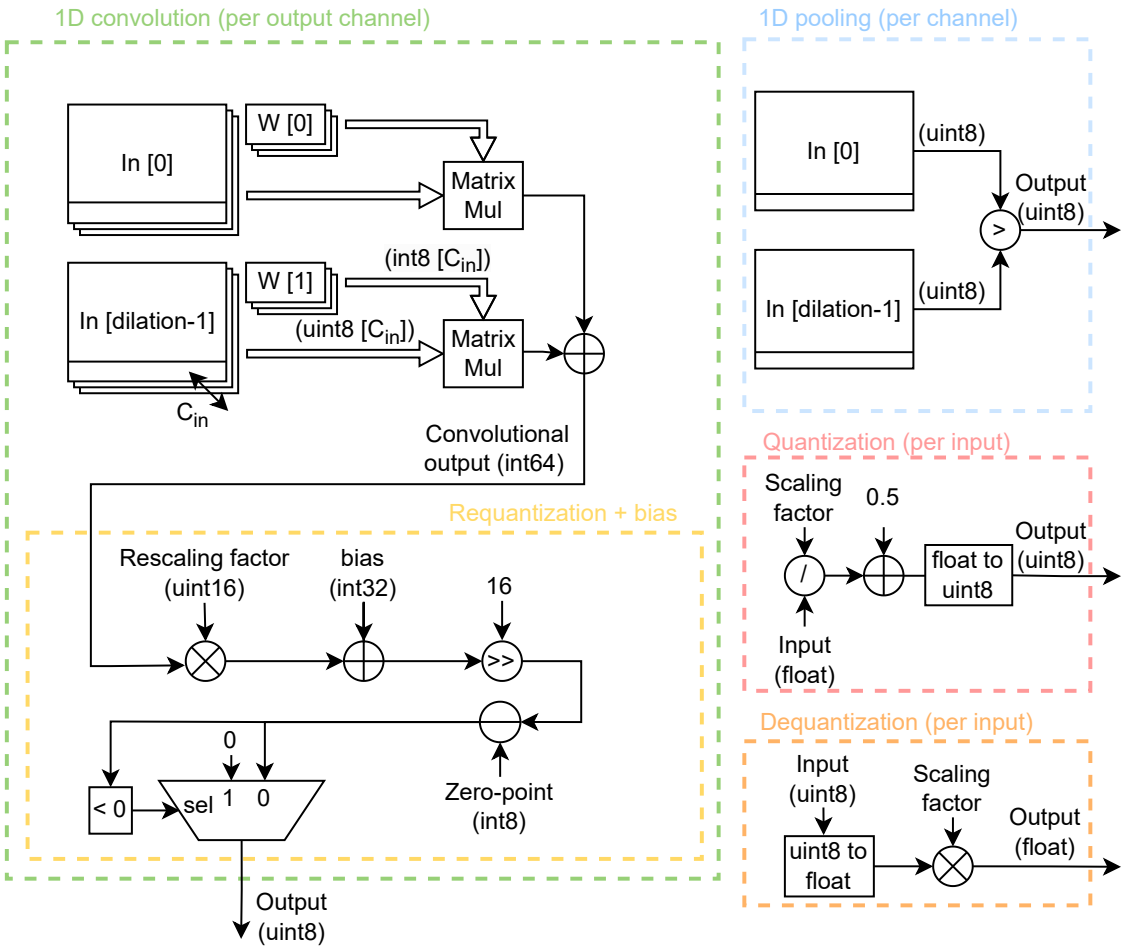


Figure 5.2: Dataflow diagram of 1D convolution, 1D pooling, quantization and dequantization as implemented in C++ and Vitis HLS. The 1D convolution is split into two segments, the convolutional operation itself and the requantization operation, where the bias is added. Requantization is needed to adjust the scaling factor and zero-point of the result for the next CNN layer. For layers with a zero-point of 0, the zero-point subtractor is omitted. For layers without a ReLU at the output, the multiplexer is omitted.

Table 5.1: Aggregated resources utilization of FAST-NN accelerator using all four SLRs.

Resource	**Required**	**Available**	**Utilization**
LUT	470563	1728000	27%
FF	114324	3456000	3.3%
DSP	12281	12288	99.9%
BRAM	0	5376	0%
URAM	0	1280	0%
CLB	89120	216000	41.3%

decrease the required number of DSPs. The large convolutional layers also limit the maximum clock speed of the design. The large number of multiply-accumulates that need to be processed in one cycle imposes a maximum fanout of 31, which takes 11.06 ns to complete. The design can run at a clock frequency of 90 MHz, producing one output every 11.11 ns. Each output corresponds to the classification on one window of 128 SNP positions.

5.5 Evaluation

5.5.1 Accuracy of quantized neural network

For each of the 6 datasets, further expanded on in table 2.2, a quantized version of FAST-NN and a floating-point version of the network is trained for 10 epochs. The quantized models are trained using QAT. The samples in the training and testing datasets have a width of 128 SNPs, and the DAFs are computed using a population of 128 samples. The training set consists of 1000 neutral and 1000 selective training samples, where 15% of the samples are used for validation. The network is saved after the epoch with the greatest validation accuracy. After training the models, each model is tested on 1000 neutral and 1000 selective test samples. Table 5.2 compares the accuracy of the quantized model to the accuracy of the floating-point model. For all datasets except D4, the difference in accuracy is negligible. For dataset D4, the quantized model has a slightly lower accuracy compared to the floating-point model.

Table 5.2: Accuracy of 8-bit quantized model and single-precision floating-point model for each dataset at a window width of 128.

	Accuracy	
Dataset	Quantized	Normal
D1	1.0	1.0
D2	0.945	0.944
D3	0.999	0.999
D4	0.920	0.944
D5	1.0	1.0
D6	1.0	1.0

5.5.2 Estimated inference speed on FPGA

The implementation can run at a clock speed of 90 MHz, processing one SNP location per cycle, and producing one output per cycle. To sustain this, the FAST-NN model requires an input of two 32-bit floating-point values per clock cycle. This equals an input data rate of 0.72 GB/s, which is not a bottleneck considering the DDR4 memory of the Alveo U250 has a theoretical maximum bandwidth of 77 GB/s [8].

Given the inference speed per SNP location of the accelerated FAST-NN implementation, the required time to perform detection on a genomic dataset can be estimated. Scanning human chromosome 1 requires processing 946,228 SNPs, and scanning all 22 human autosomes requires processing 12,157,787 SNPs. This is estimated to take 10.5 and 135 milliseconds to process, respectively, using the accelerated FAST-NN model. For reference, the FASTER-NN model is benchmarked in chapter 3. This model has a lower execution time than the FAST-NN model on a CPU, and takes 5.74 and 66 seconds to scan chromosome 1 and all 22 autosomes, respectively, on a CPU.

The classification speed of the FAST-NN model, as measured in chapter 2, is used as a reference to compare to the classification inference speed of the accelerated FAST-NN model. To perform classification using the accelerated FAST-NN model, the classification windows can be continuously

streamed to the accelerator by cascading the data of each input. To determine the classification output at the center of each window, the stream of outputs from the accelerator needs to be sub-sampled at a rate of 1 sample for every W outputs, where W is the width of the input window.

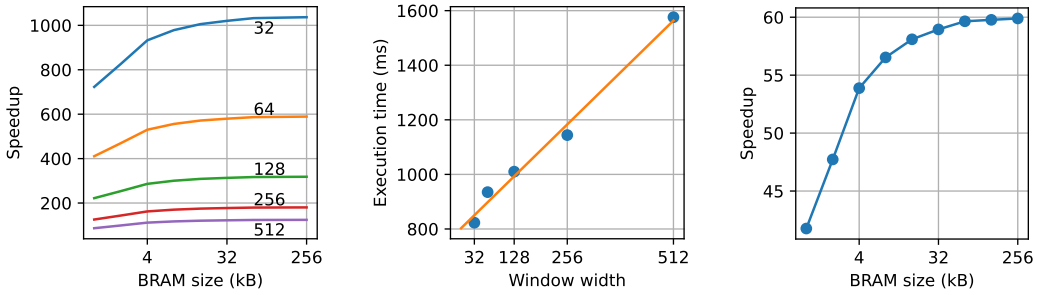
Table 5.3 shows the inference time of a CPU, GPU, and FPGA device running the FAST-NN model to classify 1000 neutral and 1000 selective DAF vectors. A single Intel Xeon CPU core at 2.1GHz running Ubuntu 20.04 and an Nvidia A40 GPU were used for benchmarking. The CPU and GPU implementations were realized using Pytorch version 2.3 [45], which supports a CPU backend and a CUDA (GPU) backend. The CUDA backend uses CUDA version 11.8 and is designed using the cuDNN (version 8.7) library, which is a CUDA library for accelerating deep neural networks. The FPGA inference speed estimation is computed by integrating the time to process every input window and adding the latency of 66 cycles. While the CPU inference time scales linearly with the window width, the GPU execution time is more-or-less constant with respect to window width, and has an outlier at a window width of 128, which runs significantly slower. The lack of linearity in the GPU timing is likely due to the jitter caused by the GPU being used as a shared resource, hindering the accurate timing of processes with a short execution time.

Table 5.3: Classification inference time of 2000 simulations using for input widths. *Execution time of FPGA is estimated based on the simulated throughput.

Window width	**Execution time (ms)**		
	CPU	**GPU**	**FPGA***
32	823	863	0.7
64	935	862	1.4
128	1010	1320	2.8
256	1144	894	5.7
512	1576	901	11.4

The CPU and GPU implementations load the data in batches, requiring multiple memory transfers, whilst the FPGA model assumes that all data is already loaded on the DDR4 memory of the Alveo board, and can be accessed without any latency. Table 5.3 describes a best-case scenario for the FPGA where the data can be loaded as a continuous data stream and data loading does not pose a bottleneck. To evaluate the impact of a more realistic scenario for the FPGA, an alternative data loading pipeline, where the data in the DDR4 memory is first buffered to on-chip BRAM and subsequently processed by the accelerator, is explored. The bandwidth for loading data from the DDR4 memory depends on the amount of data that can be loaded per transaction, which is limited by the BRAM buffer size. The DDR4 bandwidth for various buffer sizes is extrapolated from the work done by Lu et al. [40]. In this work, the bandwidth of a single DDR4 bank on the Alveo U200, at a frequency of 100 MHz, is presented for various data sizes. This is close to the bandwidth of a single bank on the Alveo U250 at a frequency of 90 MHz. The data loading bandwidth of the DDR4 memory increases when loading the data in larger chunks, which requires a larger BRAM buffer. Figure 5.3a shows the speedup of the FPGA implementation compared to the CPU implementation for different BRAM buffer sizes and for datasets of various window widths, where a larger window width increases the amount of memory that needs to be transferred per window. The smallest BRAM buffer that is evaluated is 1kB, and the largest buffer is 256 kB, which requires implementing 2 and 128 BRAM banks, respectively, out of the 500 available BRAM banks per SLR.

The total execution time of the CPU is dominated by the data loading time. The execution time for a dataset with a window width of 64 is 0.94 seconds, and the total execution time for a window width of 512 is 1.58 seconds. As a result, figure 5.3a shows that the speedup for narrow windows is very high, but it plateaus for wider windows. This is because for smaller windows the computation time becomes negligible compared to the data loading time. The CPU needs to load data from external memory, imposing a constant additional data loading time, which is not present when evaluating the FPGA execution time. To evaluate the FPGA performance compared to a best-case scenario for the CPU execution time, the window width-invariant component of the CPU execution time is subtracted from the total execution time. To eliminate this component, a linear regression model is fitted to the CPU timing data, illustrated in figure 5.3b. The intercept of the linear regression model is subtracted from the CPU execution time, and the speedup of the FPGA execution time is compared to the linear component of the CPU execution time, shown in figure 5.3c. The FPGA design can achieve a speedup of up to 60x when compared to this reduced CPU execution time. Because the measured execution times on the GPU do not scale linearly with the window width, the linear regression-based analysis is not performed for the GPU. Performing a



(a) Speedup of FPGA with respect to total CPU execution time, for various input window widths (annotated).

(b) Linear regression of CPU execution time for various input window widths.

(c) Speedup of FPGA with respect to linear component of CPU execution time.

Figure 5.3: FPGA speedup with respect to CPU execution time, before and after adjusting the CPU execution time by removing the CPU execution time offset through linear regression.

more thorough analysis where window widths exceeding 512 are tested could give more insight into the scaling of the GPU implementation with respect to window width.

The bottleneck of the hardware design with regard to resources and timing is the 80-channel to 80-channel 1D convolutional layers. These operations account for most of the required DSPs of the design, and the multiply-accumulates in these convolutional layers pose a timing bottleneck. By limiting the number of input and output channels for the convolutional layers, the number of multiply-accumulates per convolutional layer is reduced. This may negatively affect the accuracy of the model but reduces resource requirements and shortens the critical path, allowing a higher clock frequency. In addition to this, using an initiation interval greater than 1 can reduce the required number of resources. By reducing the resource requirements, smaller, more affordable FPGA devices can fit the FAST-NN design. This presents a trade-off between model accuracy, throughput, and resource requirements.

5.6 Conclusion and discussion

The state-of-the-art uses CNNs for selective sweep classification, but these models are computationally more expensive than summary statistics-based methods. Chromosomes often contain millions of SNPs, and fine-grained detection using a CNN on realistic genetic datasets can be time-consuming. In this chapter, the compact allele frequency data format, and the small network size of FAST-NN, are leveraged to synthesize a fully-pipelined streaming architecture for a FAST-NN accelerator. A hardware accelerator is designed, which, due to the streaming implementation of the CNN-based detection method, lends itself well to a streaming architecture in hardware. Data reuse is exploited allowing for fine-grained selective sweep detection with minimal additional processing. Due to the compact data format of allele frequencies, a fully-pipelined CNN architecture of FAST-NN fits on a single FPGA device, even when using a high-precision quantization scheme of 8 bits. By using QAT, the classification performance is not severely affected by quantization error, while inference speed is significantly reduced.

In future work, a co-design of the network architecture and the hardware architecture, where the hyperparameters of the network are constrained by the resources and timing requirements of the hardware design, can alleviate current timing bottlenecks and high DSP utilization. Moreover, by investigating the effect of lower precision quantization on the accuracy of FAST-NN, resource requirements may be reduced without significantly reducing classification performance.

Discussion and conclusion

Single nucleotide polymorphisms (SNPs) are processed to classify the presence of a selective sweep, which is an indication of positive natural selection. Summary statistics-based methods have recently been outperformed by deep learning-based approaches, in particular by convolutional neural networks (CNNs). Many existing CNN models for selective sweep classification process raw or reordered, but uncompressed, SNP data. Despite the high classification accuracy of CNN-based methods, they are not widely adopted, in part due to the increase in execution time compared to summary statistics-based classification. The computational complexity of CNNs scales linearly with the height and width of the input data, and modern datasets in population genomics can be composed of thousands of samples (height) and millions of SNPs (width). There is a demand for efficiently processing SNP data using CNNs without compromising classification performance. Summary statistics for selective sweeps classically identify three signatures in SNP data: a shift in the site-frequency-spectrum (SFS) towards low and high allele frequencies, a local decrease in the density of polymorphisms, and a pattern in linkage disequilibrium (LD) showing increased linkage disequilibrium at either side of the beneficial mutation site and lower linkage disequilibrium across the site of the beneficial mutation. Two of the signatures, namely the shift in (SFS) and decrease in density of polymorphisms can be represented in a vector format that is size-invariant to the sample size. Using allele frequencies and pairwise SNP distances as a compact data representation, a 1D CNN architecture search is performed. By selecting the model with the highest classification accuracy, the FAST-NN architecture is chosen. This model outperforms the classification accuracies of published 2D CNN models that use either raw SNP data or summary statistics. This demonstrates that CNN-based classification for positive selection can be be made scalable, without compromising performance, by using a compact data format of derived allele frequencies and pairwise SNP distances. FAST-NN is almost as accurate as a 2D CNN model that processes raw SNP data and pairwise SNP distances but has a significantly lower execution time than any 2D CNN model. This makes FAST-NN highly suitable as a practical tool for selective sweep classification, especially on large datasets.

FAST-NN is designed for optimizing selective sweep classification accuracy. Most practical applications of selective sweep classification focus on localizing selective sweeps on genomic data. Selective sweep detection methods usually use a grid-based approach for localizing selective sweeps. In a grid-based approach, a window on the genome is evaluated at each gridpoint, separated by a fixed basepair distance. A grid-based approach can be an effective way to perform a coarse-grained scan, however, for fine-grained detection a grid-based detection approach introduces redundant computations due to overlapping windows between grid points. Leveraging the translational equivariance of CNNs, intermediate computations on overlapping data between windows can be reused, eliminating redundant computations. However, data reuse is not possible when window-wide preprocessing or data rearrangement is applied, since overlapping data is not identical between windows when this is done. By using dilated convolutions any CNN-based classification model can be transformed to efficiently perform fine-grained detection, fully reusing all intermediate computations between windows. When data reuse is applied for fine-grained detection, the computational complexity of detection becomes nearly constant with respect to the width of each classified window. Considering this, FASTER-NN is designed, which has a wider receptive field than FAST-NN. The dense output of FASTER-NN is post-processed by averaging classifications on a basepair-equidistant grid around each position. Post-processing filters out outliers and ensures a fixed evaluated width, in terms of basepairs, per classification. The efficient detection framework in which FASTER-NN is deployed allows fine-grained genome-wide scans for selective sweeps at a reduced execution time compared to a grid-based approach. Moreover, FASTER-NN has a higher detection true positive rate (TPR) and precision than FAST-NN. By investigating the impact of window width on the detection and classification performance of FAST-NN and FASTER-NN, it is found that using wide

windows of up to 512 SNPs results in better classification and detection performance, especially for FASTER-NN. By using wide input windows with a model that has a large receptive field, post-processing using a basepair-equidistant grid, and by leveraging data reuse for dilated convolutions, the efficiency and effectiveness for the detection of selective sweeps is optimized. It is found that the input window width has a profound effect on the detection precision of selective sweeps in the presence of a recombination hotspot, where a specific window width enables very precise detection and another window width causes the precision to drop drastically.

When training the FAST-NN and FASTER-NN models, for each confounding factor one model is trained to mitigate the adverse effect that the confounding factor has on classification and detection performance. In realistic datasets, for example when scanning real genomes, one can expect various confounding factors to be present throughout the data. To effectively scan such a dataset, it is desirable to have one model that is trained to robustly detect selection in the presence of a variety of confounding factors. In future work, one can train a model that is robust to many confounding factors, and perform an architecture search to identify if this requires a more complex architecture. Regardless of whether a model is trained to be robust against one, or many, confounding factors, training a model requires simulated data that behaves similarly to the real data under neutral and selective conditions. The effectiveness of the model is highly dependent on the quality of the simulated data. If the model is misspecified, because the simulation tool is imperfect, or because the simulation parameters are incorrect, this can affect the ability of the model to perform as expected on real data. Training a model that can classify selective sweeps, independent of the simulation parameters, could improve robustness to the misspecification of training data, and make a model more versatile. In future work, the sensitivity of FAST-NN and FASTER-NN to simulation parameters can be explored, and the training data can be extended to decrease sensitivity to simulation parameters.

Population effects and genomic effects can confound the signature effects of selective sweeps. The presence of a confounding factor can thus influence the likelihood of a false positive. Moreover, the presence of a confounding factor can affect the optimal window width for selective sweep detection, by affecting the polymorphic density, as is demonstrated when analyzing the precision of selective sweep detection in the presence of recombination hotspots. It is therefore valuable to consider the presence of confounding factors when attempting to detect selective sweeps. Due to the versatility of CNNs, if the model has sufficient complexity it can classify multiple genomic effects simultaniously, performing inference once. To investigate this, FASTER-NN is extended, outputting two classification scores, one scoring the presence of a selective sweep and one scoring the presence of a recombination hotspot. The allele frequency-based FASTER-NN model is not able to perform well on classifying recombination hotspots, due to the fact that recombination hotspots are mostly detectable through their signature in LD. Compressing the raw SNP data into allele frequencies removes linkage information between SNPs. To partially conserve LD, samples are subdivided into equally sized groups, and from each group, the DAF is computed. Each DAF group is separately processed, and through a Grouped Pooling Block, the intermediate results of each DAF group are combined, allowing the model to identify LD patterns. By dividing the data into a larger number of groups, more information regarding LD is conserved, but the processing time increases. Using grouped DAF, it is demonstrated that selective sweeps and recombination hotspots can be classified accurately and simultaneously. In addition, insight is given into the importance of LD for classifying recombination hotspots, and a heuristic hotspot classification approach is designed, that can reduce inference time compared to processing raw SNP data. From the results it is not clear whether classifying the confounding factor improves the ability to classify a selective sweep, since, in the case of the recombination hotspot, the simulations with a selective sweep are correctly classified by each model. Potentially, multi-label classification can be extended to classifying various confounding factors simultaneously, in addition to selective sweeps. Chapter 4 has focused on the combined classification of two genomic effects, the analysis of multi-label classification on detection performance is left for future work. Additionally, in chapter 3, it is shown that, in the presence of a recombination hotspot, the post-processing window width has a profound effect on the detection precision of FASTER-NN. In future work, the presence of confounding factors can be used to optimize the post-processing algorithm for selective sweep detection.

FPGAs can be implemented to accelerate specific algorithms by leveraging high parallelism in custom hardware architectures. Accelerating CNNs using FPGAs is challenging due to the large number of parameters that need to be stored on stored on-chip, and large number of multiplications and additions performed in its matrix multiplications. Fortunately, using the compact allele frequency data representation, the FAST-NN model only required 1D convolutions, reducing the number of parameters and the size of the matrix multiplications compared to 2D convolutions. The

dense detection algorithm can be implemented as a streaming architecture on an FPGA, fitting an 8-bit quantized FAST-NN model. The accelerated FAST-NN model can generate one inference per clock cycle. The accuracy of the quantized FAST-NN model is equal to the floating points model for all datasets except one. Using the accelerated FAST-NN model, whole-genome scans can be executed faster than when using conventional hardware, such as a CPU or GPU, with limited impact on the classification accuracy. The current hardware implementation of FAST-NN is specific to the exact architecture of FAST-NN, and changing the model architecture requires adjusting the hardware architecture. In future work, coarse-grained reconfigurable architectures, such as those available on the Versal adaptive SoC, can be explored, allowing for a more flexible hardware implementation.

Conclusively, to use CNNs for scalable analysis of SNP data, it is quintessential to understand the underlying information in the data that can serve as evidence for a selective sweep. For selective sweeps, the SNP data can be reduced to allele frequencies and pairwise SNP distances, preserving most information required to perform selective sweep classification. This compression only has a minor impact on the separability of neutral data and selective data. Through this compact data format, selective sweeps can be classified in a scalable manner without sacrificing robustness. By viewing selective sweep localization as a detection problem rather than a classification problem, and by leveraging data reuse between overlapping classification windows, a detection method is designed that allows for effective and efficient fine-grained detection. This method is accelerated further through hardware acceleration on reconfigurable hardware. FASTER-NN and FAST-NN can analyse genomic data to identify regions affected by positive natural selection with higher accuracy and lower execution time than existing published methods. Verifying the effectiveness of these models is challenging, since simulated data is used for training and testing, and no labeled data sampled from real populations is available. The application of the proposed models for real case studies will be required to demonstrate the effectiveness of these models on real data.

Bibliography

[1] N. Alachiotis and P. Pavlidis. RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors. *Commun Biol*, 1:79, 2018.

[2] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis. Exploring fpgas for accelerating the phylogenetic likelihood function. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, 2009. doi: 10.1109/IPDPS.2009.5160929.

[3] N. Alachiotis, A. Stamatakis, and P. Pavlidis. OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinformatics*, 28(17):2274–2275, 07 2012. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts419. URL https://doi.org/10.1093/bioinformatics/bts419.

[4] Nikolaos Alachiotis and Alexandros Stamatakis. Fpga acceleration of the phylogenetic parsimony kernel? In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 417–422, 2011. doi: 10.1109/FPL.2011.83.

[5] Nikolaos Alachiotis, Charalampos Vatsolakis, Grigorios Chrysos, and Dionisios Pnevmatikatos. Accelerated inference of positive selection on whole genomes. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 202–2027, 2018. doi: 10.1109/FPL.2018.00041.

[6] Nikolaos Alachiotis, Charalampos Vatsolakis, Grigorios Chrysos, and Dionisios Pnevmatikatos. Raisd-x: A fast and accurate fpga system for the detection of positive selection in thousands of genomes. *ACM Trans. Reconfigurable Technol. Syst.*, 13(1), dec 2019. ISSN 1936-7406. doi: 10.1145/3364225. URL https://doi.org/10.1145/3364225.

[7] M. T. Alam, D. K. de Souza, S. Vinayak, S. M. Griffing, A. C. Poe, N. O. Duah, A. Ghansah, K. Asamoa, L. Slutsker, M. D. Wilson, J. W. Barnwell, V. Udhayakumar, and K. A. Koram. Selective sweeps and genetic lineages of Plasmodium falciparum drug -resistant alleles in Ghana. *J Infect Dis*, 203(2):220–227, Jan 2011.

[8] *Alveo U200 and U250 Data Center Accelerator Cards Data Sheet (DS962)*. AMD, 06 2023.

[9] Andreacute; Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11), November 2019. ISSN 2476-0757. doi: 10.23915/distill.00021. URL http://dx.doi.org/10.23915/distill.00021.

[10] N. Arnheim, P. Calabrese, and M. Nordborg. Hot and cold spots of recombination in the human genome: the reason we should find them and how this can be achieved. *Am J Hum Genet*, 73(1):5–16, Jul 2003.

[11] Dimitrios Bozikas, Nikolaos Alachiotis, Pavlos Pavlidis, Evripides Sotiriades, and Apostolos Dollas. Deploying fpgas to future-proof genome-wide analyses based on linkage disequilibrium. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2017. doi: 10.23919/FPL.2017.8056814.

[12] J. M. Braverman, R. R. Hudson, N. L. Kaplan, C. H. Langley, and W. Stephan. The hitchhiking effect on the site frequency spectrum of DNA polymorphisms. *Genetics*, 140(2):783–796, Jun 1995.

[13] J. Chan, V. Perrone, J. P. Spence, P. A. Jenkins, S. Mathieson, and Y. S. Song. A Likelihood-Free Inference Framework for Population Genetic Data using Exchangeable Neural Networks. *Adv Neural Inf Process Syst*, 31:8594–8605, Dec 2018.

[14] Federico Corradi, Zhanbo Shen, Hanqing Zhao, and Nikolaos Alachiotis. Accelerated spiking convolutional neural networks for scalable population genomics. In *Proceedings of the 14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, HEART '24, page 53–62, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400717277. doi: 10.1145/3665283.3665285. URL https://doi.org/10.1145/3665283.3665285.

[15] Reinout Corts, Niek Sterenborg, and Nikolaos Alachiotis. Accelerated ld-based selective sweep detection using gpus and fpgas. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 196–205, 2022. doi: 10.1109/IPDPSW55747.2022.00044.

[16] Natasja G. de Groot and Ronald E. Bontrop. The hiv-1 pandemic: does the selective sweep in chimpanzees mirror humankind's future? *Retrovirology*, 10(1):53, May 2013. ISSN 1742-4690. doi: 10.1186/1742-4690-10-53. URL https://doi.org/10.1186/1742-4690-10-53.

[17] Michael DeGiorgio, Christian D. Huber, Melissa J. Hubisz, Ines Hellmann, and Rasmus Nielsen. SweepFinder2: increased sensitivity, robustness and flexibility. *Bioinformatics*, 32(12):1895–1897, 02 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw051. URL https://doi.org/10.1093/bioinformatics/btw051.

[18] A. Auton et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, Oct 2015. ISSN 1476-4687. doi: 10.1038/nature15393. URL https://doi.org/10.1038/nature15393.

[19] Dionysios Filippas, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. Streaming dilated convolution engine. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(3):401–405, 2023. doi: 10.1109/TVLSI.2022.3233882.

[20] Lex Flagel, Yaniv Brandvain, and Daniel R Schrider. The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference. *Molecular Biology and Evolution*, 36(2):220–238, 12 2018. ISSN 0737-4038. doi: 10.1093/molbev/msy224. URL https://doi.org/10.1093/molbev/msy224.

[21] Yun-Xun Fu and Wen-Hsiung Li. Coalescing into the 21st century: An overview and prospects of coalescent theory. *Theoretical Population Biology*, 56(1):1–10, 1999.

[22] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[23] T. Ryan Gregory. Understanding natural selection: Essential concepts and common misconceptions. *Evolution: Education and Outreach*, 2(2):156–175, Jun 2009. ISSN 1936-6434. doi: 10.1007/s12052-009-0128-1. URL https://doi.org/10.1007/s12052-009-0128-1.

[24] Benjamin C Haller and Philipp W Messer. SLiM 3: forward genetic simulations beyond the Wright–Fisher model. *Molecular biology and evolution*, 36(3):632–637, 2019.

[25] Garrett Hellenthal and Matthew Stephens. msHOT: modifying Hudson's ms simulator to incorporate crossover and gene conversion hotspots. *Bioinformatics*, 23(4):520–521, 12 2006. ISSN 1367-4803. doi: 10.1093/bioinformatics/btl622. URL https://doi.org/10.1093/bioinformatics/btl622.

[26] Richard R. Hudson. Generating samples under a Wright–Fisher neutral model of genetic variation . *Bioinformatics*, 18(2):337–338, 02 2002. ISSN 1367-4803. doi: 10.1093/bioinformatics/18.2.337. URL https://doi.org/10.1093/bioinformatics/18.2.337.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

[28] Mengfei Ji, Zaid Al-Ars, Peter Hofstee, Yuchun Chang, and Baolin Zhang. Fpqnet: Fully pipelined and quantized cnn for ultra-low latency image classification on fpgas using opencapi. *Electronics*, 12(19), 2023. ISSN 2079-9292. doi: 10.3390/electronics12194085. URL https://www.mdpi.com/2079-9292/12/19/4085.

[29] Lin Kang, Guijuan He, Amanda K. Sharp, Xiaofeng Wang, Anne M. Brown, Pawel Michalak, and James Weger-Lucarelli. A selective sweep in the spike gene has driven sars-cov-2 human adaptation. *Cell*, 184(17):4392–4400.e4, 2021. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2021.07.007. URL https://www.sciencedirect.com/science/article/pii/S0092867421008333.

[30] Server Kasap and Khaled Benkrid. A high performance fpga-based core for phylogenetic analysis with maximum parsimony method. In *2009 International Conference on Field-Programmable Technology*, pages 271–277, 2009. doi: 10.1109/FPT.2009.5377652.

[31] Server Kasap and Khaled Benkrid. High performance phylogenetic analysis with maximum parsimony on reconfigurable hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(5):796–808, 2011. doi: 10.1109/TVLSI.2009.2039588.

[32] Andrew D Kern and Daniel R Schrider. diploS/HIC: an updated approach to classifying selective sweeps. *G3: Genes, Genomes, Genetics*, 8(6):1959–1970, 2018.

[33] Y. Kim and R. Nielsen. Linkage disequilibrium as a signature of selective sweeps. *Genetics*, 167(3):1513–1524, Jul 2004.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[35] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2020.107398. URL https://www.sciencedirect.com/science/article/pii/S0888327020307846.

[36] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018.

[37] M. Elise Lauterbur, Mariz Izabel A. Cavassim, Ariella L. Gladstein, Graham Gower, et al. Expanding the stdpopsim species catalog, and lessons learned for realistic genome simulations. *bioRxiv*, 2022. doi: 10.1101/2022.10.29.514266. URL https://www.biorxiv.org/content/early/2022/10/31/2022.10.29.514266.

[38] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[39] Xinyu Liu, Gaole Sai, and Shengyu Duan. Hardware acceleration for 1d-cnn based real-time edge computing. In Shaoshan Liu and Xiaohui Wei, editors, *Network and Parallel Computing*, pages 192–204, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-21395-3.

[40] Alec Lu, Zhenman Fang, and Lesley Shannon. Demystifying the soft and hardened memory systems of modern fpgas for software programmers through microbenchmarking. *ACM Trans. Reconfigurable Technol. Syst.*, 15(4), jun 2022. ISSN 1936-7406. doi: 10.1145/3517131. URL https://doi.org/10.1145/3517131.

[41] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks, 2017.

[42] Pavlos Malakonakis, Andreas Brokalakis, Nikolaos Alachiotis, Evripides Sotiriades, and Apostolos Dollas. Exploring modern fpga platforms for faster phylogeny reconstruction with raxml. In *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 97–104, 2020. doi: 10.1109/BIBE50027.2020.00024.

[43] Arnaud Nguembang Fadja, Fabrizio Riguzzi, Giorgio Bertorelle, and Emiliano Trucchi. Identification of natural selection in genomic data with deep convolutional neural network. *BioData Mining*, 14(1):51, Dec 2021. ISSN 1756-0381. doi: 10.1186/s13040-021-00280-9. URL https://doi.org/10.1186/s13040-021-00280-9.

[44] J. nguez, L. Wienbrandt, J. C. ssens, D. Ellinghaus, M. Schimmler, and B. Schmidt. Parallelizing Epistasis Detection in GWAS on FPGA and GPU-Accelerated Computing Systems. *IEEE/ACM Trans Comput Biol Bioinform*, 12(5):982–994, 2015.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, et al. Pytorch: An imperative style, high-performance deep learning library, 2019.

[46] P. Pavlidis, D. ivkovic, A. Stamatakis, and N. Alachiotis. SweeD: likelihood-based detection of selective sweeps in thousands of genomes. *Mol Biol Evol*, 30(9):2224–2234, Sep 2013.

[47] Thomas D. Petes. Meiotic recombination hot spots and cold spots. *Nature Reviews Genetics*, 2(5):360–369, May 2001. ISSN 1471-0064. doi: 10.1038/35072078. URL https://doi.org/10.1038/35072078.

[48] B. Pfeifer, U. Wittelsburger, S. E. Ramos-Onsins, and M. J. Lercher. PopGenome: an efficient Swiss army knife for population genomic analyses in R. *Mol Biol Evol*, 31(7):1929–1936, 2014.

[49] Sebastian E. Ramos-Onsins and Julio Rozas. Statistical Properties of New Neutrality Tests Against Population Growth. *Molecular Biology and Evolution*, 19(12):2092–2100, 2002.

[50] S. Schaffner and P. Sabeti. Evolutionary Adaptation and Positive Selection in Humans. https://www.nature.com/scitable/topicpage/evolutionary-adaptation-in-the-human-lineage-12397/, 2008.

[51] Stephan C. Schuster. Next-generation sequencing transforms today's biology. *Nature Methods*, 5(1):16–18, Jan 2008. ISSN 1548-7105. doi: 10.1038/nmeth1156. URL https://doi.org/10.1038/nmeth1156.

[52] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks, 2014.

[53] J. M. Smith and J. Haigh. The hitch-hiking effect of a favourable gene. *Genet Res*, 23(1): 23–35, Feb 1974.

[54] F Tajima. Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics*, 123(3):585–595, 1989.

[55] Kosuke M. Teshima and Hideki Innan. mbs: modifying hudson's ms software to generate samples of dna sequences with a biallelic site under selection. *BMC Bioinformatics*, 10(1): 166, May 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-166. URL https://doi.org/10.1186/1471-2105-10-166.

[56] Luis Torada, Lucrezia Lorenzon, Alice Beddis, Ulas Isildak, Linda Pattini, Sara Mathieson, and Matteo Fumagalli. Imagene: a convolutional neural network to quantify natural selection from genomic data. *BMC Bioinformatics*, 20(9):337, Nov 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-2927-x. URL https://doi.org/10.1186/s12859-019-2927-x.

[57] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, page 65–74, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450343541. doi: 10.1145/3020078.3021744. URL https://doi.org/10.1145/3020078.3021744.

[58] Sjoerd van den Belt, Hanqing Zhao, and Nikolaos Alachiotis. Scalable CNN-based classification of selective sweeps using derived allele frequencies. *Bioinformatics*, 2024. doi: 10.1093/bioinformatics/btae385. In press.

[59] P. M. Visscher, M. A. Brown, M. I. McCarthy, and J. Yang. Five years of GWAS discovery. *Am J Hum Genet*, 90(1):7–24, Jan 2012.

[60] J. D. Wall and L. S. Stevison. Detecting Recombination Hotspots from Patterns of Linkage Disequilibrium. *G3 (Bethesda)*, 6(8):2265–2271, Aug 2016.

[61] Lai Wei, Dongsheng Liu, Jiahao Lu, Lingsong Zhu, and Xuan Cheng. A low-cost hardware architecture of convolutional neural network for ecg classification. In *2021 9th International Symposium on Next Generation Electronics (ISNE)*, pages 1–4, 2021. doi: 10.1109/ISNE48910.2021.9493657.

[62] Hannah Weigand and Florian Leese. Detecting signatures of positive selection in non-model species using genomic data. *Zoological Journal of the Linnean Society*, 184(2):528–583, 04 2018. ISSN 0024-4082. doi: 10.1093/zoolinnean/zly007. URL https://doi.org/10.1093/zoolinnean/zly007.

[63] Paul N. Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas Kolala Venkataramanaiah, Jae sun Seo, and Matthew Mattina. Fixynn: Efficient hardware for mobile computer vision via transfer learning, 2019.

[64] Lars Wienbrandt, Jan Christian Kässens, Jorge González-Domínguez, Bertil Schmidt, David Ellinghaus, and Manfred Schimmler. Fpga-based acceleration of detecting statistical epistasis in gwas. *Procedia Computer Science*, 29:220–230, 2014. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2014.05.020. URL https://www.sciencedirect.com/science/article/pii/S1877050914001975. 2014 International Conference on Computational Science.

[65] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation, 2020.

[66] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016.

[67] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, page 161–170, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333153. doi: 10.1145/2684746.2689060. URL https://doi-org.ezproxy2.utwente.nl/10.1145/2684746.2689060.

[68] Hanqing Zhao, Pavlos Pavlidis, and Nikolaos Alachiotis. Sweepnet: A lightweight cnn architecture for the classification of adaptive genomic regions. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701900. doi: 10.1145/3592979.3593411. URL https://doi.org/10.1145/3592979.3593411.

[69] Hanqing Zhao, Matthijs Souilljee, Pavlos Pavlidis, and Nikolaos Alachiotis. Genome-wide scans for selective sweeps using convolutional neural networks. *Bioinformatics*, 39 (Supplement_1):i194–i203, 06 2023. ISSN 1367-4811. doi: 10.1093/bioinformatics/btad265. URL https://doi.org/10.1093/bioinformatics/btad265.

[70] Stephanie Zierke and Jason D. Bakos. Fpga acceleration of the phylogenetic likelihood function for bayesian mcmc inference methods. *BMC Bioinformatics*, 11(1):184, Apr 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-184. URL https://doi.org/10.1186/1471-2105-11-184.

[71] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2023.