UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

# Locating Selective Sweeps with Accelerated Convolutional Neural Networks

**Matthijs L. Souilljee**

**M.Sc. Thesis**
**September 19, 2021**

**Supervisors:**
dr. ir. S.H. Gerez
dr. ir. N. Alachiotis
dr. C.G. Zeinstra

Computer Architectures for
Embedded Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7522 NH Enschede
The Netherlands

# ABSTRACT

Discovering how a species adapted to a specific environment over a long period, and how this affected the evolution of that species is of great importance to researchers. A major force that drives the shaping of the evolution of a species is positive selection. Positive selection provides information on how a species evolved, and therefore how the species adapted to its environment. The act of positive selection leaves a selective sweep in the genetic material of a species. The detection and localization of selective sweeps and therefore traces of positive selection is a goal for the development of various methods and tools. For sweep detection, various signature-based methods and tools are developed. Besides these signature-based methods and tools, the use of convolutional neural networks (CNN) for whole-genome sweep detection is not yet explored. This work presents ASDEC (Accurate Sweep Detection Enabled by a CNN), a CNN-based method for whole-genome sweep detection. ASDEC was developed in a user-configurable way and shows great performance against current signature-based methods and tools. ASDEC is, to the best of my knowledge, the first whole-genome CNN-based sweep detection method. For the development of ASDEC, a hand-designed neural architecture search (NAS) was used and led to a final CNN architecture (dubbed SweepNet). ASDEC was compared with signature-based methods and tools such as RAiSD, OmegaPlus, SweeD, and SweepFinder2. ASDEC showed equal to increasing performance for almost all data-sets compared with the top performer signature-based method. The performance evaluation of ASDEC consisted of three different confounding factors bottleneck, migration, and recombination. Besides the use of simulated data-sets, ASDEC can be deployed for real genomic data-sets. A scan of the first chromosome of the human genome (Yoruba population, 1000Genomes data-set) was performed, showing nine different candidate genes. The nine candidate genes discovered by ASDEC have already been identified by previous research to be targets of positive selection. ASDEC provides support for conventional hardware such as multi-core CPUs and GPUs. Extending the usability of ASDEC even further a CNN inference accelerator is implemented and compared with a multi-core CPU in terms of performance. Execution on a state of the art FPGA achieves a 10.7x faster processing than a general-purpose six-core CPU.

# ACKNOWLEDGMENTS

Firstly I would like to thank my supervisor N. Alachiotis for all the help. He provided me with enough information to help me get through the various subjects and helped me understand the core concepts of the research field of population genetics. He also provided me with feedback that enabled me to constantly improve my work. I would also like to thank N. Alachiotis for the weekly meetings and the guidance in the design of the ASDEC framework. I would also like to thank my other supervisors S.H. Gerez and C.G. Zeinstra for reviewing my M.Sc. research and providing me with the necessary feedback. Lastly, I would like to thank friends and family for all the support.

# CONTENTS

# List of Figures

# List of Tables

# NOMENCLATURE

| | |
|---|---|
| HGP | Human Genome Project |
| CNN | Convolutional Neural Network |
| NN | Neural Networks |
| CPU | Central Processing Unit |
| GPU | Graphical Processing Unit |
| FPGA | Field Programmable Gate Array |
| ASIC | Application-Specific Integrated Circuits |
| TPU | Tensor Processing Unit |
| TSS | Transcription Start Sites |
| FPS | Frames Per Second |
| ASDEC | Accurate Sweep Detection Enabled by a CNN |
| DPU | Deep-learning Processing Unit |
| NAS | Neural Architecture Search |
| AI | Artificial Intelligence |
| ANN | Artificial Neural Networks |
| A | Adenine |
| T | Thymine |
| G | Guanine |
| C | Cytosine |
| SNP | Single-Nucleotide Polymorphism |
| DNA | DeoxyriboNucleic Acid |
| RNA | RiboNucleic acid |
| MSA | Multiple Sequence Alignment |
| ISM | Infinite Site Model |
| SFS | Site Frequency Spectrum |
| LD | Linkage Disequilibrium |
| ANN | Artificial Neural Networks |
| ReLU | Rectified Linear Unit |
| YOLO | You Only Look Once |
| RGB | Red Green Blue |
| IP | Intellectual Property |
| HLS | High-Level Synthesis |
| DNNDK | Deep Neural Network Development Kit |
| CLR | Composite Likelihood Ratio |
| SweeD | Sweep Detector |
| RAiSD | Raised Accuracy in Sweep Detection |
| VCF | Variant Call Format |
| FASTA | FAST-All |
| kb | KiloBase |
| bp | BasePairs |
| IO | Input Output |
| TPR | True Positive Rate |
| FPR | False Positive Rate |
| MAC | Multiply Accumulate |
| FLOP | Floating Point Operation |

| | |
|---|---|
| DSP | Digital Signal Processor |
| LUT | Look Up Table |
| RAM | Random-Access Memory |
| PCIe | Peripheral Component Interconnect express |
| USB | Universal Serial Bus |
| PL | Programmable Logic |
| LV | Low Voltage |
| HBM | High Bandwidth Memory |
| PE | Processing Engine |
| GOPs | Giga Operations Per second |
| MPSoC | Multiprocessor System-On-Chip |
| PP | Pixel Parallelism |
| ICP | Input Channel Parallelism |
| OCP | Output Channel Parallelism |
| FP32 | 32-bit Floating-Point |
| INT8 | 8-bit Integer |
| CAT | Climate Ambient Temperature |
| UVR | Ultraviolet Radiation |
| SD | Sunlight Duration |
| CMS | Composite of Multiple Signals |
| GRoSS | Graph-aware Retrieval of Selective Sweeps |
| IHS | Intergrated Haplotype Score |
| FCRLs | Fc receptor-like molecules |

# 1 INTRODUCTION

Populations genetics studies the variations within the evolution of all genetic material (genome) in populations[6]. Finding variations within genomic data is coupled to finding a locus or a sub-genomic region, where a locus refers to a position within the genome. A milestone within population genetics with great importance is the sequencing of the complete human genome which is part of the Human Genome Project (HGP)[7], and paved the way to population genetics becoming a research field where large and complex data-sets are ample. The information provided by the HGP can for example be deployed to develop new ways to treat, cure or prevent human diseases.

Besides the ability to sequence a complete human genome, modern population genetics also includes the generation of genome-wide sequence data-sets. Generation is achieved with the help of the coalescent process[8],[9], and thereby enabling software-based on coalescent simulation such as ms[10] to create vast amounts of synthetic data with relatively low afford. Processing genomic information with high throughput techniques enables researchers to use the vast amounts of data available to them. These high throughput techniques are (mostly) based on machine learning methodologies, and more recent deep learning techniques[11].

Besides all the advancements made with regards to the population genetics research field, another research field has also seen vast improvements. This is the research field of image recognition and computer vision. Techniques from computer vision are based on machine learning and in more recent years deep learning methodologies. A driving force behind these advancements in computer vision has been the improvement of deep learning methods that led to the evolution of Convolutional Neural Networks (CNN). CNNs are most prominent in state-of-the-art image classification & object detection tooling [12]. A well-known application for CNNs and Neural Networks (NN) in general is for example the classification of the MNIST data set, which was first introduced in 1998 by LeCun et al. [13].

Convolutional Neural Networks are not restricted to applications regarding image processing, but could also be used for speech recognition, natural language processing and speech synthesis. Genomic data can be represented as single or multiple images, enabling the use of CNNs. Combining both the research field of population genetics, and the research field of computer vision and image processing. The problem of finding certain locations within the genome can now be regarded as a pattern recognition problem. This translation provides a way to make use of CNNs for the field of population genetics as presented earlier by Flagel et al.[14].

Twenty years ago central processing units (CPUs) used to be the mainstream options for implementing machine learning algorithms, back then matrix multiplication and factorization techniques were not widely used[15]. Nowadays graphical processing units (GPUs) are popular due to their high level of parallelism and even natively supported by many CNN software libraries such as TensorFlow[16]. Another hardware solution that could make use of the distinct computational features of a CNN is the field-programmable gate array (FPGA), and can be a strong competitor to the high-speed GPU implementation as shown by Wang and Gu[17]. Considering that genomic data-sets are quite large, and therefore vast amounts of images are needed to represent a single genomic data-set the need for fast acceleration techniques are of great importance to provide reasonable execution times.

## 1.1 Motivation

The identification and locating of (new) genetic variants and evolutionary forces in genome-wide data-sets have a relation to topics such as complex diseases as autoimmune diseases[18]. In genome-wide data sets, various genetic variants and evolutionary forces can be identified these variants for example transcription start sites (TSSs), splice sites, positive selection, promoters, enhancers or positioned nucleosomes[19]. The presented research focuses on the detection and localization of one of the before-mentioned evolutionary forces called positive selection. Positive selection is a major contributor to the shaping of the evolution of a species. The possibility to accurately locate positive selection can yield information regarding forces that drive adaption. Which could be for example used for identifying drug-resistant mutations in pathogens. The region that was under positive selection is also denoted as a selective sweep, and the detection of this region is named sweep detection.

Locating a region that has undergone positive selection can be done in various ways. One approach is to use signatures left by positive selection and resulting forces such as the hitchhiking effect[20]. While in cases where the positive selection is easily identifiable (strong selection) performance of sweep detection tools is good. The performance of sweep detection tools deteriorates significantly for cases where positive selection becomes harder to identify (weak selection) as shown by Alachiotis and Pavlidis[21]. Signature-based sweep detection approaches assume that certain evolutionary effects are present and are caused by positive selection, while other evolutionary effects could also create similar signatures. Signature-based methods and tools counter this problem by adjusting for these faulty evolutionary factors, but considering that these faulty signatures exist could make signatures based methods and tools more error-prone.

Instead of using signature-based methods and tools for sweep detection, another option is to use methods and tools based on machine learning. One of the major advantages of deploying machine learning methods and tools instead of signature-based methods and tools is the ability to automatically identify patterns in data, this is especially true when expert knowledge is incomplete, inaccurate and/or when data is too large to be manually checked as stated by Yip et al.[22]. Using machine learning and more recently deep learning methods and tools for sweep detection has multiple differences with regards to more conventional approaches (signature-based). Such a difference is for example by employing a more black-box technique instead of the more white-box techniques now employed. This results in a model that trains itself to find features to determine if a genomic region has undergone positive selection. Especially Convolutional Neural Networks (CNNs), a deep learning technique have proven to be able to yield great results for problems such as sweep detection as discussed by Flagel et al.[14], Kern & Schrider[23] and Torada et al.[24].

While CNNs promise great performance they are not yet to my knowledge applicable to whole-genome data-sets (able to locate positive selection within a larger genomic data-set) as the signature-based approaches are. Meaning that there exists the need for a CNN based method that can perform whole-genome sweep detection. A major challenge concerning the whole-genome CNN method is the number of images needed to represent the whole data-set. The number of images required to represent all positions of for example an ms generated data-set quickly goes into the thousands. Consider the numbers provided by Wang and Gu[17], who achieved 31.7 frames per second (fps) with a commercial GPU for the YOLOv3[25]. When thousands of images need to be processed times quickly go towards multiple minutes or even hours depending on the length of the genomic data-sets. Wang and Gu[17] also showed great improvements (around 2.4 faster) by deploying Field Programmable Gate Arrays (FPGAs) for CNN acceleration, this is to my knowledge not yet available for CNN sweep detection methods and tools. The application of sweep detection methods based on CNN's acceleration techniques could yield great benefits due to the amounts of images and computation required. The ASDEC (Accurate Sweep Detection Enabled by a CNN) framework presented in this research can process whole-genome data-sets and uses acceleration techniques to increase performance.

## 1.2 Research question

Sweep detection methods and tools based on signatures do not yield the desired results for genomic data-sets where weak selection is present[21]. To increase the performance with regards to these weak selection data-sets CNN based methods and tools can be applied for sweep detection[14]. The current CNN based sweep detection methods and tools[14], [24], [23] are to the best of my knowledge not yet able to perform sweep detection on whole-genome data-sets, by enabling a CNN based sweep detection method to process whole-genome data-sets, a comparison with signature-based methods and tools can be made possible. A disadvantage of CNN based sweep detection methods and tools is the computational intensity, acceleration with the help of either an FPGA or Deep-learning Processing Unit (DPU) can greatly reduce the execution time.

### Main research question

Can an accelerated convolutional neural network outperform currently available methods and tooling, regarding sensitivity, accuracy, success rate, and execution times for the detection of a selective sweep?

### research sub-questions

1. Is it possible to design a CNN that can be used in the detection of selective sweeps?
2. How should the CNN be modelled to achieve sufficient results with regards to the qualitative evaluation?
3. Which parts of the model should be performed on the accelerator?
4. What is the performance potential for acceleration of the developed sweep detection method?

## 1.3 Contributions

Due to the absence of CNN based methods and tools for sweep detection within whole-genome data-sets, this research introduces a framework Accurate Sweep Detection Enabled by a CNN (ASDEC). ASDEC provides a complete workflow starting from coalescent[8],[9] simulations to a final list of probabilities including performance metricises (sensitivity, accuracy, success rate, and execution times). With the help of the ASDEC framework, further research is performed by providing a manual neural architecture search (NAS) to determine the best performing CNN network architecture regarding various free parameters. Also, one of the major disadvantages of a CNN based method is tackled by providing acceleration techniques for the ASDEC framework in combination with a model taken from the NAS.

The contributions are the following:

- The ASDEC framework provides a CNN based method to perform sweep-detection on whole-genome data-sets. ASDEC includes all steps consisting of data generation, custom data inputs, image generation, data pre-processing, CNN model training, CNN model inference, post-processing, and determining performance metrics. The framework is based upon TensorFlow[16] and Keras[26], and allows custom models to be easily implemented within separate files and given using input parameters. ASDEC has support for multiple coalescent simulation software tools, supports multiple types of post-processing, and is fully configurable by a user. The ASDEC software is available at https://github.com/SMattieS/ASDEC.
- With the help of ASDEC, a manual NAS is performed consisting of various CNN designs with a total of 48 CNN designs, 190 trained models, and 380 inferences. Resulting in a final CNN architecture dubbed Sweepnet, which was the top performer concerning the given metrics of sensitivity, accuracy and success rate.
- ASDEC being computationally expensive and having native support for both CPU and GPU, because of the support directly provided by TensorFlow. Still more computational operation (operations per second) can be achieved by employing an FPGA and/or DPU acceleration solution, which is achieved with the help of Vitis AI[3]. Quantization is performed on the Sweepnet model, and a theoretical possible throughput is calculated with a given architecture. The ASDEC hardware implementation is available at https://github.com/SMattieS/ASDEC_HARDWARE.

## 1.4 Report outline

Chapter 2 provides an introduction to the subjects of genetics, selective sweeps, artificial intelligence, artificial neural networks, convolutional neural networks and accelerators. Chapter 3 describes related work including both signature-based methods and CNN based methods. Chapter 4 elaborates the method, describing and discussing all the different techniques used for sweep detection. Chapter 5 focuses on the NAS performed with the help of the developed ASDEC framework. This chapter leads to a sufficient CNN architecture which is determined with the help of predefined metrics. Chapter 6 describes the hardware acceleration and performance of a chosen model based on Chapter 5. Chapter 7 gives the final results with regards to ASDEC and acceleration. Lastly, chapter 8 concludes the thesis and provides a conclusion in combination with possible future work.

# 2 BACKGROUND

This Chapter describes the relevant background regarding population genomics, sweep detection and positive selection, artificial intelligence (AI), artificial neural networks (ANN), CNNs, and accelerators.

## 2.1 Population genetics and genomics

The field of population genomics provides the basis for the performed research and is a growing field stemming from almost a century of developments within the study of population genetics[6]. The research field of population genetics studies genetic variation in populations[6]. Genetic variance is present within the genome of for example a human population. Here a genome stands for all genetic material of an organism. If for example two individuals out of a single population are taken and a variation between them is present each variant is called an allele. This variation could be that one individual has for example blue eyes, which is the first allele and the other individual which has brown eyes having the second allele. To localize the genetic variance of the given example or any other genetic variance in a population, the examination and modelling of changes in the frequencies of genes and alleles over space and time is needed[27].

The occurrence of genetic variance is due to differences in the genetic length and/or the nucleotide content[6]. The nucleotide content is divided into four distinct types (adenine (A), thymine (T), guanine (G), and cytosine (C)). Furthermore, various genes that are present within the population are denoted as a single-nucleotide polymorphism (SNP). SNPs are defined in the following way: ”A single-nucleotide polymorphism (SNP) is a single genetic code variation (i.e., polymorphic). Although multiallelic SNPs do exist, the SNPs are usually biallelic (two alternative bases occur) and require a minimum frequency (>1%) in the population”cited from [28]. This is illustrated in Fig. 2.1.

The field of population genetics had accumulated a substantial amount of mathematical theory by 1966[29]. Pioneers within this field include Fisher, Haldane and Wright, but the research field also greatly benefited from a collection of large amounts of data provided by laboratory research. Charlesworth and Charlesworth provided a view into the fields growth and changes starting from 1966 going to 2016 [29].

Where the field of population genetics focuses on the study of genes and the way alleles are affected by evolution. Population genomics focuses more on the study of all genomic material of an organism and for example large-scale comparison of Deoxyribonucleic acid (DNA) and Ribonucleic acid (RNA) sequences. The field of population genetics and the research field of population genomics are related to each other. Both research fields play their role within health and diseases. Within the remainder of this section, the focus is on modern population genomics as described in the book Statistical Population Genomics by Dutheil[6].

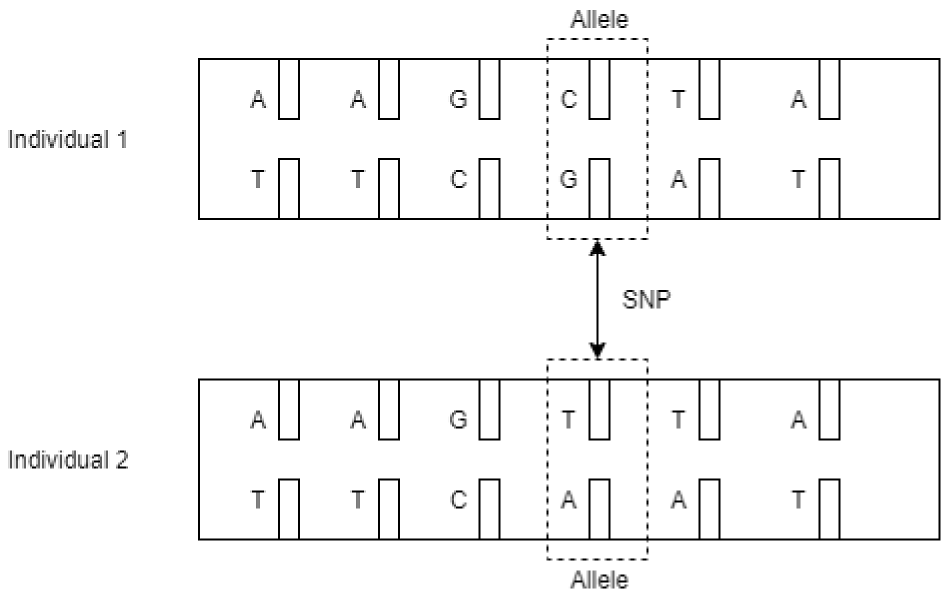


Figure 2.1: Example of Single Nucleotide Polymorphism (SNP), given two individuals within the same populations

### 2.1.1 Mutations

An important molecular event is a mutation which is a change in the genome. A mutation can be of various types including the substitution of a nucleotide into another nucleotide. But also the addition or removal of one or several nucleotides and lastly the multiplication of a part of the genome. New alleles are created when a genome has undergone mutation. Mutations can occur on different locations within the genome, a location within a genome is better known as a locus.

### 2.1.2 data preparation and the infinite site model

Obtaining the sequenced data from populations of interest is another challenge within the research field of population genomics. First of all the samples of the genomic material should be obtained from the population of interest. With the help of this genomic material various techniques for obtaining the sequence can be deployed such as the ‘shotgun‘[30] technique deployed for the initial sequencing of the human genome[7]. These sequencing techniques takes as input raw genomic material (for example DNA), and provide as output an assembly containing the different nucleotides.

*”...ACCGTAAATGGGCTGATCATGCTTAAACCCTGTGCATCCTACTG...”*

Then Multiple Sequence Alignment (MSA) is applied to create multiple biological sequences of a similar length, this is done to study the evolutionary relationships between the sequences. Within the MSA data, different mutations can be present, and these mutations can be modelled.

The model deployed for describing the mutations that occur during a given time frame is the infinite site model (ISM)[31] of molecular evolution. Within the ISM, the term ”sites” is defined as a single nucleotide pair. Although the author Kimura[31] also denotes that the term ”sites” can within the theory still be appropriate for a small group of nucleotides. The infinite site model makes various assumptions, first of all, during the time frame of evolution each locus has undergone at most one mutation. Secondly, it is assumed that each mutation creates a new allele and finally that no backward or reverse mutations exist[6]. Here a backward mutation means

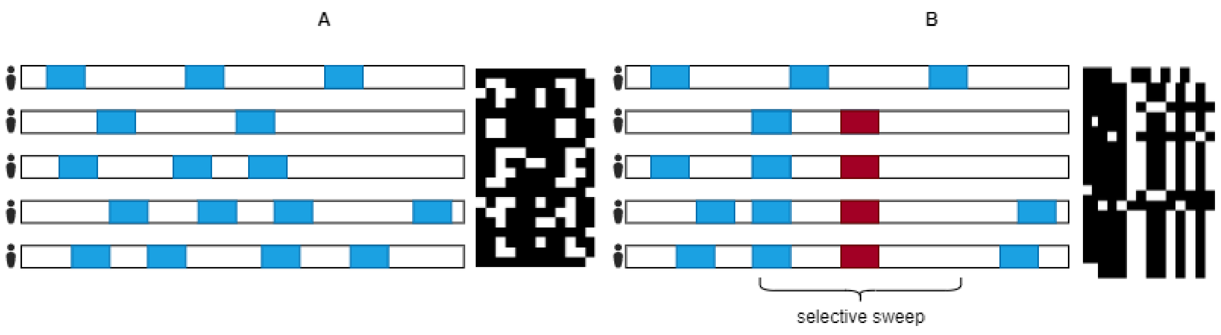


Figure 2.2: A: shows neutral genomic data and B: shows partial selective genomic data. Both show on the left image: in blue the derived (mutant) alleles and red the beneficial/positive alleles. Both show on the right image: in white the derived alleles and black the ancestral alleles.

that the process is reversed. This causes a site in a mutant gene to restore to its original state. The normal or reversed site is also denoted by the term wild type.

### 2.1.3 Positive selection and selective sweeps

Positive selection is a driving force in shaping the evolution of species. Alleles that are under positive selection are alleles that increase fitness. These alleles increase in frequency within a population over time. Until at some point in time, the fitness increasing (beneficial) alleles become fixed within the population, thereby substituting the non-fitness increasing alleles. Information provided by the identification of genes affected by positive selection can yield information about forces driving adaption[32]. But also provides practical information about for example the identification of drug-resistant mutations in pathogens [33] and the design of more effective drug treatments [34]. Not all selection is by definition positive. A different type of selection is negative selection, also known as purifying selection. Negative selection decreases the frequency of alleles that impair fitness. Both positive and negative selection decrease the genetic diversity.

When positive selection is present meaning that an allele is favoured by natural selection. The advantageous allele spreads throughout the population and produces a loss of variation near the locus of positive selection. The loss of variation is explained by the closely linked neutral alleles also increasing in frequency because they were originally linked to the beneficial allele, while the remaining non-linked neutral alleles decrease in frequency. The effect is denoted as the hitchhiking effect[20]. This effect can be described as a neutral allele getting a lift by a closely linked beneficial allele. The genetic diversity in the region of the allele with positive selection is swept away referring to this region by the term of ”selective sweep”.

Finding sub-genomic regions associated with selective sweep yields information about an individual and or species history. Two major types of regions are defined, first of all, a neutral region that has experienced no positive selection and therefore does not show hitchhiking effect. The second region is the selective region (the strength of the selective sweep within the selective region can vary) which has experienced positive selection and also shows the hitchhiking effect. The difference between neutral and selective regions in genomic data is illustrated in Fig. 2.2. In Fig. 2.2 neutral genomic data (A) and selective genomic (B) data are compared using a schematic example. Detection of positive selection is possible due to three distinct signatures left in the genome by a selective sweep as defined in [35]:

1. Shift in the site frequency spectrum (SFS) toward low- and high-frequency derived variants

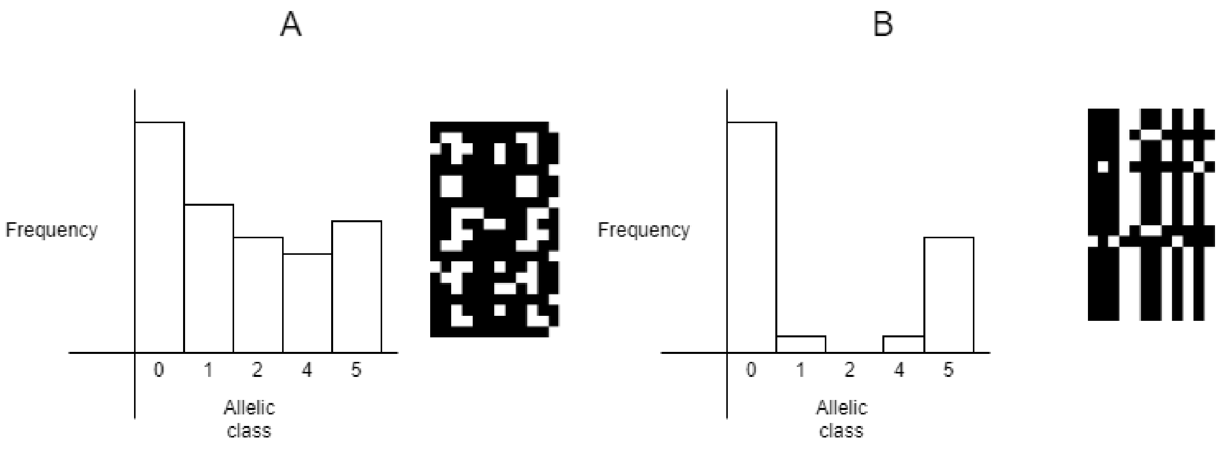


Figure 2.3: In the monochrome image (polymorphic table) in white the derived alleles and black the ancestral alleles. The plot next to the polymorphic table's show a schematic SFS, here the effect of a selective sweep is observed (A no selective sweep B selective sweep).

2. Reduced genetic diversity in the region surrounding the locus of positive selection
3. A distinct pattern of linkage disequilibrium (LD)

The first signature of a selective sweep is a shift towards the high- and low-frequency derived variants in the SFS, and is shown in the following studies [36],[37]. The causes of the shift in SFS is (as described earlier in this chapter) "the closely linked neutral alleles also increasing in frequency because they were originally linked to the beneficial allele. While the remaining non-linked neutral alleles decrease in frequency". An schematic example is provided in Fig.2.3, which draws inspiration from [35]. The second signature is simply the loss of variation associated with positive selection, by determining the amount of lost variation sweep detection is possible. The third and final signature of a selective sweep denotes a distinct pattern of linkage disequilibrium (LD) where it is defined as. "Linkage disequilibrium (LD) is the nonrandom association of alleles of different loci. There is no single best statistic that quantifies the extent of LD. Several statistics have been proposed that are useful for different purposes." cited from [38].

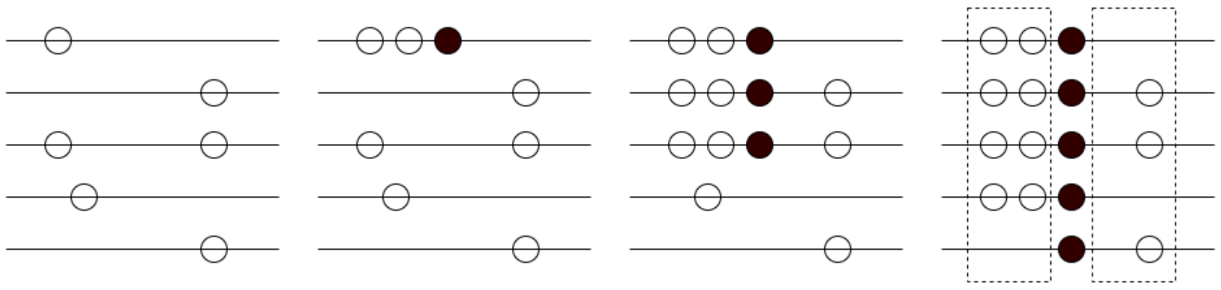


Figure 2.4: LD signature around a region of positive selection, the beneficial mutation is shown in black and becomes fixed in the population. Hitchhiking is affecting the neutral variants and finally, two regions on each side of the beneficial mutation are provided with the help of the dotted lines.

The pattern in LD describes a pattern that emerges between SNPs around the area of positive selection. This pattern emerges during the process of fixation of the positive selection within the genome and is defined by an emerging LD on both sides, of the area of positive selection. While decreased LD levels are observed between the sites found on the different sides of the positive selection area. This process is graphically represented in Fig.2.4, which draws inspiration from [35]. When a single rearrangement within the genome (recombination event) happens the existing SNPs on the side of the selective sweep can escape the sweep. While the SNPs on the opposite side need at least two recombination events to be able to escape. Given that no relations between different recombination events exist, the decreased LD levels on the different sides of the area of positive selection are present[35].

For more information please refer to the book of Dutheil[6], which was extensively used in the section.

## 2.2 Artificial Intelligence

Artificial Intelligence (AI) is the simulation of intelligence by a machine and is an overarching name for a field such as Machine Learning. Machine learning is a method of data analysis that enables a computing system to learn how to solve a problem, instead of providing all information explicitly by for example a programmer. Within Machine Learning different techniques exist such as support vector machines, data-clustering and artificial neural networks (ANN) just to name some. This research focuses on artificial neural networks and then specifically on a subset of artificial neural networks called Deep Learning. To keep an overview of all these different relations see Fig.2.5 [1].

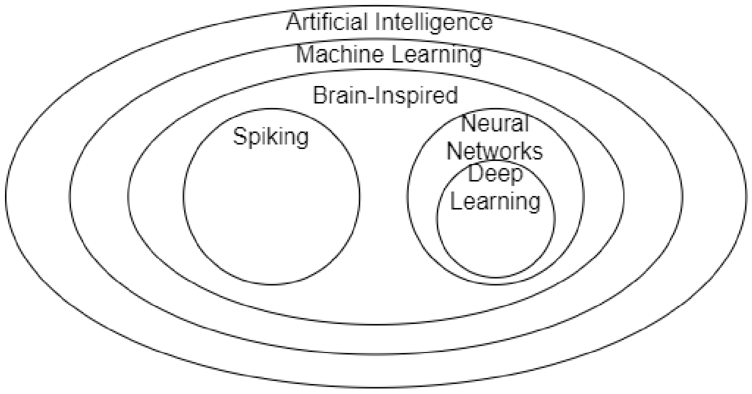


Figure 2.5: Deep Learning in the context of Artificial Intelligence[1]

### 2.2.1 Artificial Neural Networks

An artificial neural network draws its inspiration from the biological working of the human brain shown in Fig.2.6, hence the name. An ANN is built out of artificial neurons that resemble simplified neurons in a brain. To perform complicated tasks these artificial neurons are connected to build complex networks. ANNs are in stark contrast with the more conventional computational paradigms where an explicit set of instructions are provided, whereas an ANN enables itself to learn the solution to the problem by studying a given set of examples[2].

In a paper by McCulloch and Pitts[39], a mathematical expression for a single artificial neuron is presented. Were a single artificial neuron can be expressed by a set of input values ranging from $x_1$ to $x_n$ as a vector **x** $= [x_1 ... x_n]$. First of all the inputs are multiplied by a weight denoted by $w_i$, and the results of the multiplications are summed together. The final result after summation ($a$) is the input for the activation function (denoted by $g$). The final result $z$ is the output of the activation layer. The complete equation is given in Equation.2.1 and is shown in a more graphical way in Fig.2.7. Within an artificial neuron one special type of weight exists, which is called the bias $w_0$. The bias is an extra input whose value $x_0$ is always equal to +1. With this

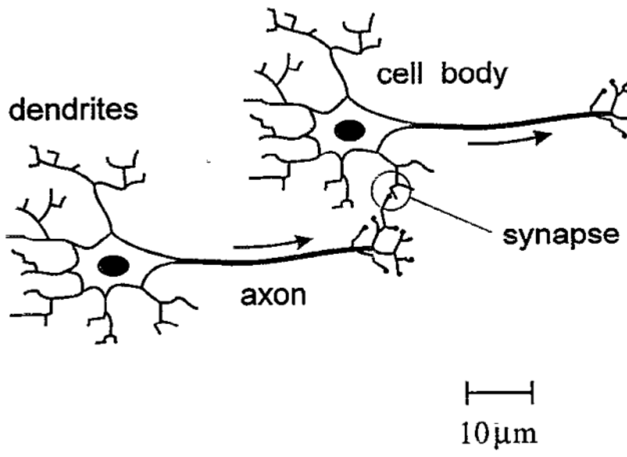


Figure 2.6: Two biological neurons schematic illustrated. To provide an input performed by dendrites, and upon firing the action potential propagates in the direction of the arrow along the axon. The interaction between two biological neurons takes place in the region called synapse[2].

knowledge Equation.2.1 can be rewritten to Equation.2.2 with the input vector **x** $= [1, x_1...x_n]$.

$$z = g(\sum_{i=1}^{n} w_i x_i + w_0) \tag{2.1}$$

$$z = g(\sum_{i=0}^{n} w_i x_i) \tag{2.2}$$

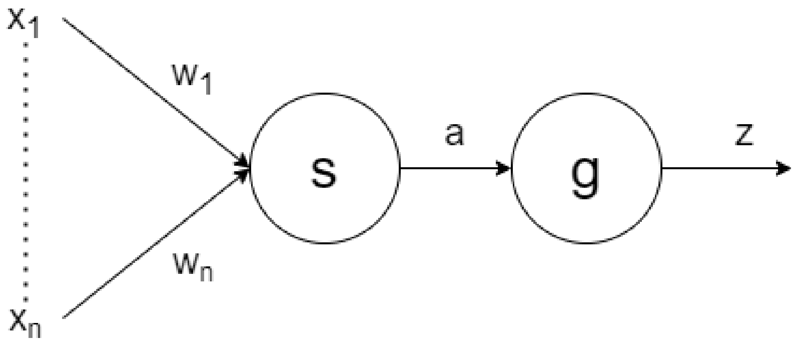


Figure 2.7: A single artificial neuron with an input vector of **x**, weight vector **w**, summation ($s$), and an activation function ($g$)

As mentioned earlier an artificial neuron includes an activation function ($g$). There exists various different activation functions, some well known examples are given in Fig.2.8. These include for example the identity activation function which simple passes the input ($x$) to the output ($y$), shown in Equation.2.3, and also the sigmoid activation function (Equation.2.6). One of the more well known and used activation functions is the Rectified Linear Unit (ReLU) activation function (Equation.2.4), which has a extended version called Leaky ReLU (Equation.2.5). Which is present in for example the YOLO[40] network. Lastly also the Tanh activation function is provided in Equation.2.7.

$$y(x) = x \tag{2.3}$$

$$y(x) = max(0, x) \tag{2.4}$$

$$y(x) = \begin{cases} \alpha x & if\, x < 0 \quad where\, \alpha \epsilon (0,1) \\ x & if\, x \geq 0 \end{cases} \tag{2.5}$$

$$y(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$y(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.7}$$

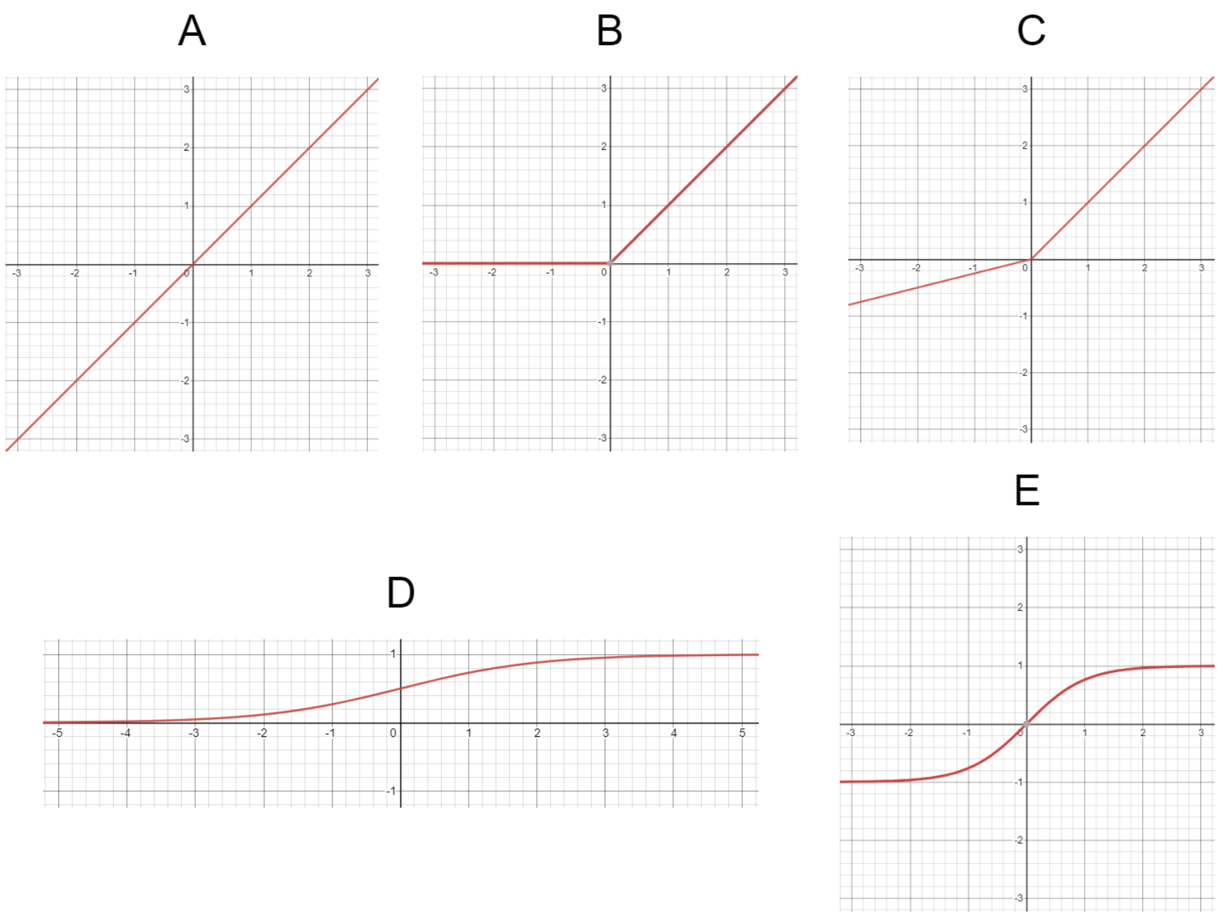


Figure 2.8: Activation functions, A identity, B Rectified Linear Unit (ReLU) , C Leaky Rectified Linear Unit (Leaky ReLU) with $\alpha = 0.25$, D sigmoid , E Tanh

Now with the definition of a single artificial neuron (Fig.2.7) it is possible to combine all these neurons into a network called an artificial neural network (ANN). By defining an input vector **x** $= [x_0...x_d]$, where $x_0$ is known as the bias and equal to 1 and a vector which is defining the weights **w**. The weight vector (**w**) has a more complex description, because it is present between for example the input and a hidden layer **w** $= [w_{10}^{(1)}..w_{md}^{(1)}]$. The activation function of all the artificial neurons present in each individual artificial neuron is denoted by $g(x)$ (see Fig.2.7). The final output of the ANN is the output vector **y** $= [y_1...y_k]$. A generic description of a feed-forward neural network is provided in Fig.2.9 in combination with Equation.2.8 which formulates the figure (Fig.2.9) [41].

$$y_k(\mathbf{x}, \mathbf{w}) = g(\sum_{j=1}^{m} w_{kj}^{(2)} g(\sum_{i=0}^{d} w_{ji}^{(1)} x_i)) \tag{2.8}$$

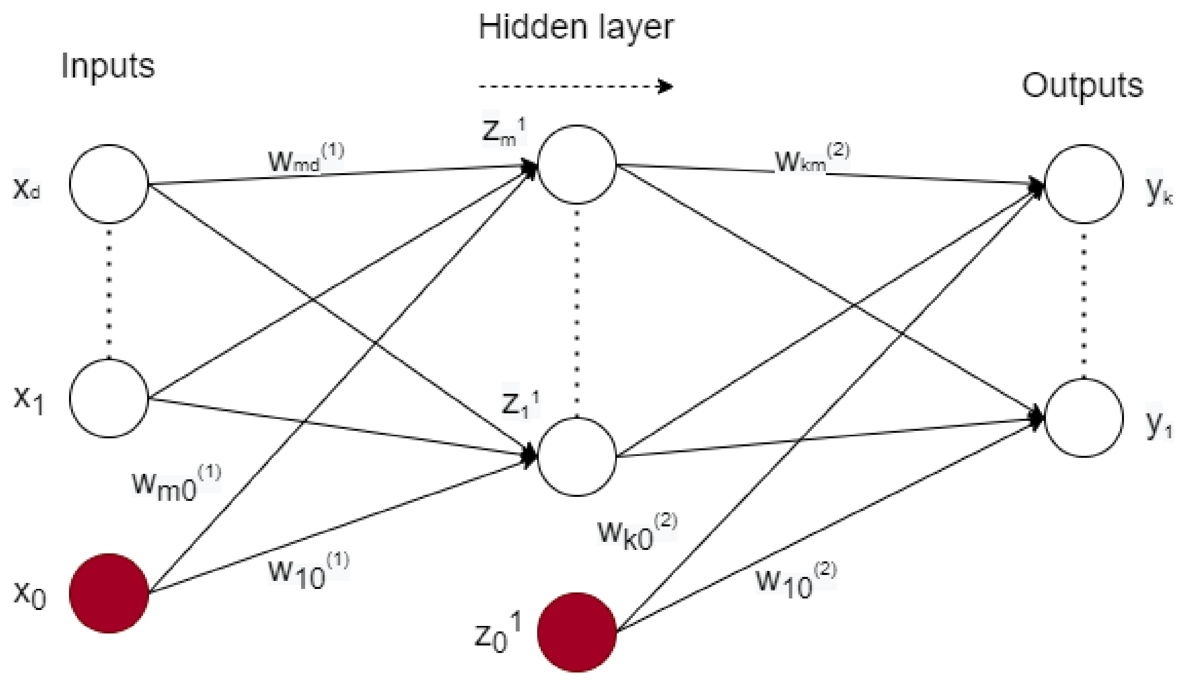


Figure 2.9: Feed-forward neural network

### 2.2.2 Types of machine learning problems

ANNs and more generally machine learning techniques are used to solve different types of problems. One of the simplest problems that can be solved by deploying machine learning techniques is the binary classification problem, here two classes are defined with their label. Consider for example a problem where the result is either true or false, here two classes are provided with the corresponding labels (true and false). An extension of a binary classification problem is the multi-class classification problem, here more than two classes are present. This class of problem is for example present when the output of our classification problem could not only be either true or false but also a class unknown. Until now a label had a fixed range of values or a fixed value, but labels could also have a continuous value, these continuous problems are known as regression problems[22]. An overview of the three different types of problems are also provided in Fig.2.10 [42].

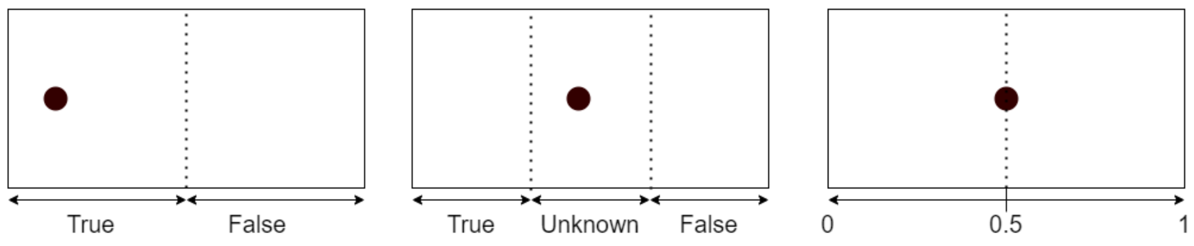


Figure 2.10: Different types of problems left shows binary classification problem, centre shows multi-class classification problem, right shown regression problem

### 2.2.3 Training and inference

ANNs know two different applications where the output of one of the applications is the input for the other one. These two applications are the training of a neural network and the inference performed by a trained neural network model. During the training of a model the weight ($w$) are normally first of all filled with random values. Then first a normal forward phase is performed where all inputs are simply propagated through the entire CNN. Then the backward phase is performed, here the gradients are back-propagated and weights are updated. A more extensive description of back-propagation is provided in [41]. With the trained model inference can be performed which provides a probability of an input belonging to a certain class (classification) or value (regression). When performing inference the weights ($w$) are fixed. Training and inference can be performed for all types of earlier mentioned problems (shown in Fig.2.10).

Different types of training are known (mostly) depending on the availability of training data. First of all supervised learning, where a model is constructed from a set of observed training data with a known label. Supervised learning assumes there exists sufficient data to be able to train a model. When no training data is available or the objective is to cluster data then unsupervised learning can be applied. Unsupervised learning relates data based on their distribution of feature values alone. Unsupervised learning could also yield great success if the amount of classes is still undefined. Both types of training are depicted in Fig.2.11. There also exist situations where supervised learning is preferred but simply not enough training data is available to sufficiently train a model. This could be tackled by combining both supervised learning and unsupervised learning by providing partly labelled data and partly unlabeled data, this technique is called semi-supervised learning.

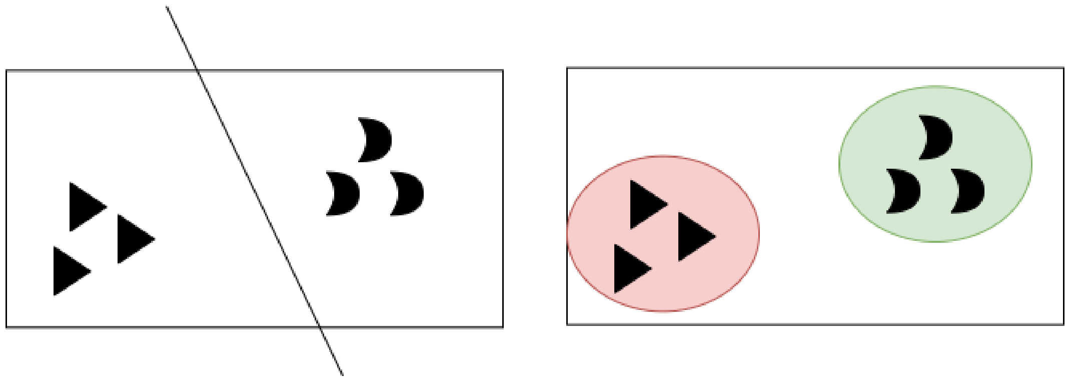


Figure 2.11: Supervised learning (left image) and unsupervised learning (right image)

### 2.2.4 Convolutional Neural Networks

One specific type of neural network is the Convolution Neural Network (CNN) this type of neural network has various applications in the image processing field. CNNs have had groundbreaking results in research fields related to pattern recognition in for example image applications[43]. Concerning this research, CNNs are mostly considered by their application in the image processing field, but this is not the only field where CNNs can be applied. Other fields include for example voice processing or voice recognition.

CNNs are built out of various layers, each layer has properties that could make it useful for certain applications. The size and complexity of a CNN are determined by the number of layers and the complexity and/or size of the layer itself. A CNN can be explained by describing the different layers where it consists of, the following sections will describe the layers considered in this research.

The layers of a CNN can be divided into two different classes feature-extraction and classification, both types of layers serve their specific purpose [44]. First of all the feature extraction layers contains various convolutional layers followed by pooling layers and some type of activation function (see Fig.2.8). Within these layers, the features in images are found which could later be used to perform classification. As a rule of thumb the deeper the feature-extraction layers go the more complex features are found. For example, the first feature-extraction layer only finds edges while a deeper feature-extraction layer finds complex shapes like faces. The other part of the CNN is the classification layers which are various fully connected layers also called the earlier discussed ANNs.

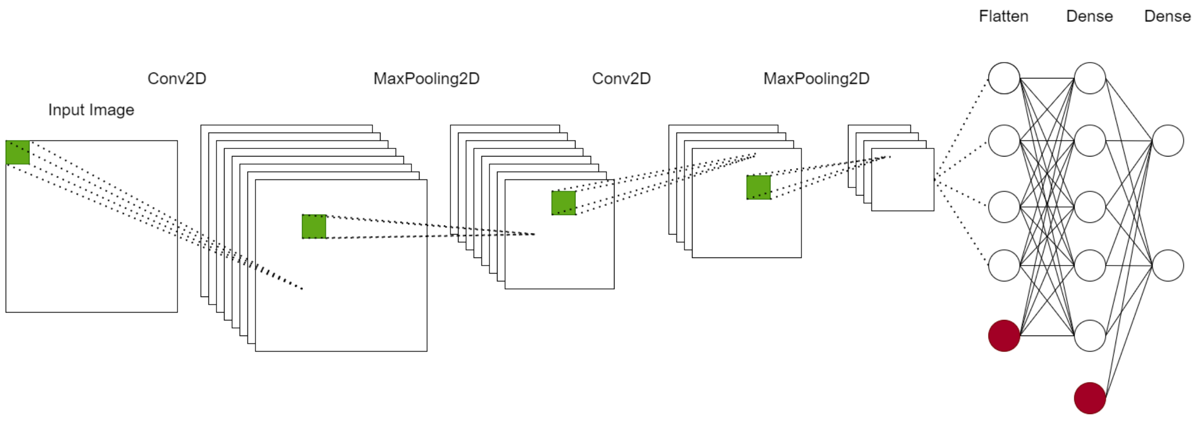


Figure 2.12: Schematic Convolutional Neural Network

### Re-scale layer

Providing a re-scaling step to the input before any other layers are done to increase the performance of the final model. The re-scaling provides further layers with inputs that are proportionally scaled by a scalar factor of the pixel value. Re-scaling is normalizing all values between [0,1] or [-1,1], one does not provide a great benefit over the other, but by exploring both options a preference can be determined. In Equation. 2.9 the re-scaling to the range of [0,1] is given and in Equation. 2.10 the re-scaling to the range of [-1,1] is given. Here $\bar{A}$ denotes the resulting matrix and $A$ denotes the original matrix. The other values are just applied piece-wise to all the elements in the input matrix.

$$\bar{A} = \frac{1}{255} \cdot A \tag{2.9}$$

$$\bar{A} = \frac{1}{127.5} \cdot A - 1 \tag{2.10}$$

### Convolutional layer

One of the layers that are essential within a CNN is the convolutional layer. Mathematically a convolution takes two functions as input and outputs a function, the convolution operator ($*$) is given in Equation.2.11 (continuous). Here the input function $f(x)$ is our image or the output of a previous layer, and the function $g(x)$ shifts over the input $f(x)$ and is called the convolutional filter. Within the convolutional filter, the different weights and parameters are present. The final output function is called the feature map. With regards to CNNs and this research, the input is always of a finite size with an input (image) ($I$) array of a certain dimension ($N$) and secondly as input a kernel ($K$) of arbitrary size. In Equation.2.12 (discrete) an output **i** $= [i_1 ... i_N]$ is calculated based on the input coordinates **j** $= [j_1 ... j_N]$[45]. In Fig.2.13 an example is provided with respect to an red, green, blue (RGB) input image with a kernel ($K$) of shape ($[2, 2]$), a stride shape of ($[1, 1]$) and "valid" padding. "Valid" padding simply stands for no padding applied, next to "valid" padding also "same" padding exists which applies padding around the edges of the data. Resulting in output with the same dimensions as the input. Only a single filter with bias is represented in the example (Fig.2.13) there could be multiple filters present that will correlate to our larger output dimension.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx \tag{2.11}$$

$$(I * K)(i) = \sum_j I(j)K(i-j) \tag{2.12}$$

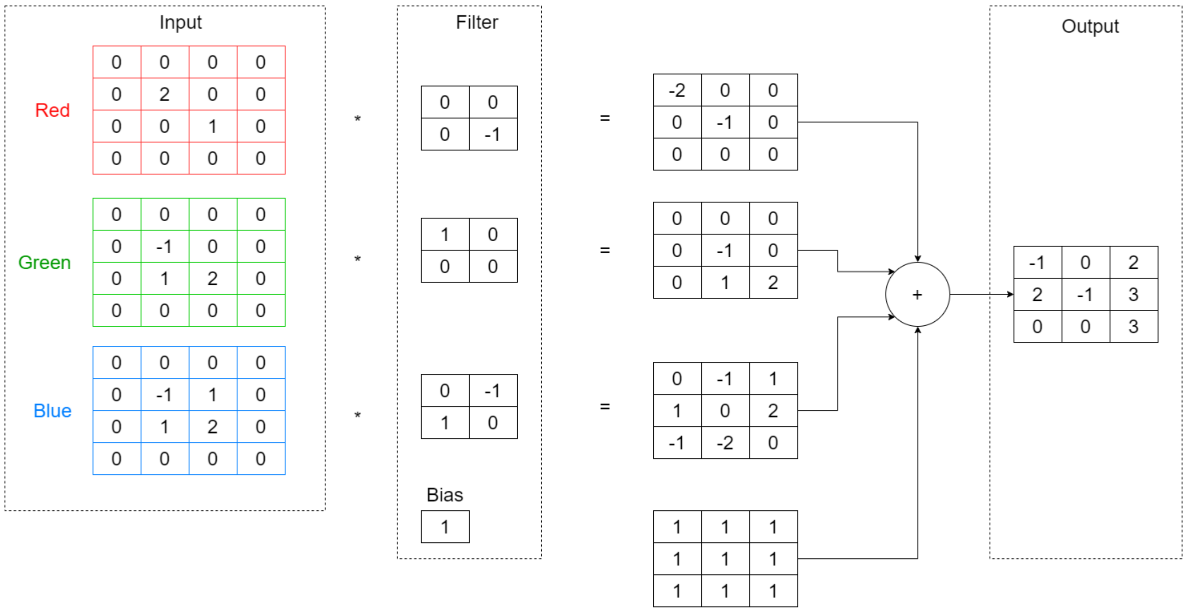


Figure 2.13: Convolution operation on a RGB-image, with a kernel ($K$) of shape ($[2,2]$), a stride shape of ($[1,1]$) and "valid" padding

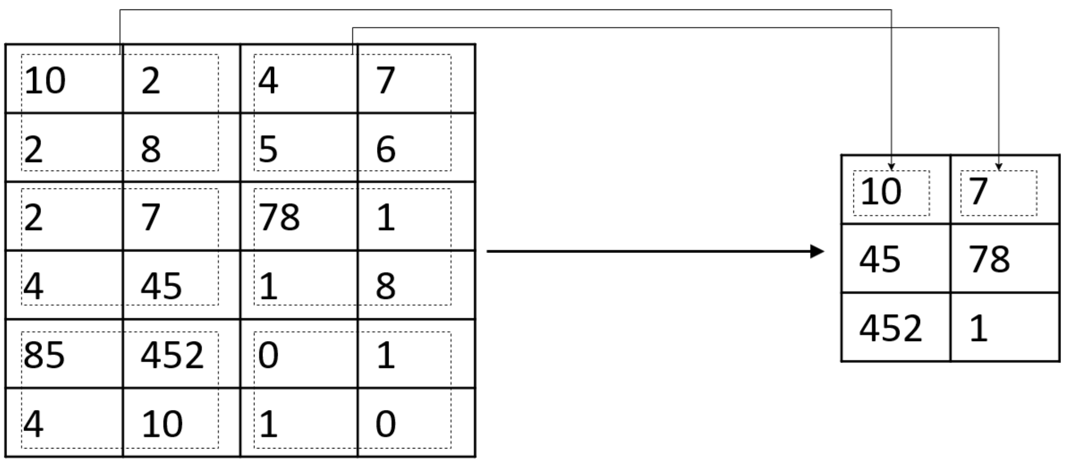


Figure 2.14: Max-pooling 2-Dimensional example, with a kernel of **d** $= [2,2]$ and a stride of **s** $= [2,2]$

### Pooling layer

Pooling layers provide a way to reduce the dimensionality of feature data. The pooling layers provide a down-sampling operation by taking for example the average or maximum of a certain subset, called the kernel ($d$) of the feature data[46]. The factor to which the amount of down-sampling is correlated is the stride ($s$). The stride is the amount of shift per kernel, so when the kernel size is 2, the stride is 3 and an input of **x** $= [1, 2, 3, 1, 2, 3]$ is provided the result is the output **y** $= [3, 3]$ when max-pooling is applied. This example applies max-pooling to a 1-dimensional example in Fig.2.14 a 2-dimensional example is provided with a kernel of **d** $= [2, 2]$ and a stride of **s** $= [2, 2]$. During training pooling layers do not provide any trainable parameters.

**Regularization**

The dimensionality of the data set is generally determined by the number of input and output units. Whereas the design of the neural network includes various free parameters such as the number of hidden units (denoted by $M$) in the network. These free parameters are adjusted in such a way that the best predictive performance is obtained. Note that $M$, in this case, controls the number of parameters (weights and biases) in the neural network. There exists an optimal value for $M$ that gives the best generalization results, corresponding to an optimal balance between under- and over-fitting[41]. Over-fitting corresponds to fitting too closely to a limited data set, and under-fitting is when the model neither fits well to the training and validation data. When under-fitting occurs it is (normally) concluded the model does not fit the data sufficiently. An approach to determine the optimal value for $M$ is to choose a relatively large value for $M$, and also adding a regularization term to the error function to control for the complexity obtained.

Various types of regularization exist the following ones are either mentioned in this research or common types of regularization and are explained in further detail in this section.

1. Drop out
2. Bath Normalization
3. L1 regularization
4. L2 regularization

**Drop out layer**
The dropout layer is a technique for regularization in an ANN and serves the purpose to prevent a model from over-fitting. A dropout layer achieves this task by generating a number ($\rho$) between 0 and 1 using a uniform distribution, for each neuron in a network. If $\rho$ is less than a given value $\alpha$, the neuron is dropped out from the network, including all its connections within the network [46]. After the dropout is completed both the forward and backward passes are computed on the network with the dropped neurons, this process is done for all samples in the training set.

**Batch Normalization**
During the training of an ANN, the distribution of each layer's inputs changes, as the parameters of the previous layers change. The effect caused is the slow down of the training process by the requirement of a lower learning rate and careful parameter initialization. To tackle this problem batch normalization performs normalization on the layer inputs (denoted by $x$). By including batch normalization into the model architecture, a higher learning rate can be used in combination with a less careful initialization[47].

Batch normalization performs the following transformation on the input($x$), resulting in the output ($z$) see Equation.2.13[46]. Here mean-variance normalization on the input $x$ using $\mu$ and $\sigma$, the method they apply is the exponential moving average. For the current layer over the training set. Moreover linearly scaling is applied by $\gamma$ and shifting by $\beta$ [46].

$$z = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta \tag{2.13}$$

**L2 and L1 regularization**
L2 regularization is a way for regularizing an ANN. L2 regularization computes the L2 norm of the weights and adds the results to a loss function. The resulting function is shown in Equation.2.14, here **W** denotes the weights of all layers. Instead of the original method of minimizing $L(\mathbf{x})$. The $\|\mathbf{W}\|^2$ is the L2 norm of the weights and $\lambda$ can be used to control the amount of L2 regularization applied to the loss function. Therefore the value of $\lambda$ should be considered to not create weights too close to zero or too large [46].

$$L_{L2}(\mathbf{x}) = L(\mathbf{x}) + \lambda \|\mathbf{W}\|^2 \tag{2.14}$$

Besides L2 regularization also L1 regularization exists, here the same principle is applied only concerning the use of an L1 norm instead of an L2 norm. This leads to the following Equation.2.15, here the same user control parameter $\lambda$ is present and serves the same purpose. L1 regularization differs from L2 regularization by the reason it can but not always produces a sparse weight vector, where some weights are very close to zero or even exactly zero. L1 regularization also lends itself very well for classification tasks. Important to note is that L1 and L2 regularization can also be combined[46].

$$L_{L1}(\mathbf{x}) = L(\mathbf{x}) + \lambda |\mathbf{W}| \tag{2.15}$$

**Flatten and dense layer**

The flatten layer marks a point in the architecture of a CNN that moves from the feature-extraction layers into the classification layers. From this point forwards the features extracted out of an image are used to perform classification. The flatten layers reshape the data into a 1D vector as illustrated in Fig.2.15. For example, if the input layer of a flatten layer is $[(20, 50, 3)]$ ($[(imageheight, imagewidth, amountofcolourchannels)]$) the output of the flatten layer is of size $[(3000)]$. The output of the flatten layer is normally the input to a dense layer, the dense layer is simply a fully connected layer also known as an artificial neural network (ANN) which was already discussed in detail in this chapter.

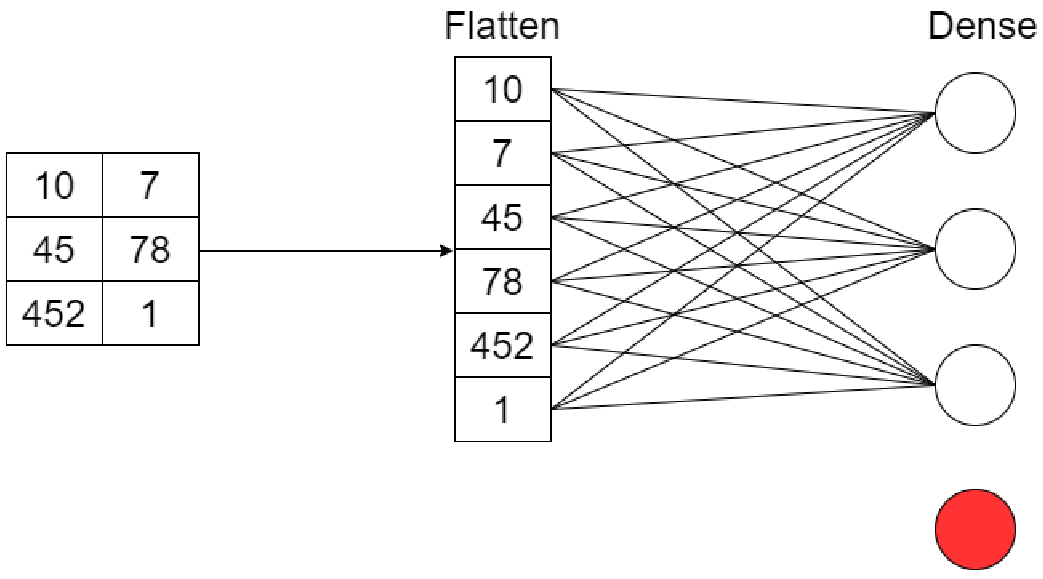


Figure 2.15: Flatten and dense layer, where the flatten layers reshape the data into a 1D vector and the dense layer is simply a fully connected layer also known as an artificial neural network (ANN)

For more information please refer to the books of Bishop[41], and Aghdam & Heravi[46], which were extensively used in this section.

## 2.3 Convolutional Neural Network Accelerators

Developments in the field of CNNs have led to the support of various hardware platform implementations which can perform training and/or inference. These hardware platforms include central processing units (CPU), graphical processing units (GPU), Application-specific integrated circuits (ASIC) and Field-programmable gate arrays (FPGA). There also exists a special type of ASIC introduced by google back in 2017 named a tensor processing unit (TPU)[48] this specific architecture is aimed towards data-centre applications.

Both a CPU (multi-core) and GPU are present in most to almost all conventional computers nowadays, both are natively implemented in CNN frameworks such as TensorFlow[16], Pytorch[49] and Caffe[50]. TensorFlow is an end-to-end open-source platform for machine learning, introduced by Google and therefore also includes support for Google's TPUs.

### 2.3.1 CNN FPGA acceleration

To perform the acceleration of a CNN on an FPGA various methods can be deployed, such methods include designing an intellectual property (IP) from scratch or using already existing IPs. Implementing and designing IPs can be done with for example tools such as Vivado[51] by Xilinx. But IP design and implementation can also be done by the use of High-Level Synthesis (HLS) tooling which provides the user with a hardware implementation given as input a higher-level language such as C or C++. HLS tooling includes for example Vitis HLS[52], Catapult[53], and LegUp[54] just to name some. A completely different approach is to solely make use of predefined IPs and abstracting the FPGA design further. An example of tooling that can perform this task is Vitis AI[3] the successor of DNNDK[55] both are developed by Xilinx. The main advantage of this approach is that highly optimized IP cores, tools, libraries, and models are available and therefore it is possible to greatly reduce design time.

Vitis AI performs AI inference acceleration on different Xilinx hardware platforms, including both Edge devices and Alveo accelerator cards. A workflow for Vitis AI is presented in Fig.2.16 and shows three distinct blocks. First of all the build model flow includes the conversion from 32-bit floating-points weights and activation's to fixed-point such as INT8. Models designed in either TensorFlow, Caffe and Pytorch are supported. After quantization of the model, the model can be compiled here the quantized model is transformed into Deep-learning Processing Unit (DPU) instruction. Here the hardware architecture connects with the model and the final compiled model can be deployed by building the software in either C, C++ or Python on the targeted hardware platform/architecture. Xilinx also presents some benchmarks for some well-known applications in [56].

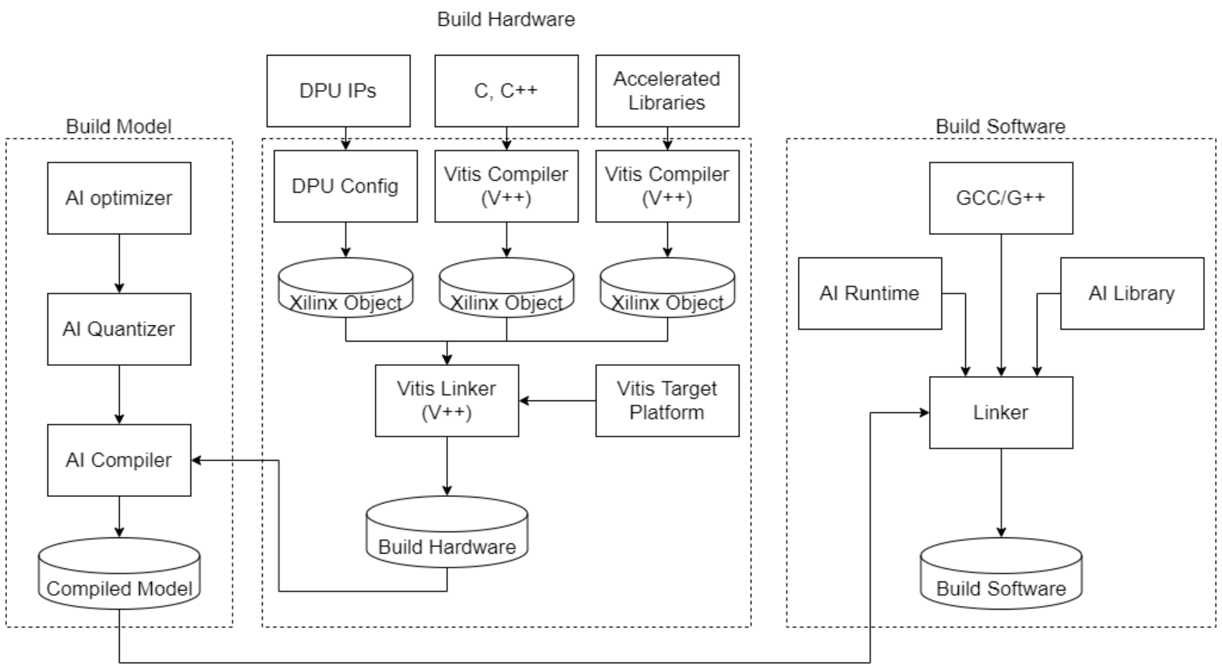


Figure 2.16: Vitis AI workflow[3]

# 3 RELATED WORK

With regards to the related work, two different types of sweep detection methods are considered. First of all various signature-based sweep detection methods and secondly various CNN based sweep detection methods.

## 3.1 Signature based sweep detection

Determining the locus of positive selection within a genomic data-set is an important research topic. This topic has seen development and applications in the past years, including numerous statistical methods to identify these loci within a genomic data-set[18]. Signature-based sweep detection methods and tools use signatures (explained in Chapter 2) to perform sweep detection, the methods and tools considered with respect to this research are: SweepFinder2[57], SweeD[58], OmegaPlus[59] and RAiSD[21].

### 3.1.1 SweepFinder2

SweepFinder2[57] is based upon SweepFinder[60], which uses a composite likelihood ratio (CLR) test for positive selection developed by Kim and Stephan[61]. The CLR consist of the likelihood of a sweep at certain loci in the genome divided by the neutral model[35]. The information required for the CLR is provided with the use of an empirical background frequency spectrum (based on SFS). SweepFinder2 extends upon Sweepfinder with some improvements such as accounting for background selection (caused by negative selection), local recombination rate, and general improvements in stability, flexibility and performance[57]. While SweepFinder2 still performs sweep detection based on the statistical framework provided by SweepFinder.

### 3.1.2 SweeD

SweeD (Sweep Detector)[58] focuses on rapid sweep detection in the whole genome, based on the SFS signature and is also represented as an extension on Sweepfinder[60]. The promise of SweeD is mostly aimed towards being less computationally expensive and being able to use multi-core processors resulting in vastly improved execution times. Overall SweeD is a more stable, and scalable implementation of Sweepfinder.

### 3.1.3 OmegaPlus

OmegaPlus[59] uses the LD signatures to determine the locus of positive selection and focuses on the rapid detection of selective sweeps within the whole genome data-set. The author of OmegaPlus reported up to two orders of magnitude faster execution than existing programs in combination with up to two orders of magnitude smaller memory requirements and improved scalability. OmagePlus uses the $w$ statistic first proposed by Kim and Nielsen[62]. The $w$ statistic is given in Equation.3.1. The omega statistic assumes a genomic region with $S$ SNPs that is split into two sub-regions, one on the left denoted by $L$ and one on the right denoted by $R$ both with $l$ and $S - l$ SNPs. The correlation coefficient between sites $i$ and $j$ are presented by the common measure of LD denoted as $r_{ij}^2$. Positive selection is characterised by a high $w$ value.

$$w = \frac{((\frac{l}{2}) + \frac{S-l}{2})^{-1})(\sum_{i,j \epsilon L} r_{ij}^2 + \sum_{i,j \epsilon R} r_{ij}^2))}{(l(S-l))^{-1} \sum_{i \epsilon L, j \epsilon R} r_{ij}^2} \tag{3.1}$$

### 3.1.4 Raisd

Raisd (Raised Accuracy in Sweep Detection)[21] is one of the latest tools developed by Alachiotis and Pavlidis. The authors of Raisd report a higher sensitivity and accuracy than the current state of the art(SweepFinder2, OmegaPlus, and SweeD) for a large set of the selected tested cases, while the computational complexity is greatly reduced (up to 1000 times faster execution) in combination with negligible memory requirements. Where the tools discussed until now only used a single signature to locate positive selection, Raisd introduces a composite statistic based on multiple signatures. The composite signature is called the "$\mu\, statistic$" (Equation.3.2) and relies on all three sweep signatures. The $\mu\ statistic$ assumes a data-set ($D$) consisting of $S$ individuals, secondly, the amount of SNPs in a genomic region is denoted by $D_{SZ}$ and its length is denoted by $D_{ln}$. Each of the signatures is computed separately and all results are summed in combination with the length of the genomic region ($D_{ln}$). The final $\mu\ statistic$ is computed for a given amount of windows ($W$) with a fixed-size SNPs which is further split into two non-overlapping sub-regions of equal amounts of SNPs denoted by left($L$) and right($R$). For a complete description of the computations of all individual signatures see [21].

$$\mu_t = D_{ln} \times \mu_t^{VAR} \times \mu_t^{SFS} \times \mu_t^{LD} \tag{3.2}$$

## 3.2 CNN based sweep detection

Besides the signature-based approaches, another relevant approach is the use of CNNs. Various methods and tools which use CNNs to perform sweep detection are already developed. The methods and tools presented in this section are: ImaGene[24], a tool by Flagel et al.[14], and DiploS/HIC[23].

### 3.2.1 ImaGene

ImaGene[24] is a tool that applies CNNs (supervised learning) on population genomic data for the detection and quantification of natural selection. The classification classes considered by ImaGene are not only divided into neutral and selection but also consider the selective strength ($S$). The number of classes to classify are variable and either binary classification or multi-class classification are supported. The authors present a multi-class classification problem consisting of three classes: neutral evolution, weak positive selection ($S = 200$) and strong positive selection ($S = 400$).

The training set is built with the use of msms[63] coalescent simulation software. msms includes the functionality of the coalescent simulation software ms[10] with the addition of a model for deme- and time-dependent selection using forward simulations. This is used by ImaGene to generate hard and soft selective sweeps. The exact parameters provided to ImaGene are provided in [24]. With regards to the images representation of the coalescent simulation, populations are arranged along the rows, loci along the columns, and the sampling frequency of each allele in the depth (colour). So each pixel/position within the image contains information about the frequency of each nucleotide (A, C, G, T). Concerning this definition, the colours are presented by their red, green and blue (RGB) value, also a monochrome approach is presented by the authors, to reduce the dimensionality. Lastly also sorting based on rows and columns is presented and supported by ImaGene and shows great potential for increasing the accuracy.

ImaGene directly interacts with Keras[26] and supports defining custom CNN architectures. Various CNN architectures are presented by the authors of ImaGene for different classification and regression problems. The provided CNN architecture is for the multi-class classification problem in Fig.3.1. The binary problem changes only the filter size of the 2D convolutional layer to a constant value of 32 and the number of classes to 1. The authors of ImaGene state accurate predictions, when sorting based on rows is applied. For the execution time, a bottleneck is present in the coalescent simulation software.

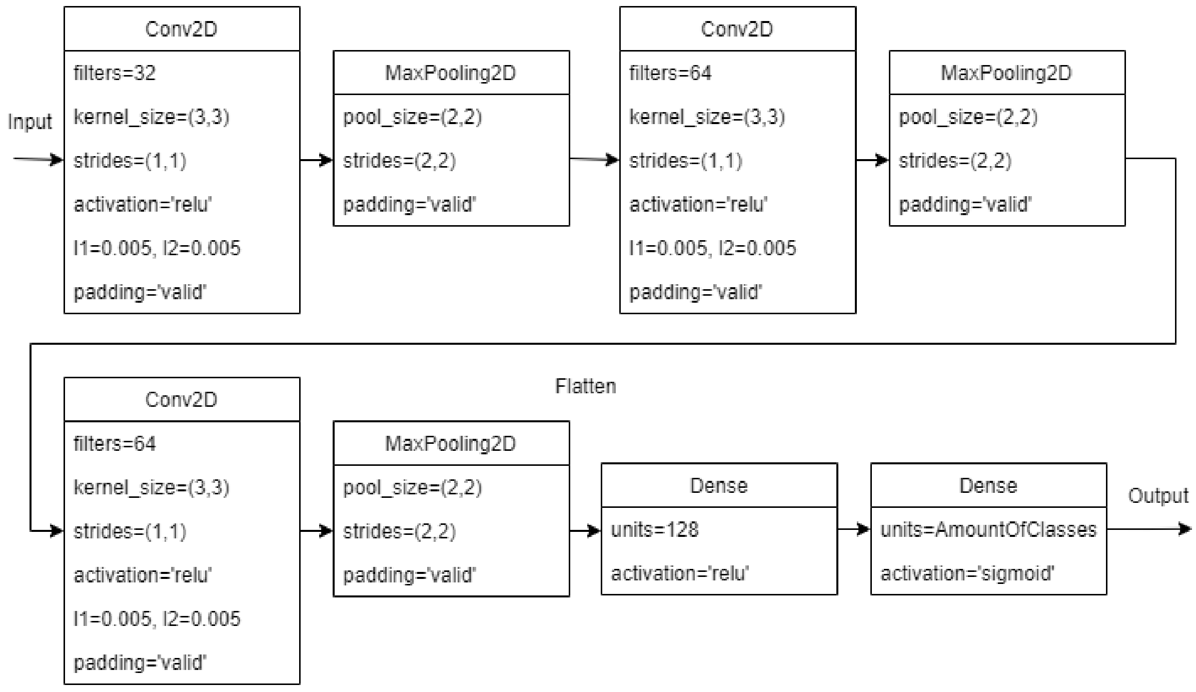


Figure 3.1: ImaGene multi-class classification CNN architecture

### 3.2.2 Flagel et al. sweep detection

The research presented by Flagel et al. in [14], does not present a complete program for the use of sweep detection. But none the less their research shows promising results and can be seen as a valid design. Within this research, CNNs are applied to several evolutionary questions consisting of identifying local introgression, estimating the recombination rate, detecting selective sweeps, and inferring population size changes. The focus with regards to this research is sweep detection, and therefore from this point forwards the research presented by Flagel et al.[14] is regarded by only the sweep detection sections.

The method used by Flagel et al. in [14] for sweep detection is based on the S/HIC method presented by Schrider and Kern[64]. This method performs the casting into a classification task where five different classes are considered: recent (classic) hard sweep, recent soft sweep, region linked to a nearby hard sweep, a region linked to a nearby soft sweep, or a neutral region. A hard sweep is the more classical type of sweep as described in Chapter 2. Summarized a hard sweep is an event when a single haplotype (group of genes inherited from a single parent) with an advantageous allele rises in frequency. A soft sweep is an event where multiple haplotypes with advantageous alleles rise in frequency simultaneously.

The image representation is an alignment image, where each pixel/position is the allele present in a given chromosome at a given site. Meaning that each row represents a chromosome and each column represents a segregating site. Similar to ImaGene also sorting is present, within this paper sorting is done based on rows. Given that the goal of the paper was not to introduce a framework, but the prove the feasibility of CNNs within the research field of population genetics no extensive framework is provided. The method of Flagel et al. in [14] shows an increase in performance concerning S/HIC is for sweep detection. With regards to computation complexity, the training time of the sweep detection is provided for an Nvidia K80 GPU of 6.6 hours by the authors of [14].

Concerning the training data of the CNN, the same coalescent simulation software is used as Schrider[65] in combination with coalescent simulation data from Discoal[66] (ms[10] like program able to generate both hard sweeps and soft sweeps). Next, the JPT population (Japanese individuals from Tokyo) from phase 3 of the 1000 Genomes data-set[67] are used for inference. The network architecture is provided in Fig.3.2. Two different inputs are defined input1 takes an alignment image as input, and input2 takes positions of polymorphisms as input.

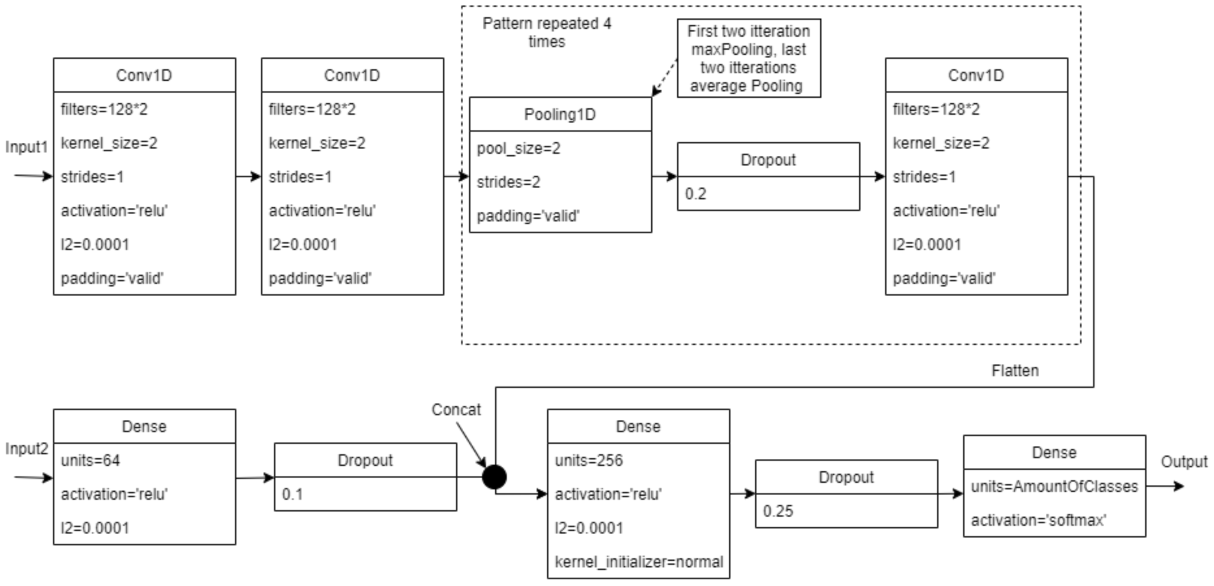


Figure 3.2: Flagel et al. CNN architecture for sweep detection

### 3.2.3 DiploS/HIC

DiploS/HIC[23] is CNN based approach that is from the authors of S/HIC[64]. The development of Diplo/HIC is a direct effect of S/HIC disadvantage of not being able to process unphased haplotypes. Meaning that it should be known on which chromosome the allele is located, Diplo/HIC resolves this problem. Diplo/HIC is based on 12 summary statistics, including for example the $w$ statistic discussed in the section describing OmegaPlus. The final image representation of DiploS/HIC is on each row a summary statistic, each column a sub-window, and each pixel/position presents the resulting value.

Because DiploS/HIC is based on S/HIC it consists of the same classes: recent (classic) hard sweep, recent soft sweep, region linked to a nearby hard sweep, a region linked to a nearby soft sweep, or a neutral region. Both the training and testing data are generated with the coalescent simulation software Discoal[66]. The network architecture is provided in Fig.3.3 and passes the input image to three different branches which are later concatenated.

DiploS/HIC offers a complete approach with the possibility to use custom genomic data-sets by either Discoal or other coalescent simulations if they follow the same format. It is shown by the authors that DiploS/HIC is quite powerful compared to S/HIC. The main downside of DiploS/HIC comes with the fact that summary statistics are still used and therefore including all the complexity and disadvantages that are present in signature-based sweep detection methods and tools.

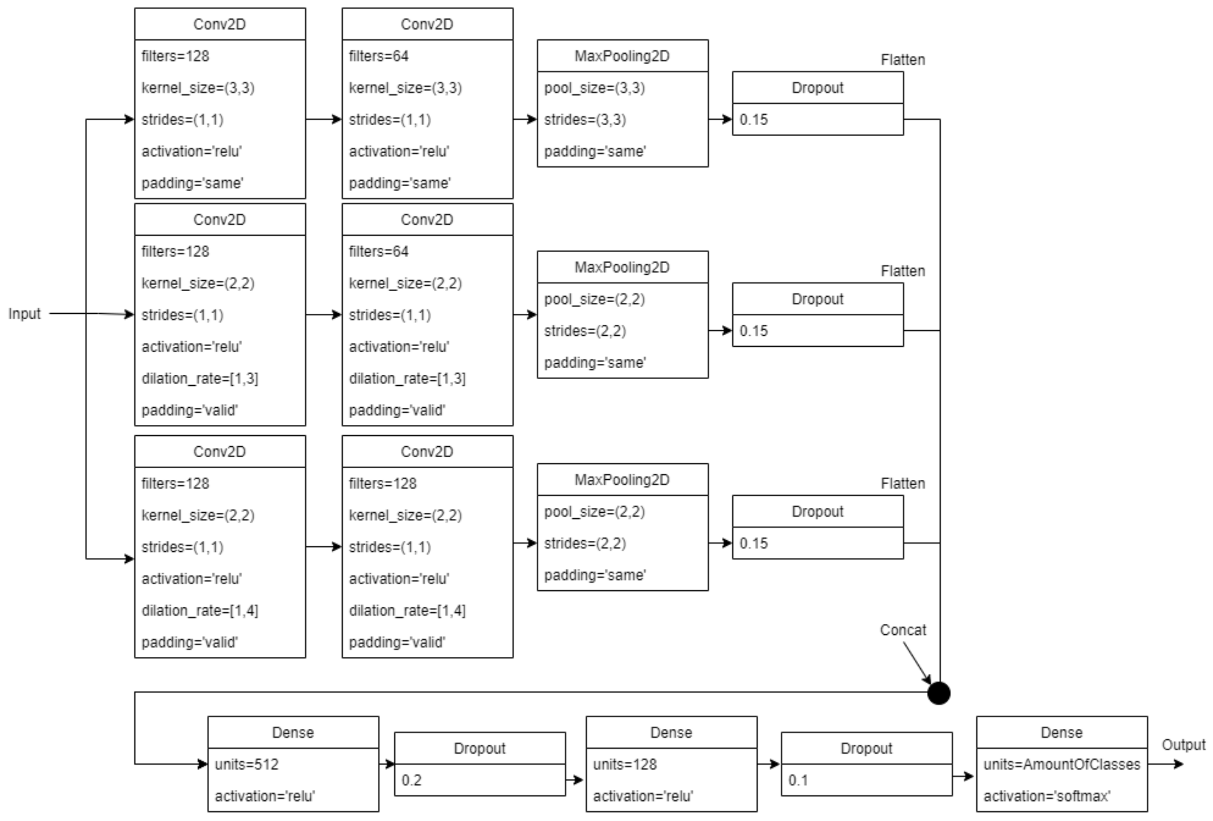


Figure 3.3: DiploS/HIC CNN architecture for sweep detection

## 3.3 Novelties of ASDEC

ASDEC differs compared to the tools and methods discussed in this chapter. ASDEC is compared separately to both the signature-based methods and the CNN based methods.

### 3.3.1 Signature based methods and tooling

The discussed signature-based methods and tools could be less beneficial than just using the raw data directly. By for example only using one signature information could be missed and this could negatively affect the accuracy of the prediction, this problem is partly solved by RAISD[21] but it is still unclear if all information is present in the used signatures of RAISD. Also, false positives could be present as was for example the case with SweepFinder[60] and background selection which left a similar signature as positive selection. A method such as ASDEC is based on the raw data as input and has, therefore, no reliance on signatures which possibly can have great benefits.

### 3.3.2 CNN based methods and tooling

While all CNN based methods and tools provide a way to classify genomic data-sets in a variety of different classes. With regards to sweep detection, they are all, to the best of my knowledge not easily able to process whole-genome data. Meaning that only small examples/data-sets can be provided and the result is simply the probability of that data-set having a certain strength of positive selection per image and not the loci of the sweep.

The processing of the whole-genome data-sets enables researchers to not only determine if a certain region has undergone positive selection. But also pinpoint the loci within complete genomic data-sets that have undergone positive selection. ASDEC is just like the signature-based methods and tools able to perform sweep detection with regards to whole-genome data-sets.

ASDEC does not rely on summary statistics as other methods and tools such as DiploS/HIC, and also provides support for multiple coalescent simulation software programs (ms, mssel, mbs and msHOT). Besides support for coalescent simulation data-sets ASDEC is also able to process real genomic data-sets. ASDEC provides support for easily importing custom CNN network architectures. A disadvantage shared by most CNN based methods and tools is the relatively long execution time, this is due to the vast number of images needing to be processed (can be in the thousands to millions depending on the input). ASDEC alleviates the long execution times by implementing an acceleration possibility on an FPGA or DPU. Acceleration is to my knowledge not yet possible with current CNN based sweep detection methods and tooling.

# 4 ASDEC FRAMEWORK

The ASDEC framework is built upon many independent blocks, by combining these blocks a complete solution for sweep detection is achieved. The blocks present in the ASDEC framework are data generation, data encapsulation, pre-processing, CNN inference, CNN training and post-processing. The relation between them is illustrated in Fig.4.1.

When using the ASDEC framework a user while possible, does normally not interact with all separate blocks but rather interacts with multiple blocks using overarching calls. While all independent blocks are still usable by themselves (with one exception pre-processing and data encapsulation can only be called together by image generation) providing flexibility. Two overarching blocks are available one focused on training an ASDEC CNN based on a given CNN network architecture, and one for calling a trained model for inference. Both overarching blocks include data generation, data pre-processing, and data encapsulation, logging (time and execution parameters), when calling inference also the post-processing is included (also illustrated in Fig.4.1). The overarching calls greatly reduce complexity and provide greater usability. The final result of the ASDEC framework is either a TensorFlow model (.pb format) when performing training or a list of probabilities of selectivity on a certain position within a genomic data-set.

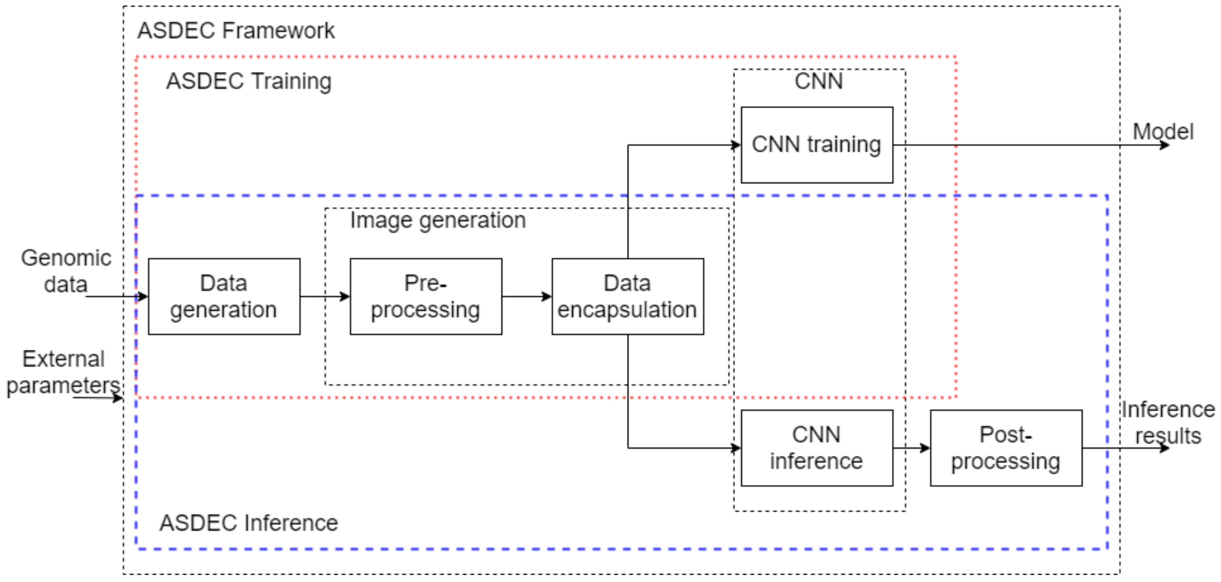


Figure 4.1: Relation between all separate blocks implemented in the ASDEC framework

To elaborate the inner workings of the ASDEC framework an explanation with the help of the independent blocks is provided in the following sections, as illustrated in Fig.4.1. The connection between the blocks is a direct pass-through of the output of a block to the input of another block.

## 4.1 Data generation

ASDEC implements direct data generation to facilitate for example the neural architecture search (NAS). The direct data generation provides an easy way to compare ASDEC with the signature-based methods and tools discussed in chapter 3. Besides an easy comparison, it enables others to recreate the obtained results in this research. The implementation of ASDEC supports both the ms file format, the VCF file format, and the FASTA file format (parsing with RAiSD to VCF required). With regards to this research both simulated genomic data-sets, and the first human chromosome of the Yuruba population (1000Genomes data-set[68]) are used. Genomic data-set samples are generated under a variety of neutral and selective models. The tooling required for the simulated data-set generation consists of ms[10], mssel (kindly provided by R.R. Hudson), msHOT[69] and mbs[70]. Here the ms tool generates genomic data-sets which contain no selective sweep, and the tool mssel generates genomic data-sets which contains a selective sweep on a predefined location in the genome. The coalescent simulation software mbs generates samples of DNA sequences when in both copies of a particular gene a mutation is present on a site (biallelic) targeted by selection[70]. The tool mbs is based upon the tool ms. The coalescent simulation software msHOT is also based on ms and ”allows for implementation of multiple crossover hotspots and/or multiple gene conversion hotspots in the simulated genetic region” as stated by Hellenthal & Stephens[69]. All the supported data-sets are present in Table.4.1 and this table is a subset of the data-sets previously used by Alachiotis & Pavlidis [21] for evaluation and comparison. To enable the processing of VCF files (another format such as the ms file format) that are for example describing the first human chromosome of the Yuruba population a separate parsing tool (based on RAiSD) is integrated into ASDEC.

Table 4.1: Data-sets supported by ASDEC out of the box

| Dataset | Sweep type | Confounding factor | Simulation software | Varying parameters | Range of values |
|---|---|---|---|---|---|
| 1-60 | Hard, complete | bottleneck | Hudson’s ms and mssel | Severity (−eN) | 0.005–0.5 |
| | | | | Duration (−eN) | 80–400 ($4N_0$ generations) |
| | | | | Beginning (−eN) | 800–20,000 ($4N_0$ generations) |
| 61-70 | Hard, complete | Migration | Hudson’s ms and mssel | Population join (−ej) | 0.003–3 ($4N_0$ generations) |
| 71–91 | No sweep | Recombination | msHOT | Hotspot region size (−v) | 5–10 kb |
| | | | | Hotspot intensity (−v) | 2-100 |
| 92–101 | Hard, complete | Recombination | msHOT and mbs | Hotspot region size (−v) | 5 kb |
| | | | | Hotspot intensity (−v) | 2-20 |

In the following sections, the different confounding factors are elaborated.

### 4.1.1 Bottleneck

A population bottleneck is a fast shrinkage in the population size, this effect can be caused by various events such as environmental disasters[6]. The effect of a population bottleneck is the reduction of variation in the gene pool and therefore a (possible) reduction in the genetic diversity. ASDEC includes 60 bottleneck simulations, each simulation comprises both a neutral data-set and a data-set with a selective sweep at the centre of the simulated region. The bottleneck simulations use the same parameters as used in [21]. The severity of each bottleneck is ranging from 0.5 to 0.005. The severity is determined by the relative population size during a bottleneck compared to the present-day population size. The bottleneck duration varied between 80 and 400 generations, and the beginning of the bottleneck lays between the 800 and 20,000 generations backwards in time. Lastly, the conversion from coalescent time units to generations includes the assumption that the present-day population size consists of 50,000 haploid genomes.

#### 4.1.2 Migration

Genetic migration is the transfer of alleles (genetic variants) from one sub-population to another sub-population [6]. For the migration models, 10 data-sets are presented as taken from [21]. The migration models implement a continent-island model, within this model the population size of the island is 20 times smaller than the population size of the continent, where the continent acts as a population without any information (ghost population). The migration rate is set to 3 ($M_{ic}$) and $t_{merge}$ denote the time that the island and continent population merged into a common population. The value of $t_{merge}$ (in Table.4.1 denoted as population join) varies from very recent ($0.003$) to very old ($3$) and is expressed in the usual coalescent timescale ($4N_c$ where $N_c$ is the effective population size of the continent)[21]. Both the population mutation rate ($\theta$) and recombination rate ($\rho$) are fixed to 2000 for the whole genomic region[21].

#### 4.1.3 Recombination

Genetic recombination refers to both the shuffling of non-homologous (chromosomes that do not belong to the same pair) chromosomes and the breaking and rejoining of homologous chromosomes [6]. The recombination events are divided into two sets, where the first set includes recombination intensity relative to the rest of the genome and ranging from 2 to 100, and the hotspot region sizes of 5kb, 10kb, and 3 times 5kb [21]. The second set includes recombination hotspots that include a selective sweep, with a recombination intensity ranging from 2 to 20, and a hotspot region size of 5kb [21].

### 4.2 Image generation

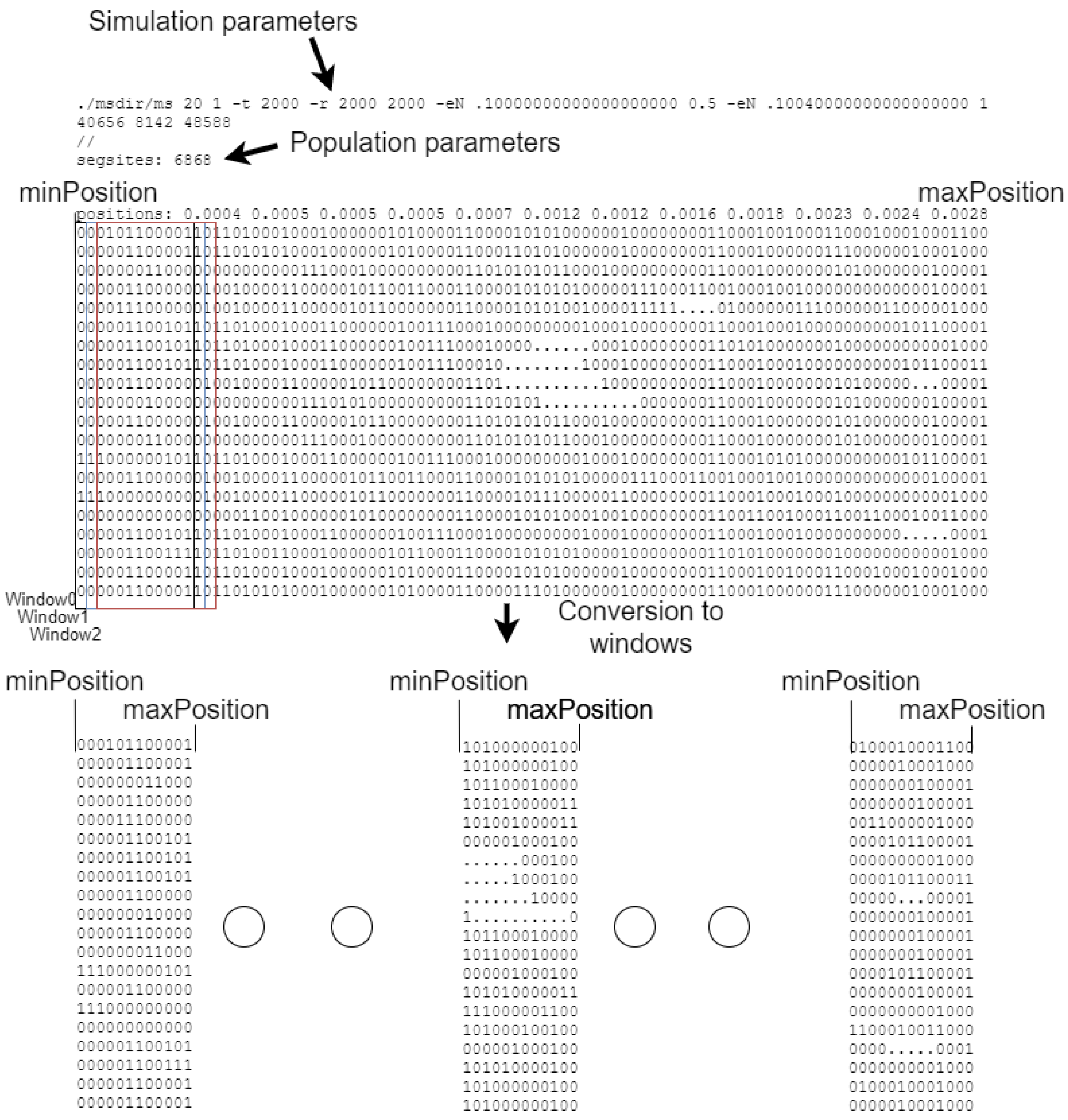
The generation of the image after the generation of data is divided into two separate blocks on the one hand the data needs to be pre-processed and afterwards encapsulated inside an image.

#### 4.2.1 Pre-processing

Processing a complete genomic data-set in a single image does not result in a clear indication of the possible loci of a selective sweep. The reason for this is that each image leads to a single set of probabilities regarding selective sweep presence in that complete image. This means when the whole genomic data-set is offered as a single image the result is not able to determine the locus/loci of selection in the genomic data-set. This problem is tackled in the pre-processing, here the genomic data is first divided into smaller subsets and afterwards, multiple images are saved all containing a different subset of the complete genomic data-set.

The division into sub-regions is performed with the help of a sliding window approach. This approach is based on SNPs, it enables a user to specify a window size in SNPs and a step size between windows in SNPs. This results in a large set of images per genomic data-set. By presenting each image individually to the CNN it is possible to predict the location of a selective sweep.

In more detail first, the obtained simulated genomic data-sets (represented in Fig.4.2 top half) is loaded and a variety of different parameters of the data-set are recovered, dubbed simulation parameters in Fig.4.2. The simulation parameters relate to the parameters used during the coalescent simulation or for real data are provided. The simulation parameters are used to retrieve the number of populations (number of data-sets) and individuals within each population (amount of rows in a single data-set/population). The number of populations determines the number of times to execute the complete process.



```
./msdir/ms 20 1 -t 2000 -r 2000 2000 -eN .1000000000000000000 0.5 -eN .1004000000000000000 1
40656 8142 48588
//
segsites: 6868
positions: 0.0004 0.0005 0.0005 0.0005 0.0007 0.0012 0.0012 0.0016 0.0018 0.0023 0.0024 0.0028
000101100001101101000100010000001010000110000101010000000100000000110001001000110001000100011 00
00000110000110110101010001000000101000011000110101000000010000000011000100000011100000001000100 0
000000011000000000000000111000100000000000110101010110001000000000001100010000000101000000000100001
000001100000010010000110000010110011000110000101010100000111000110010001001000000000000000100001
00001110000001001000011000001011000000001100001010100100001111 1....0100000011100000011000001000
000001100101101101000100011000000100111000100000000010001000000000110001000100000000000000101100001
000001100101101101000100011000000100111000010000......00010000000001101010000000100000000000001000
000001100101101101000100011000000100111000010........100010000000001100010001000000000000101100011
000001100000010010000110000010110000000001101..........1000000000011000100000000101000000...00001
000000010000000000000000111010100000000000011010101..........00000000110001000000000101000000000100001
000001100000010010000110000010110000000001101010110001000000000001100010000000101000000000100001
000000011000000000000000111000100000000000110101010110001000000000001100010000000101000000000100001
111000000101101101000100011000000100111000100000000010001000000000110001010100000000000101100001
000001100000010010000110000010110011000110000101010100000111000110010001001000000000000000100001
111000000000010010000110000010110000000001100001011100000110000000011000100010001000000000001000
000000000000000001100100000010100000000110000101010001001000000000110011001000110011000100011000
000001100101101101000100011000000100111000100000000010001000000000110001000100000000000.....0001
000001100111101101001100010000001011000110000101010100001000000000110101000000001000000000001000
000001100001101101000100010000001010000110000101010000000100000000110001001000110001000100010001000
000001100001101101010100010000001010000110000111010000000100000000110001000000111000000001000100 0
```

```
000101100001    101000000100    010001000110
000001100001    101000000100    000001000100 0
000000011000    101100010000    000000010000 1
000001100000    101010000011    000000010000 1
000011100000    101001000011    001100000100 0
000001100101    000001000100    000010110000 1
000001100101    ......000100    000000000100 0
000001100101    .....1000100    000010110001 1
000001100000    .......10000    00000...0000 1
000000010000    1..........0    000000010000 1
000001100000    101100010000    000000010000 1
000000011000    101100010000    000000010000 1
111000000101    000001000100    000010110000 1
000001100000    101010000011    000000010000 1
111000000000    111000001100    000000000100 0
000000000000    101000100100    110001001100 0
000001100101    000001000100    0000.....000 1
000001100111    101010000100    000000000100 0
000001100001    101000000100    010001000100 0
000001100001    101000000100    000001000100 0
```

Figure 4.2: Genomic data in the ms format, A derived allele is presented by ’1’, An ancestral allele is presented by ’0’, and a blank is represented by a ’.’

The processing of a single population starts with the localization of the beginning of a population, within the file denoted by ’//’ (also shown in Fig.4.2). With the start of the population determined various arbitrary parameter checks are performed, and next the number of segsites are extracted from the file (number of columns in a single data-set/population) in Fig.4.2 denoted by population parameters. Next, the position vector is extracted (**p**), this vector relates a position in the genome with the position of an SNP (SNPs are not uniformly spaced). To support modern biological applications multiplication with a constant value for all values of (**p**) is supported. From the position vector (**p**) two special positions are denoted, the $minPosition$ stating the first SNP position of a window and $maxPosition$ denoting the last SNP position of a window. Before applying the sliding window the $minPosition$ and $maxPosition$ are set to the first SNP position and the last SNP position of the whole population(also shown in Fig.4.2). In the line after the position vector the genomic data is presented, this data consists of a matrix ($A$) containing: '1', '0' and '.', here the '1' represents a derived allele, '0' represents an ancestral allele and a '.' represents a blank spot.

Within the pre-processing two further options are provided, Genomic extraction where a subset of $A$ is created based on given parameters, and a sliding-window that creates multiple subsets of $A$ using a sliding-window approach. Both these options are further elaborated in the following two sub-sections.

### Genomic extraction

With both the genomic matrix $A$ and the position vector **p** a genomic extraction option is provided to the user. The extraction works by extracting a given amount of SNPs ($extractionRange$) of the genomic matrix $A$ on both the left and right of a given position ($extractionPosition$). The given position for extraction ($extractionPosition$) is related to SNP index by finding the closest value in the position vector **p** to $extractionPosition$, and the index of the closest value in the vector **p** is the SNP index (extractionIndex). Genomic extraction is (mostly) used for training to extract only the sub-genomic region that has a selective sweep present and discards the remaining genomic data. The algorithm is presented in Algorithm 1.

**Algorithm 1** Genomic extraction

```
 1: function Genomic extraction($A$, p, $extractionRange$, $extractionPosition$)
 2:     extractionIndex = GetClosestPositionIndex(p, $extractionPosition$)
 3:     minIndex = extractionIndex - $extractionRange$        ▷ left most SNP index
 4:     maxIndex = extractionIndex + $extractionRange$        ▷ right most SNP index
 5:     if not IndexWithinRange(minIndex, maxIndex) then
 6:         ERROR
 7:     end if
 8:     $A$ = $A$[:, minIndex:maxIndex]
 9:     p = p[minIndex:maxIndex]
10:     $minPosition$ = p[0]
11:     $maxPosition$ = p[len(p)-1]
12:     return $A$, p, $minPosition$, $maxPosition$
13: end function
```

### Sliding-window

The sliding-window implementation in ASDEC converts the whole-genomic data-set (defined in the matrix $A$) to multiple sub-genomic regions for inference. The sliding-window requires both a $windowSize$ and $stepSize$ (distance between two windows), both expressed in a number of SNPs. Each sub-genomic region is presented by its own $minPosition$, $maxPosition$ and matrix

($\bar{A} \subset A$). The sliding window approach is also presented in Fig.4.2 (lower half) and Algorithm 2.

**Algorithm 2** Sliding-window approach

```
 1: function Sliding-window($A$, p, $windowSize$, $stepSize$)
 2:     if not SufficientSizeForWindow($windowSize$, $A$) then
 3:         ERROR
 4:     end if
 5:     for i in range($0$, AmountOfColumns($A$)-$windowSize + 1$, $stepSize$) do
 6:         $\bar{A}$ = A[:,i:i+$windowSize$]
 7:         $minPosition$ = p$[i]$
 8:         $maxPosition$ = p$[i + windowSize - 1]$
 9:         SaveImage($\bar{A}$, $minPosition$, $maxPosition$)
10:     end for
11: end function
```

### 4.2.2 Data encapsulation

To encapsulate the Genomic data $\bar{A}$ (if extraction mode is not used $\bar{A} = A$) with the information regarding the $minPosition$, and $maxPosition$ both the image itself and the filename are required. Starting with the creation of the image firstly all the $'.'$ are parsed to a value of $'2'$, this creates an $\bar{A}_{parsed}$ containing only the values $'0','1','2'$. Then $\bar{A}_{parsed} = \bar{A}_{parsed} \odot 127$, where $\odot$ is an element-wise multiplication with all values in $\bar{A}_{parsed}$. The final result is a grayscale image as shown in Fig.4.3. Still, $\bar{A}_{parsed}$ does not contain any position information to relate the SNP locations with positions in the genome, this is encoded in the filename with the help of $minPosition$ and $maxPosition$. The position of each SNP within the image is not preserved, because after the image is processed the probability is based on a complete image and not on a single SNP within the image. An image filename example is provided in Fig.4.4.

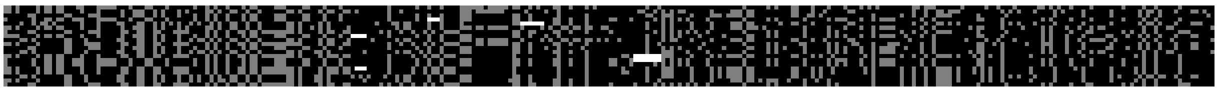


Figure 4.3: Genomic data encapsulated inside a monochrome image, grey pixels denote derived alleles, black pixels denote ancestral alleles, white pixels denote blank spots. Each column represents a SNP.

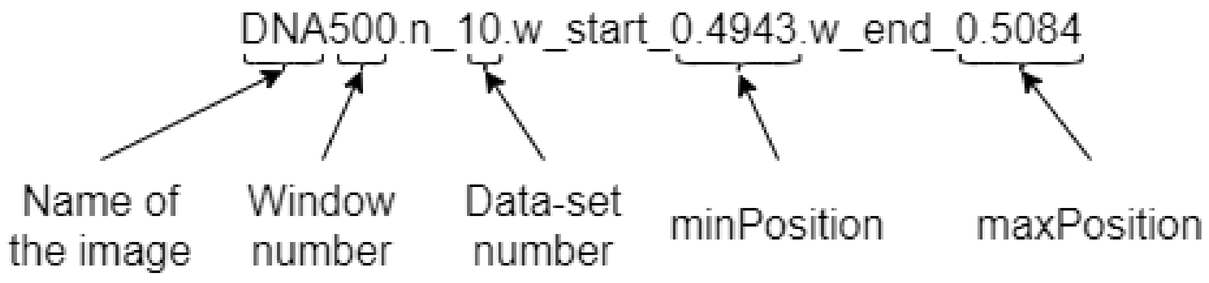


Figure 4.4: Example image filename with elaboration on information encoding

## 4.3 Convolutional Neural Network

Parsed and encapsulated genomic data can be provided to a CNN for either training or inference. Both training and inference are defined as black boxes and are elaborated with the help of their inputs and outputs. For both the training and inference all the values are re-scaled to a range between 0 and 1.

### 4.3.1 Training

When training a CNN with ASDEC a couple of parameters need to be defined, these parameters include the model name, image directory, batch size, epochs, image height, image width, and model design. An important parameter is the model design, this is a separate file that can be included by a user of ASDEC to use a custom CNN model design using Tensorflow [16] and Keras [26]. Also, the image directory should include two different folders (neutral and selection), where each folder should contain image's coupled to the folder name (label). Lastly, both the image height and image width should correspond to the size of the images present in the image directory. When training is finished the final model is saved in a .pb format and log files are included regarding the accuracy and loss of each epoch during training of both the training and validation data. Next to the standard logging present by ASDEC also TensorBoard[71] (extension of TensorFlow) is included to provide extra logging in the form of accuracy and loss of both the training and validation data overall epochs, and more importantly profiling of the CNN architecture. In Fig.4.5 an illustration containing all the various inputs and outputs are provided.

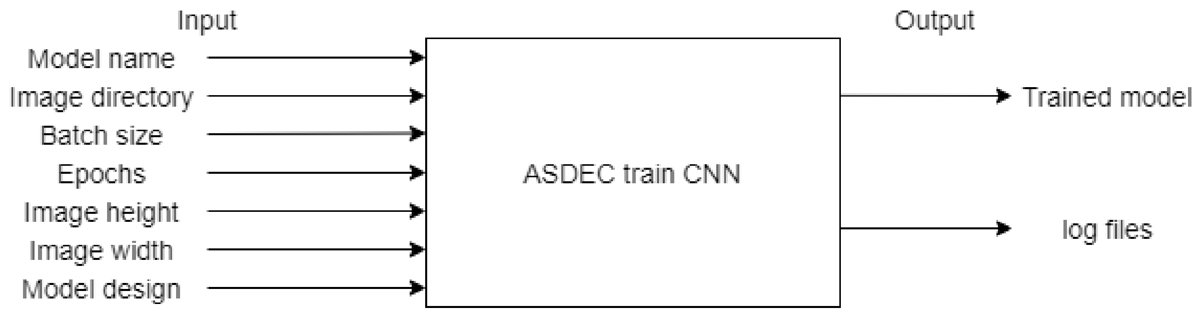


Figure 4.5: Black box showing IO of training ASDEC

### 4.3.2 Inference

Inference with ASDEC only requires three parameters: the model, directory with images (created in the image generation step), and a directory for storing the results. The final output is a file containing for each image the $minPosition$(left most SNP position), $maxPosition$(rightmost SNP position), average position of the $minPosition$ & $maxPosition$ ($avgPosition$), probability of neutrality, and probability of selection.

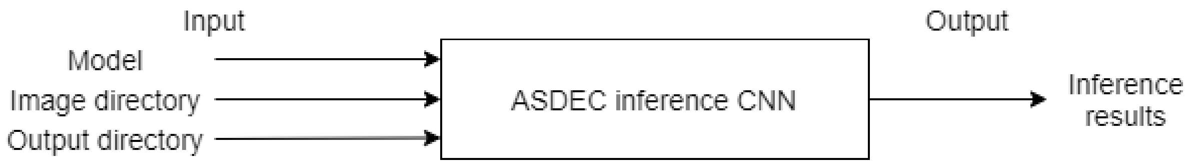


Figure 4.6: Black box showing IO of inference ASDEC

## 4.4 Post-processing

The output of the inference performed by the CNN has a high level of granularity to provide sensible arguments for the obtained results a post-processing step is provided, but not required. The reduction in granularity also reduces the number of outliers by different methods elaborated

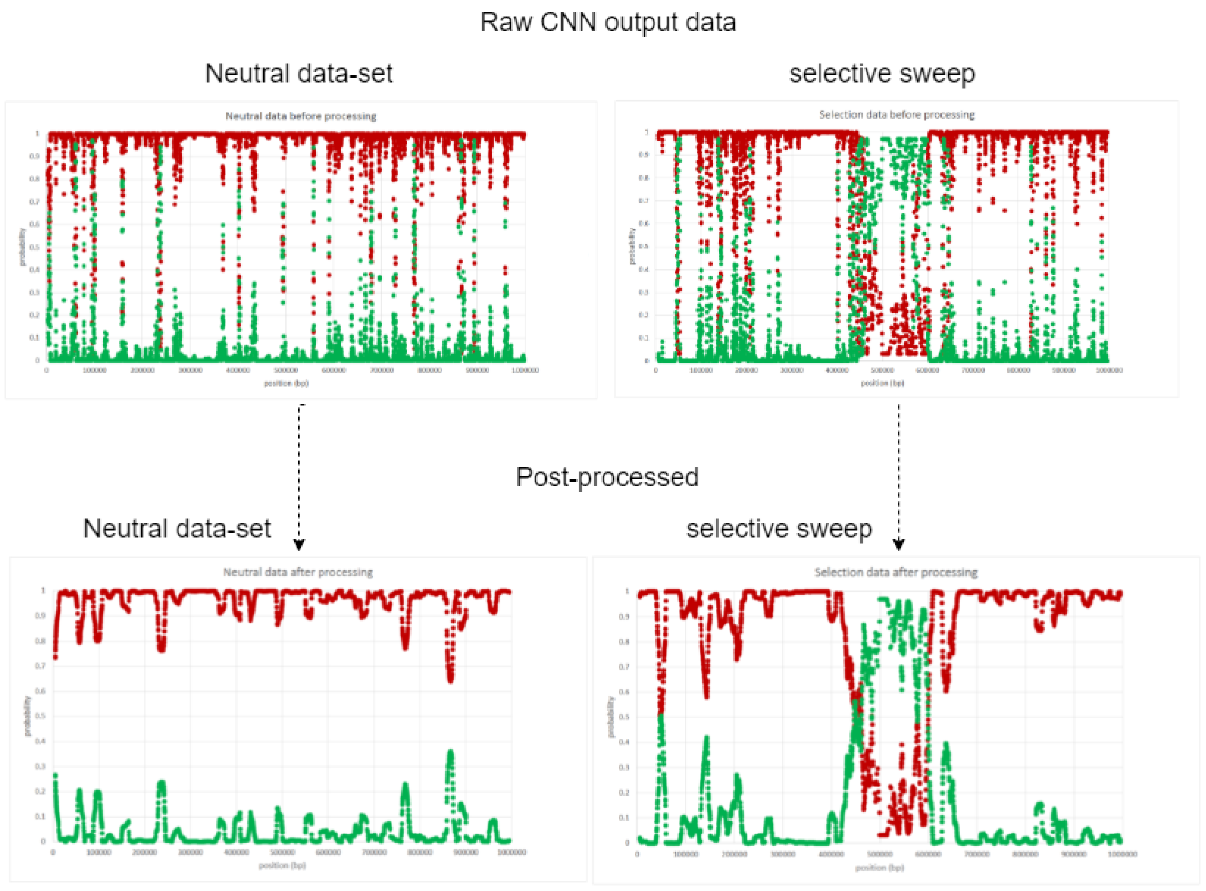


Figure 4.7: Effect of post-processing in green the probability of selectivity and in red the probability of neutrality, with mode Grid mode normal, a = 4000, b=10000

in more detail in this section. The ASDEC framework includes four different post-processing modes, all these post-processing modes apply some sort of averaging on the CNN raw output data after inference. Each post-processing mode also results in a new file where each line no longer represents a single image but multiple images, when averaging is applied to $minPosition$ and $maxPosition$ are set accordingly based on the left-most and right-most image. Next to the normal information present in the file such as the $minPosition$, $maxPosition$, $avgPosition$, probability of neutrality, and probability of selection another column is added which gives the number of images used in the post-processing of that entry. Post-processing is an integral part of the ASDEC framework, and the effects can be observed in Fig.4.7.

### 4.4.1 Window based on SNPs

This type of post-processing applies another sliding-window based approach (equal to the pre-processing) over the raw output of the CNN. The input for this type of post-processing is a $windowSize$ in combination with a $stepSize$, both expressed in SNPs. This post-processing creates a new file where the $minPosition$ is equal to the first $minPosition$ SNP of an image within the window and the $maxPosition$ is equal to the $maxPosition$ of the last images SNP in the window. The new $avgPosition$ can be calculated with the new $minPosition$ and $maxPosition$. The new probabilities are the average of all probabilities of all images within the window, lastly, the number of images used for averaging is added to the last column in the file. This is illustrated in Fig.4.8.

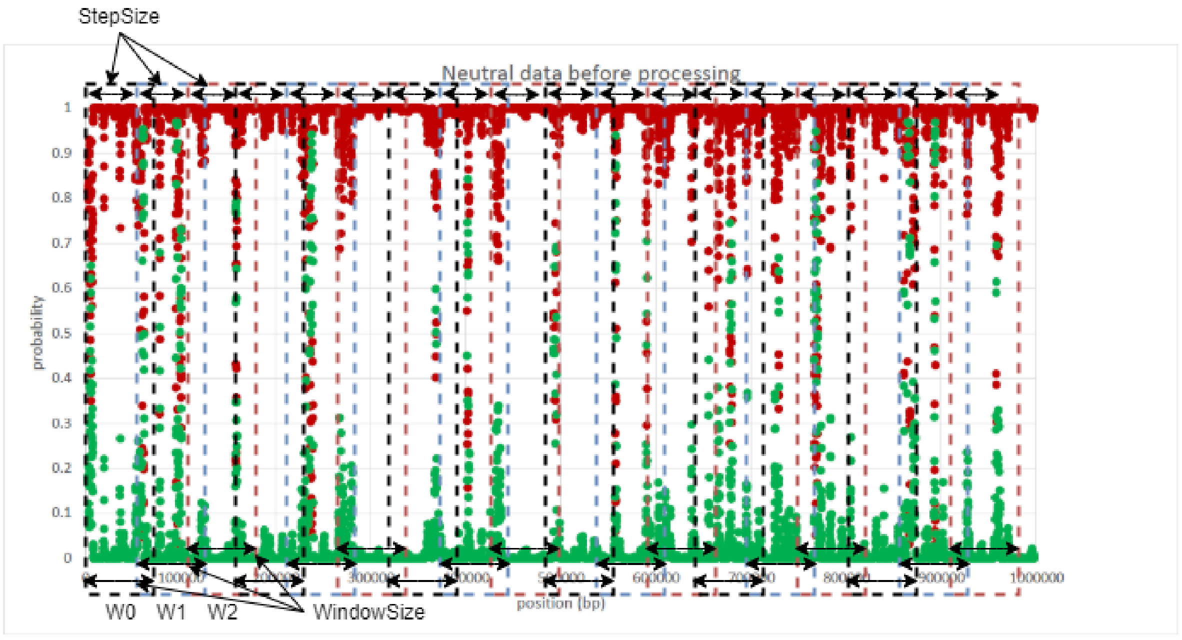


Figure 4.8: Window-based on SNPs/Position, in green the probability of selectivity and in red the probability of neutrality, the dotted lines represent a window ($W$). A window is defined by both the $windowSize$ and $stepSize$ from the previous window, both can be defined either in positions or SNPs. The first window always starts on the first position or SNP depending on the mode.

### 4.4.2 Window based on position

Concerning this type of post-processing, a sliding-window based approach over the raw CNN data is applied. When calling this type of post-processing both a $windowSize$ and $stepSize$ should be inputted, both are expressed in positions of base pairs (bp). The process starts by getting the closest SNP to the given position of the first image, and afterwards, the $minPosition$ is taken from the first image. Afterwards, the whole process is run accumulating all probabilities, until the process runs out of the given $windowSize$. Now the process is stopped and the $maxPosition$ of the last image within the window is taken. All averages are calculated, and a line is added providing the number of images used for averaging to the new result file, then a given $stepSize$ (in bp) is taken and the process is repeated. This is done until no longer one whole window fits in the remainder of the raw CNN data. This is illustrated in Fig.4.8.

### 4.4.3 Grid mode normal

The grid size approach for post-processing deploys a vastly different strategy than the approaches discussed until now. The grid size approach requires a grid size and a max distance range in positions of base pairs. This type of post-processing works by that the amount of grid size given is uniformly disturbed over the complete genome, and on every single position (grid point) a certain range is taken for both sides in positions (max distance range) this is further illustrated in Fig.4.9. All points that fall within such a grid points range are averaged and the corresponding $maxPosition$ and $minPosition$ are taken. When the amount the user inputted grid size is larger than the amount of data entries/images in the raw CNN output data, the amount of entries/images in raw CNN output data is enforced as grid size. This again leads to a new file consisting of all the needed information including the number of old entries within a new entry (after post-processing).

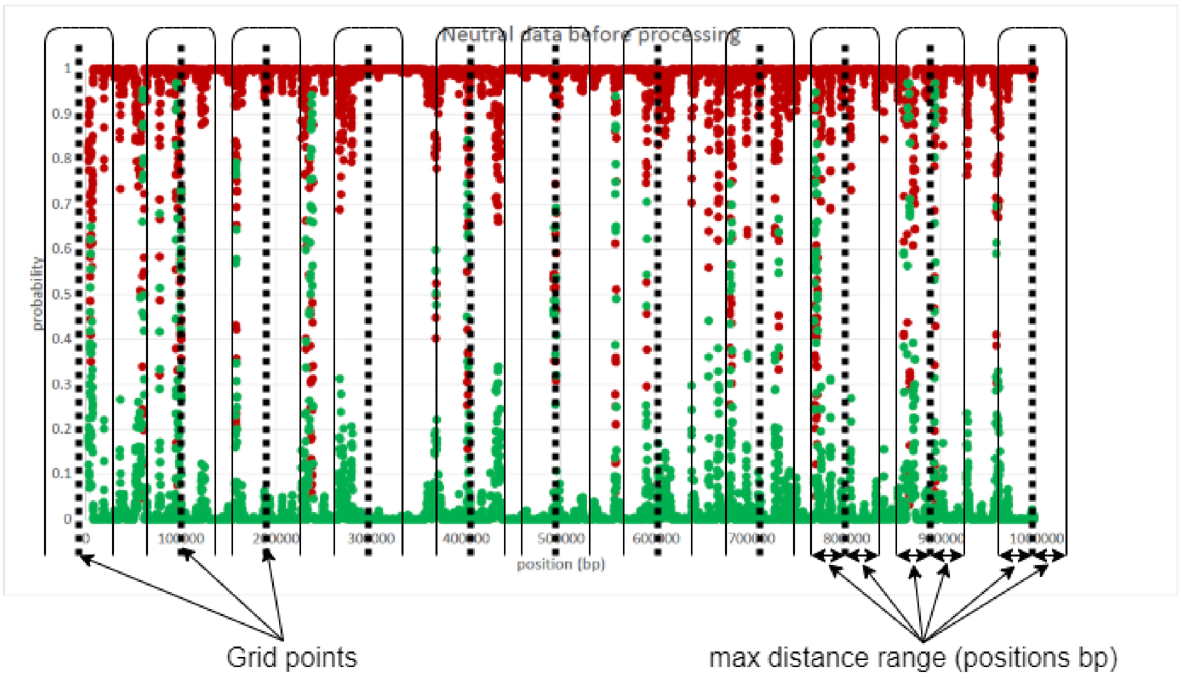


Figure 4.9: grid mode, in green the probability of selectivity and in red the probability of neutrality, the dotted lines represent grid points, the lines represent the area taken in average, and the double arrow indicates the max distance range in positions of base pairs

### 4.4.4 Grid mode always enforced

The grid mode always enforced performs exactly the same process as the grid mode normal discussed before with one exception. When the given grid size by the user is larger than the amount of data entries/images in the raw CNN output data, the grid size is still enforced instead of enforcing the amount of data entries/images as in grid mode normal. This will lead to duplicate points inside the post-processed data, but the advantage is that the amount of data entries in the final post-processing output file can be fully controlled.

# 5 NEURAL ARCHITECTURE SEARCH

A hand-designed neural architecture search (NAS) was performed to explore CNN architectures that can be deployed for sweep detection. The full NAS was performed with the ASDEC framework discussed in Chapter 4. As discussed in Chapter 4 ASDEC supports the input of custom CNN designs, and this feature was extensively used during the NAS. The NAS finally presents a final CNN design dubbed SweepNet that represents the overall top performer within the NAS.

The NAS initial CNN architecture draws its inspiration from ImaGene presented in Chapter 3, CNN architecture illustrated in Fig.3.1. To ensure compatibility with Vitis AI in later stages only layers and activation functions are used that are supported by the quantization of a TensorFlow 2 model by Vitis AI. All supported layers by Vitis AI version 1.3 are stated in table 5.1.

## 5.1 NAS setup

The setup of the NAS is defined by the input data-sets, the used metrics for evaluation, and finally, the explored ASDEC attributes. The naming of the different CNN architectures follows the following convention $ModelDesignC\alpha F\beta\gamma L\phi S\rho$. In the name $\alpha$ refers to the number of combined layers (combination of 2D-convolution layers and max-pooling layers), $\beta$ refers to the filter size deployed in each layer (can be 4816 defined as starting from 48 to 16 over the combined layers). $\gamma$ can either be E: equal, D: decreasing, or I: increasing depending on the value $\beta$ overall combined layers ($\alpha$), $\phi$ refers to the number of dense layers present in the CNN, and $\rho$ refers to the size of the dense layers (equal overall dense layers).

### 5.1.1 Deployed data-sets

The data-sets used for the NAS are implemented within the ASDEC framework as stated in table 4.1 in Chapter 4. With regards to the NAS only data-sets 1 to 60 (bottlenecks) were deployed, where inference was performed with both data-set 1 (easiest scenario) and data-set 60 (hardest scenario) for all steps in the NAS. The training data-sets for steps 1, 2, 3, and 4 were data-sets 1 to 10, data-sets 51 to 60, data-set 1, and data-set 60, meaning that each CNN architecture consisted of 4 trained models with each 2 inference runs (8 inferences per CNN architecture in total). Training for steps 5 and 6 was performed with all data-sets (1 to 60). Within each step, all models were trained with an equal number of training images.

### 5.1.2 Metrics

The evaluation metrics used for comparing the various CNN architectures are equal to the evaluation metrics presented by Alachiotis and Pavlidis[21]. The evaluation metrics provide a way to evaluate whole-genome data-sets. The metrics are directly implemented into the ASDEC framework as standalone tooling to make it easy to assess the performance of the framework. The evaluation metrics are a) the detection accuracy, b) the success rate, and c) sensitivity

Table 5.1: Vitis AI-supported Operations and APIs, as stated in the manual of Vitis AI[3]

| **Supported Operations and APIs for quantization of TensorFlow 2** |
|---|
| tf.keras.layers.Conv2D |
| tf.keras.layers.Conv2DTranspose |
| tf.keras.layers.DepthwiseConv2D |
| tf.keras.layers.Dense |
| tf.keras.layers.AveragePooling2D |
| tf.keras.layers.MaxPooling2D |
| tf.keras.layers.GlobalAveragePooling |
| tf.keras.layers.UpSampling2D |
| tf.keras.layers.BatchNormalization |
| tf.keras.layers.Concatenate |
| tf.keras.layers.Zeropadding2D |
| tf.keras.layers.Flatten |
| tf.keras.layers.Reshape |
| tf.keras.layers.ReLU |
| tf.keras.layers.Activation |
| tf.keras.layers.Add |

(True Positive Rate (TPR)).

The detection accuracy is expressed by the distance error ($Dist$) (measured in base pairs). The reported position by ASDEC is the position with the highest probability in the population ($i$) and is denoted by ($s_i$), and the distance is defined between the reported position ($s_i$) and the real selection target at position $X$. The distance error is calculated for multiple populations ($N$), and its inverse to the value of the detection accuracy, meaning a lower distance error means a higher detection accuracy. The formula for calculating the distance error is provided in Equation.5.1.

$$Dist = \frac{\sum_{i=1}^{N} |s_i - X|}{N} \tag{5.1}$$

The success rate ($suc$) gives the number of populations ($N$) that fall within a range ($\epsilon$ for the NAS is 1% in bp in both directions) of the real selection target position ($X$). were when the detected selection position ($s_i$) is within the range the return is 1, and otherwise is 0 (Iverson bracket notation) expressed in Equation.5.2.

$$Suc = \frac{\sum_{i=1}^{N} [|s_i - X| < e]}{N} \tag{5.2}$$

The true positive rate ($TPR$) uses the probabilities of selection of the neutral populations ($p_i$), so give $N$ neutral populations, and a probability threshold ($thr$) defined by an user-defined false positive rate ($FPR$) a $TPR$ can be determined. The equation for calculating the $TPR$ is provided in Equation.5.3.

$$TPR = \frac{\sum_{i=1}^{N} [p_i > thr]}{N} \tag{5.3}$$

Besides the mathematical presentation of the various evaluation metricises a graphical example is provided in Fig.5.1.

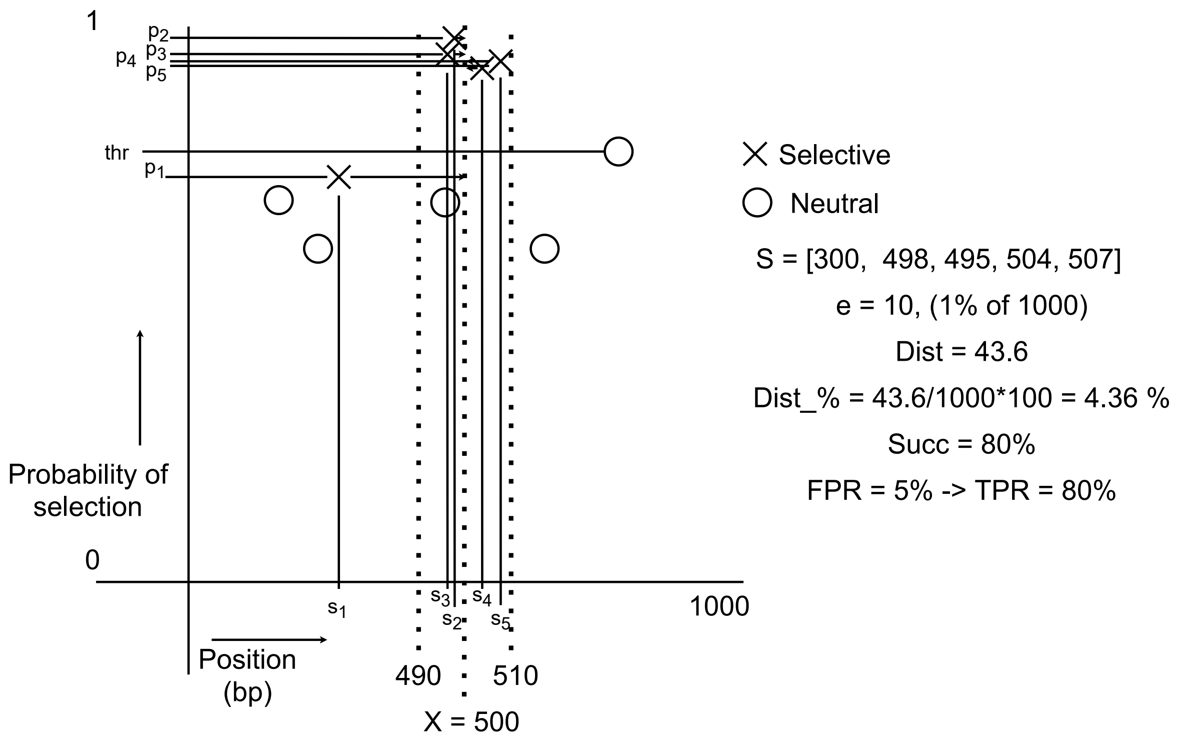


Figure 5.1: Example of all presented metrics, for a population size of 5 ($N = 5$)

To provide an easy comparison between the various CNN architectures all different CNN architectures are described by a single score. This single score is based on the various evaluation metricises (Equation.5.4). Equation.5.4 describes the final score $S^i_m$, were $i$ is the performed test, and $m$ is the model under evaluation. In the equation $w$ is the multiplication factor of each score, and considering the NAS the multiplication are defined as $w_{Dist} = \frac{1}{6}, w_{suc} = \frac{2}{6}, w_{TPR} = \frac{1}{2}$. Next, the comparison between the model under evaluations obtained scores and the best performer of the signature-based methods and tools, for example, $Dist^i_{min}$ is made with the best scores obtained by either RaisD, OmegaPlus, SweeD or SweepFinder2, while $Dist^i_m$ refers to the ASDEC model under evaluation.

$$S^i_m = w_{Dist} \times \frac{Dist^i_{min}}{Dist^i_m} + w_{suc} \times \frac{Suc^i_m}{Suc^i_{max}} + w_{TPR} \times \frac{TPR^i_m}{TPR^i_{max}} \tag{5.4}$$

### 5.1.3 Explored ASDEC and CNN attributes

The NAS consists of two different stages, where steps 1 till, and including 3 include the exploration of different CNN designs (CNN layers and CNN layer sizes). Secondly, steps 4 till, and including step 6 include different inputs and input sizes of both ASDEC and the CNN within ASDEC. In Fig.5.2 the complete NAS is illustrated, and in table 5.2 in combination with table 5.3 more detail is provided for the individual steps. Concerning the first 3 steps of the NAS, after each step multiple, CNN architectures are selected based on an individual assessment and taken to the next step. Steps 4 and 5 get all the CNN architectures selected from steps 1,2, and 3, lastly step 6 gets all the CNN architectures selected from steps 4 and 5 (as illustrated in Fig.5.2). Training of all models is performed with an evenly distributed number of images for both classes.

In more detail step 1 explores the number of combined layers in combination with the 2D-convolutional constant filter size. Step 2 explores an increasing and decreasing 2D-convolutional

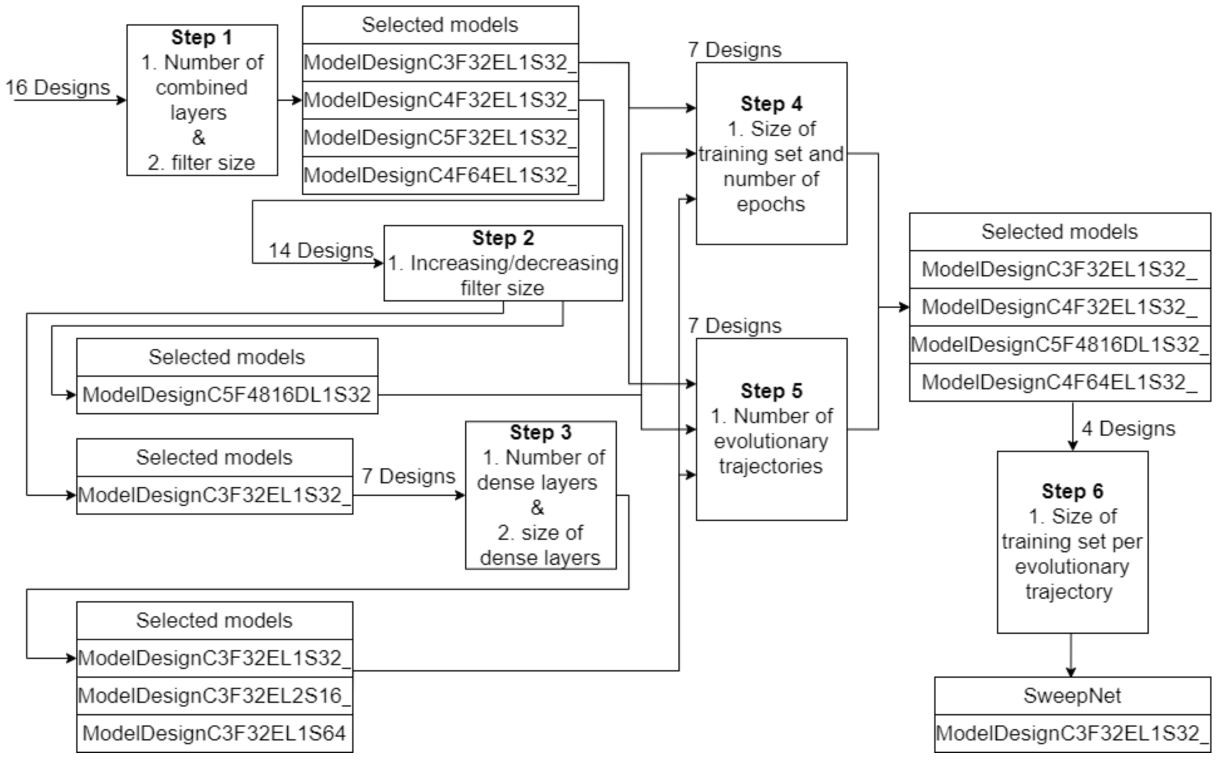


Figure 5.2: NAS for the SweepNet model exploration with ASDEC

filter size (with a factor of 2 or addition/subtraction with a constant value of 8 between combined layers). Step 3 explores the number of dense layers in combination with a constant size of the dense layers (see table 5.2). Step 4 doubles the number of training images together with doubling the number of training epochs. Step 5 uses a broader training data-set instead of the more narrow training sets used in the earlier steps. Step 6 deploys a vast amount of training images of the broader training data-set (see table 5.3).

## 5.2 NAS results

Within this section, the results of each step as illustrated in Fig.5.2 are discussed per step. The results of each step are both considered concerning the results of the easy bottleneck scenario (data-set 1) and the hard bottleneck scenario (data-set 60). The CNN architectures picked are selected by their ability to outperform or equal the performance of signature-based methods and tools. While the score provides a well defined first indication a further observation is used to select CNN architectures based on regions where for example other CNN architectures are lacking.

For each step, all possible combinations of training data and testing data are included. Both model misspecification (non-matching training and testing data-sets) and correctly specified trained models (matching training and testing data-sets) are considered. The score based on Equation.5.4 is calculated for all 8 cases and the summation of 2 final scores one for testing with data-set 1 and one for testing with data-set 60 are compared. This leads to two final scores one solely for testing with data-set 1 and one solely for testing with data-set 60, which both include model misspecification and the correct model specification.

Table 5.2: NAS of SweepNet with values and ranges performed with ASDEC consisting of the first 3 steps

| Step | Number of combined layers (Conv2D and MaxPooling) | Conv2D filter size | Filter Size between Conv2D layers | Number of dense layers | Size of all Dense layer |
|---|---|---|---|---|---|
| 1 | [2,3,4,5] | [8,16,32,64] | Equal | 1 | 32 |
| 2 | 3 | [32, (8-16-32), (32-16-8)] | [Equal, Increasing, Decreasing] | 1 | 32 |
| 2 | 4 | [64, (16-24-32-40), (40-32-24-16), (128-64-32-16), (16-32-64-128)] | [Equal, Increasing, Decreasing] | 1 | 32 |
| 2 | 5 | [32, (48-40-32-24-16), (8-16-32-64-128), (128-64-32-16-8), (16-24-32-40-48)] | [Equal, Increasing, Decreasing] | 1 | 32 |
| 3 | 3 | 32 | Equal | [1,2] | [16,32,64] |

Table 5.3: NAS consisting of training, and testing information for all NAS steps

| Step | Epochs | Number of images | Training data-sets | Testing data-sets |
|---|---|---|---|---|
| 1, 2, 3 | 3 | 210.000 | [1, 60, 51-60, 1-10] | [1, 60] |
| 4 | 6 | 420.000 | [1, 60, 51-60, 1-10] | [1, 60] |
| 5 | 6 | 420.000 | 1-60 | [1, 60] |
| 6 | 6 | 2.500.000 | 1-60 | [1, 60] |

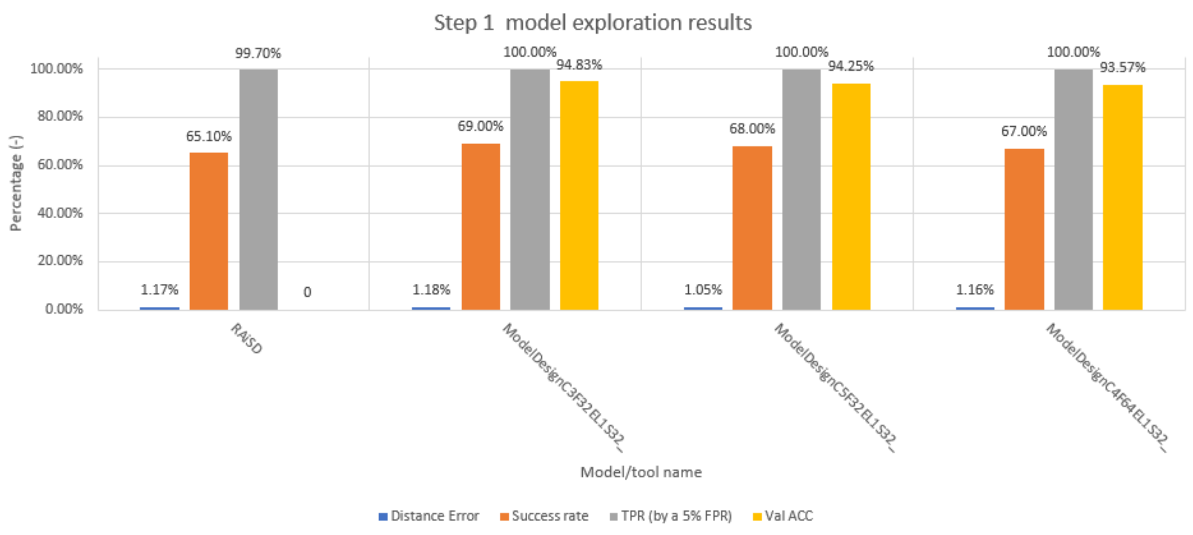


Figure 5.3: NAS trained with data-set 51-60 and inference on data-set 1 versus top performer signature-based tooling

### 5.2.1 Step 1

In step 1 correctly specified trained models for testing with data-set 60 are outperforming (score > 1, lowest score 3.267846027 of ModelDesignC2F8EL1S32_ for training with data-set 60) the current highest signature-based score. The only exception is one trained (data-set 51-60) model (ModelDesignC5F64EL1S32_) which was unable to be trained (under-fitting), with both validation accuracy and training accuracy around 50%. Because all CNN architectures performed well concerning testing with data-set 60, testing with data-set 1 was used to identify the initial selection of CNN architectures to use in the next step of the NAS. Both Fig.5.3 and Fig.5.4 were used, were again the top performers were selected ModelDesignC5F32EL1S32_ and ModelDesignC4F64EL1S32_. This leads to the CNN architectures in table 5.4 being taken to the next step of the NAS. In table 5.4 all scores of the trained models are summed together (with model misspecification 4 scores summed, and without model misspecification 2 scores summed), therefore these scores can not be compared with the signature-based top-performing methods and tools.

Table 5.4: NAS selected top performers step 1

| | With model misspecification | | without model misspecification | |
|---|---|---|---|---|
| CNN architecture name | data-set 1 | data-set 60 | data-set 1 | data-set 60 |
| ModelDesignC3F32EL1S32_ | 2.839228198 | 6.994511736 | 1.767661136 | 5.418889531 |
| ModelDesignC4F32EL1S32_ | 2.763063559 | 8.132298953 | 1.768040753 | 5.931920118 |
| ModelDesignC5F32EL1S32_ | 2.624992724 | 7.745378593 | 1.567285399 | 6.955005094 |
| ModelDesignC4F64EL1S32_ | 2.635501731 | 6.072803885 | 1.595990812 | 5.605119981 |

### 5.2.2 Step 2

Step 2 of the NAS starts with the selection of the CNN architectures provided from step 1 of the NAS. All the selected CNN architectures, get an adjusted CNN architecture where the filter size of the 2D-convolutional layers over the layers is multiplied with a factor of 2 in both increasing and decreasing directions. For CNN architectures with more than 3 combined layers, an addition/subtraction with a value of 8 between each combined layer to simulate a narrower range of

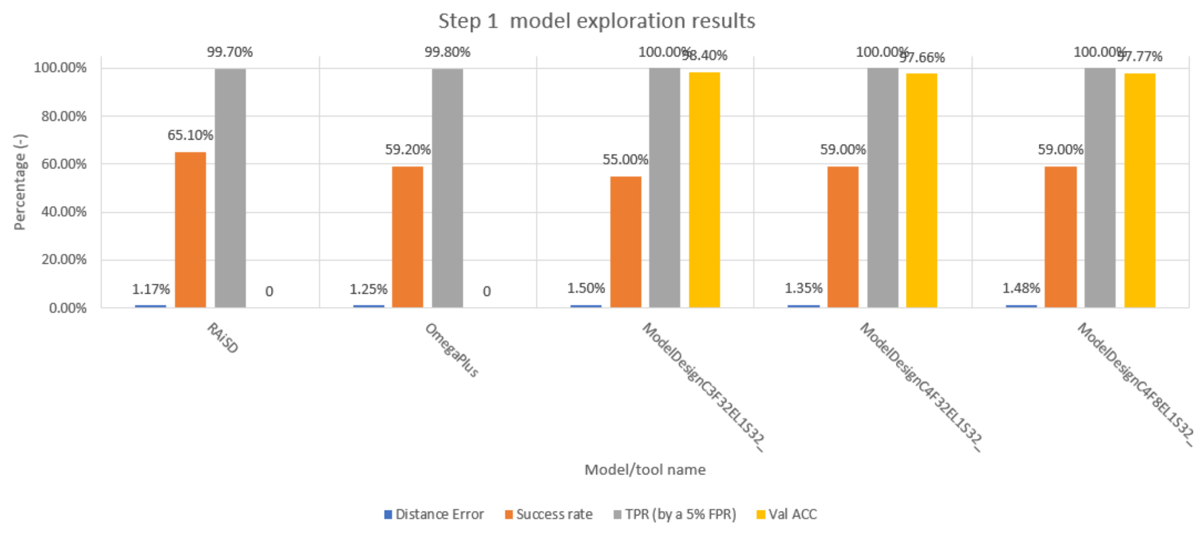


Figure 5.4: NAS trained with data-set 01-10 and inference on data-set 1 versus top performer signature-based tooling

the filter size is provided. This is not done for the CNN architecture with 3 or fewer combined layers, because the range was already considered to be quite narrow.

Regarding step 2 no major improvements were observed compared to testing with data-set 1 with both correct model specification and model misspecification. This led to the conclusion to take the highest scoring CNN architecture from data-set 1 from step 1 to step 3, ModelDesignC3F32EL1S32_. Only one CNN architecture was taken to step 3, to reduce the time required for the step. To still introduce a CNN architecture with a variable filter size between combined layers ModelDesignC5F4816DL1S32_ was included in steps 4 and step 5.

### 5.2.3 Step 3

Step 3 of the NAS starts with the CNN architecture ModelDesignC3F32EL1S32_ and adjusts the CNN architecture by changing both the number and size of the dense layers (see table 5.2). The CNN architecture ModelDesignC3F32EL1S64_ outperforms ModelDesignC3F32EL1S32_ with regards to the score of both testing with data-set 1 and data-set 60 when no model misspecification is included. When model misspecification is introduced ModelDesignC3F32EL1S64_ achieves a higher score concerning data-set 60. ModelDesignC3F32EL1S64_ is the best performer in step 3 and therefore selected. ModelDesignC3F32EL2S16_ is also selected for it the second-best performance for testing with data-set 60, only being outperformed by ModelDesignC3F32EL1S64_. The selected models with their scores are included in table 5.5.

Table 5.5: NAS selected top performers step 3

| | With model misspecification | | without model misspecification | |
|---|---|---|---|---|
| CNN architecture name | data-set 1 | data-set 60 | data-set 1 | data-set 60 |
| ModelDesignC3F32EL1S32_ | 2.839228198 | 6.994511736 | 1.767661136 | 5.418889531 |
| ModelDesignC3F32EL1S64_ | 2.757579202 | 8.384038711 | 1.781945128 | 7.451762528 |
| ModelDesignC3F32EL2S16_ | 2.664883728 | 7.603784454 | 1.641056034 | 6.676355522 |

### 5.2.4 Step 4

Step 4 of the NAS doubled both the number of training images and the number of training epochs (see table 5.3). No significant performance increase was observed for ModelDesignC3F32EL1S32_ which still showed sufficient performance for testing with data-set 1 and top performance (outperforming signature-based tooling) for testing with data-set 60. ModelDesignC4F64EL1S32_ stands out in step 4, because of the really good performance for testing with data-set 1 under model misspecification and average performance for testing with data-set 60 (see table 5.6). Both ModelDesignC4F64EL1S32_ and ModelDesignC3F32EL1S32_ are for these reasons taken towards step 6 (see Fig.5.2).

Table 5.6: NAS selected top performers step 4

| | With model misspecification | | without model misspecification | |
|---|---|---|---|---|
| CNN architecture name | data-set 1 | data-set 60 | data-set 1 | data-set 60 |
| ModelDesignC3F32EL1S32_ | 2.818561467 | 6.915314581 | 1.775626395 | 5.272820092 |
| ModelDesignC4F64EL1S32_ | 3.295490546 | 5.809527147 | 1.635601187 | 4.185958158 |

### 5.2.5 Step 5

Step 5 of the NAS performs training with all bottleneck data-sets as shown in table 5.3. where the total number of images used for training is equal to the number of images used for training in step 4. In step 5 of the NAS, model misspecification is no longer present because all models are trained with all the data-sets. The two top performers concerning testing with data-set 1 were selected (see table 5.7). Testing with data-set 60 was considered less important because all trained models achieved scores higher than 1 (no longer summation is presented as was the case in steps 1, 2, and 3), meaning that they outperform signature-based methods and tools.

Table 5.7: NAS selected top performers step 5

| CNN architecture name | data-set 1 | data-set 60 |
|---|---|---|
| ModelDesignC4F32EL1S32_ | 1.019286828 | 1.49107955 |
| ModelDesignC5F4816DL1S32_ | 0.984521145 | 1.217286052 |

### 5.2.6 Step 6

Step 6 of the NAS performs training with all bottleneck data-sets as done in step 5, but now approximately the same number of images per trajectory as in step 4 are used. The use of the same number of images per trajectory led to a significant increase in the total number of training images up to $2.5 \times 10^6$, which was equally distributed over the two classes. While a drop in performance is observed for step 5, with this large number of training images a clear conclusion can be drawn from step 6. In step 6 of the NAS, the final SweepNet architecture is based on the input CNN architectures provided from both step 4 and step 5 of the NAS. The decision for ModelDesignC3F32EL1S32_ was motivated by having the highest score for both testings with data-set 1 and testing with data-set 60 as shown in table 5.8.

Table 5.8: NAS selected top performers step 6

| CNN architecture name | data-set 1 | data-set 60 |
|---|---|---|
| ModelDesignC3F32EL1S32_ | 0.953962911 | 1.19765561 |
| ModelDesignC4F32EL1S32_ | 0.908503305 | 1.194135433 |
| ModelDesignC4F64EL1S32_ | 0.817546643 | 0.594967188 |
| ModelDesignC5F4816DL1S32_ | 0.918062245 | 0.728434178 |

# 6 ASDEC HARDWARE IMPLEMENTATIONS AND PERFORMANCE

The focus until now was to introduce the concepts that drive ASDEC and the deployed CNN architecture natively used by ASDEC, dubbed SweepNet. This chapter discusses the use of more conventional hardware such as the CPU, and GPU, and the use of acceleration using Deep-Learning Processor Unit (DPU) architectures implemented by the use of Vitis AI. While this chapter focuses mostly on the setup and argumentation, Chapter 7 discusses all the results and a comparison between all the implementations.

## 6.1 Hardware support ASDEC

One of the drawbacks of the ASDEC framework is the performance concerning execution time compared with signature-based tooling. Provided an example for the execution time on the CPU, where the assumption is made that the scaling of the populations and threads is linear, the assumptions are only made for this example justifying the use-case for acceleration.

ASDEC running on 25 threads has, for sweep-detection on data-set 1, with an execution time for 1 neutral population of 416.89 seconds, and, 1 selective population of 354.87 seconds, while RAiSD, a signature-based tool, performs the same task in 9.91 seconds and 6.89 seconds, respectively. RAiSD is considered one of the signature-based tools with the lowest execution time, whereas SweepFinder2 has respective execution times of 760.18 seconds and 1332.89 seconds for a neutral and selective population, respectively. All execution times are normalized to 1 thread to make different thread counts comparable. ASDEC is in its current CPU implementation able to achieve comparable execution times concerning slower signature-based tooling (SweepFinder2 and SweeD).

ASDEC provides support for the following hardware platforms: CPU with multi-core support, and GPU. A large part of this support is provided by TensorFlow 2 out of the box, for example, the support for the GPU for both training and inference, while the multi-core CPU support is for a large part provided by ASDEC, and relies partly on the Tensorflow 2 CPU multi-threading. ASDEC uses this out of the box support and extends it by also multi-threading pre-and post-processing.

### 6.1.1 GPU support

When using TensorFlow 2 automatic checks for supported GPUs is done. If the system that ASDEC is deployed on has a supported GPU in combination with correctly installed drivers as instructed in [16], ASDEC provides TensorFlow 2 with the instructions to use the desired hardware (user can add parameters to the command-line to specify which hardware platform should be used). Significant performance increases when using more parallel optimized hardware such as a GPU is possible.

GPU support is only available for CNN training and CNN inference, whereas the pre-processing (image generation), data-generation, and post-processing still rely on the CPU to perform the instructed tasks.

### 6.1.2 Multi-core CPU support

Besides using ASDEC on more parallel optimized hardware, affords to reduce the execution time when using a CPU are made in the form of multi-threading. The CPU multi-threading is focused on inference because training is under normal circumstances only performed once. Meaning that a speedup for inference yields greater usability for ASDEC. Initial speedups were made with the introduction of multi-threading for ASDEC inference and training. The number of threads spawned ($T$) is controlled by the user. When training a model ASDEC relies partly on the multi-threading support provided by TensorFlow 2. In Fig.6.1 the implementation of training with ASDEC is given, and in Fig.6.2 the implementation of inference with ASDEC is provided. Within these figures, the solid lines represent control lines and show the flow, and the dotted lines show data dependencies between operations or blocks of threads illustrated by an enclosing dotted box. The blue dotted lines present in both figures show the possibility to start a specified number of threads given by the user, and the large arrows illustrate stages that can be omitted by a user. Lastly upwards arrows illustrate parts that can be executed multiple times to accommodate for example processing of multiple files.

#### Training

The multi-core implementation of ASDEC training (Fig.6.1) starts with the generation of data-sets. When no data generation is required this section is skipped until the second synch illustrated by the arrow from initializing to the first synch of the master thread to the second synch of the master thread. When data-generation is required it first of all starts by accessing a script consisting of all supported commands (see Chapter 4 table 4.1). After accessing the supported commands the commands are run for the neutral files when for example 10 data-sets are required and 5 threads are given each thread runs twice illustrated by the upward arrow between the two synchs. When for example 3 data-sets are required and 5 threads are given only 3 threads are used. After the generation of the neutral files is completed the same is done for the selective files.

After data-generation is finished or ASDEC is called with a custom input (requiring no data generation), all other threads are closed and a synch is performed. Then all files are collected for image generation, the master thread is now (mostly) passive. Each thread during image generation is responsible for processing a complete file to images, when more threads than files are provided the number of threads is down scaled to the number of available files. When more files than threads are available the process is repeated for the given number of threads illustrated by the upward arrow in Fig.6.1. After the image generation is completed the training is performed the threads settings given by the user ($T$) are provided to TensorFlow 2, where $floor(T/2)$ is provided for processes within an independent operation, and $ceil(T/2)$ are deployed for use between independent operations.

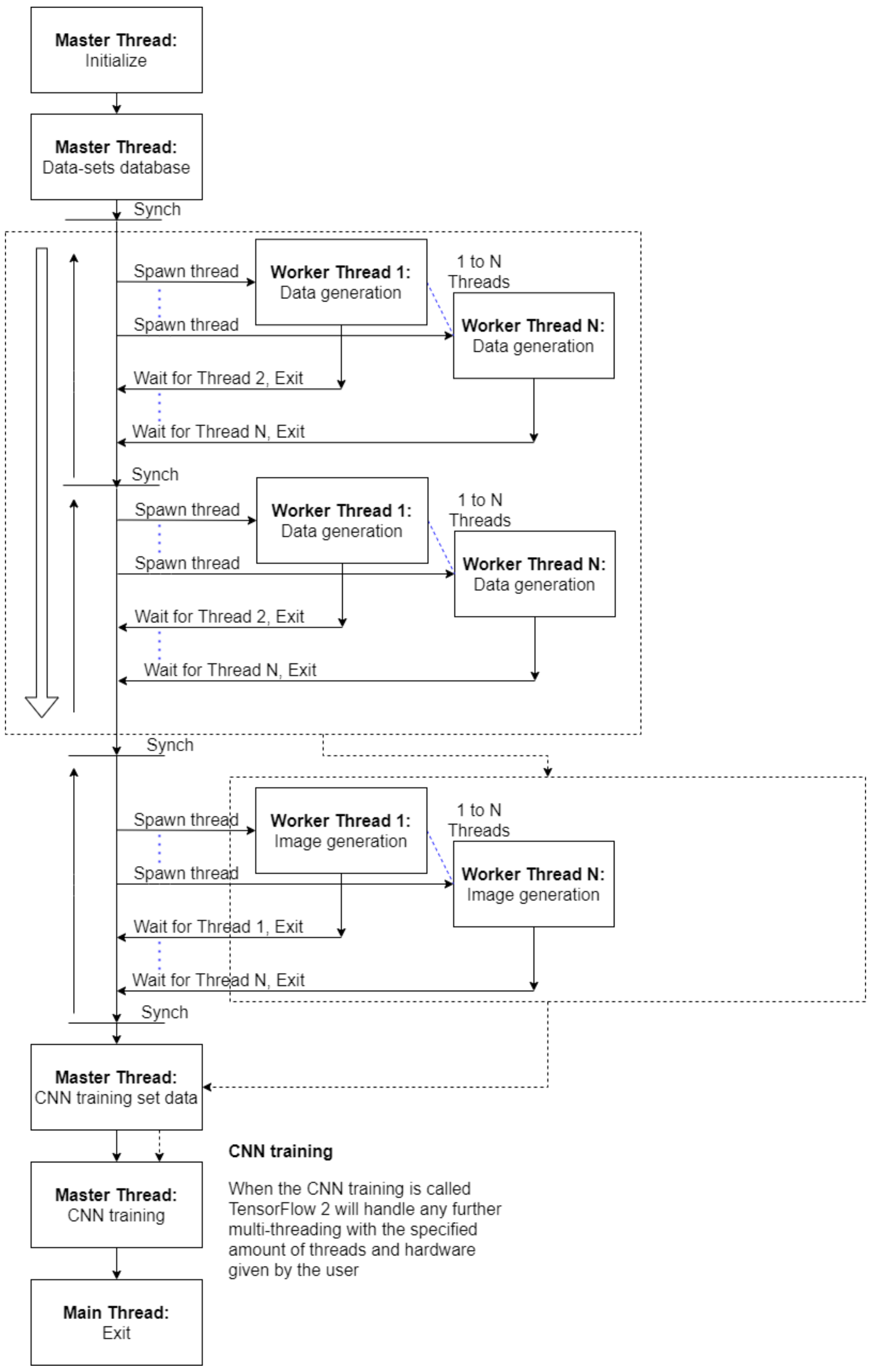

Master Thread: Initialize
Master Thread: Data-sets database
Synch
Spawn thread
Worker Thread 1: Data generation
1 to N Threads
Spawn thread
Worker Thread N: Data generation
Wait for Thread 2, Exit
Wait for Thread N, Exit
Synch
Spawn thread
Worker Thread 1: Data generation
1 to N Threads
Spawn thread
Worker Thread N: Data generation
Wait for Thread 2, Exit
Wait for Thread N, Exit
Synch
Spawn thread
Worker Thread 1: Image generation
1 to N Threads
Spawn thread
Worker Thread N: Image generation
Wait for Thread 1, Exit
Wait for Thread N, Exit
Synch
Master Thread: CNN training set data
Master Thread: CNN training
CNN training
When the CNN training is called TensorFlow 2 will handle any further multi-threading with the specified amount of threads and hardware given by the user
Main Thread: Exit

Figure 6.1: Multi-threading diagram of CNN training with ASDEC. Solid lines represent control, and dotted lines represent data-dependencies between threads, and blocks of threads (represent in blocks in dotted lines). Blue dotted lines present the possibility to start a specified number of threads given by the user. When the main thread starts with the CNN training the multi-threading provided by TensorFlow 2 is used (not elaborated in this figure)

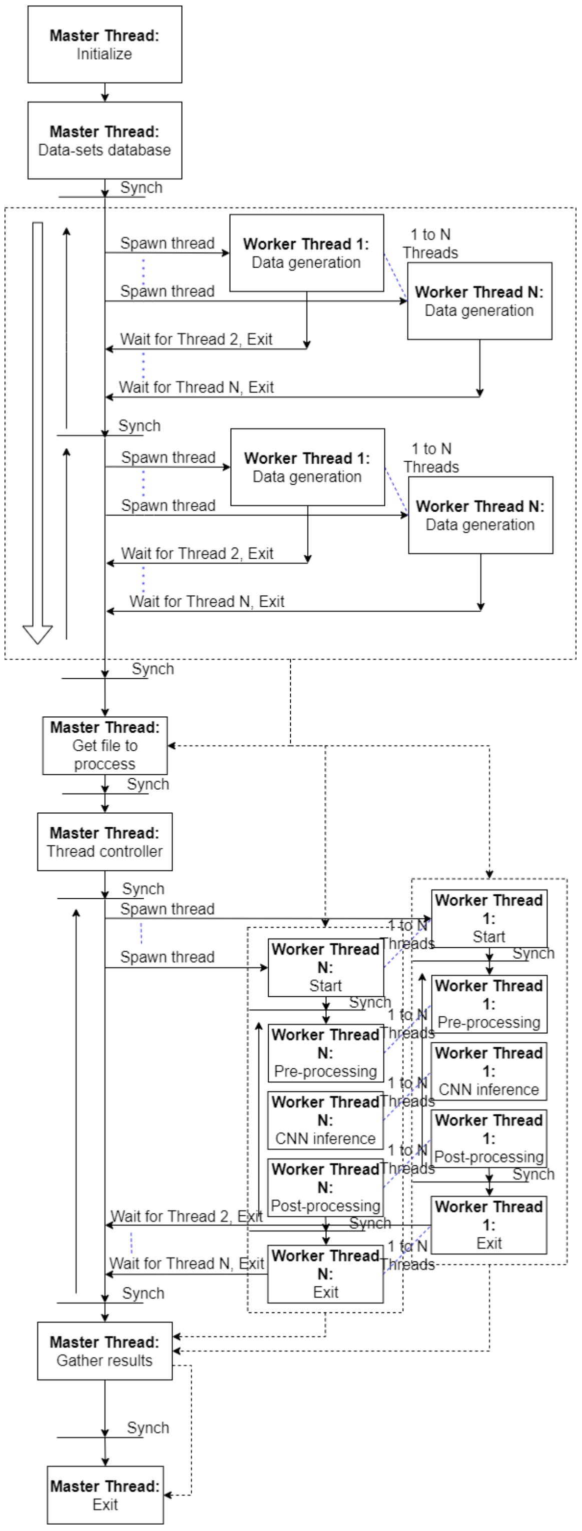
**Inference**

The multi-core implementation of ASDEC inference (Fig.6.2) start equal to the multi-core implementation of ASDEC training by performing data generation if required. After the data generation, a file to process is taken or provided, and a user-specified number of threads ($T$) are started. Each thread is responsible for processing a given number ($\alpha$) of populations (referred to as a population batch) within the file. For example, when a single file with 100 populations is provided for inference, 10 threads are given, and a population batch size of 5 ($\alpha = 5$). Then worker thread 1 should process population 1 to 11, worker thread 2 should process population 11 to 21, worker thread 3 should process population 21 to 31, and so on. Now per pre-processing, inference, and post-processing a population batch size of 5 was given so each worker thread runs the complete process twice. When a thread finishes a batch it saves the results in a personal thread summary file, and all personal thread summary files are afterwards collected by the master thread and written to the final results file, this is done to avoid data races.

**Algorithm 3** Thread termination

```
1:  function Thread-termination($T$, $P$, $\alpha$)
2:      if $T > P$ then
3:          $T = P$
4:      end if
5:      $startPop = 1$
6:      $step = floor(P/T)$
7:      $extraPop = P - step * T$
8:      for i in range($1$, $T$) do
9:          $endPop = startPop + step - 1$
10:         if $extraPop > 0$ then
11:             $endPop = endPop + 1$
12:             $extraPop = extraPop - 1$
13:         end if
14:         if $endPop - startPop > \alpha$ then
15:             $\alpha = endPop - startPop$
16:         end if
17:         $StartThread(startPop, endPop, \alpha)$
18:         $startPop = endPop + 1$
19:     end for
20: end function
```

When the number of threads ($T$) is provided all populations to process ($P$) are divided between all the threads. When $modulo(P, T)$ is not equal to $0$ it is considered not possible to evenly distribute all populations ($P$) overall threads ($T$). To alleviate this problem the minimal amount of populations to process ($step$) for each thread are calculated using Equation.6.1, and the missing number of populations ($extraPop$) are calculated using Equation.6.2. Now with both the $step$, and $extraPop$ the first started threads are all provided with one extra population until all extra populations defined in $extraPop$ are divided between the threads. The final algorithm is provided in Algorithm 3 and shows each thread being started with a start population ($startPop$) and end population ($endPop$). The multi-threading is optimized for the use-case of files with a large number of populations. If the distance between the start and end population ($endPop - startPop$) is smaller then the user-defined population batch size ($\alpha$) then the population batch size is adjusted to the end population minus the start population ($\alpha = endPop - startPop$).



Master Thread: Initialize
Master Thread: Data-sets database
Synch
Spawn thread
Worker Thread 1: Data generation
1 to N Threads
Worker Thread N: Data generation
Wait for Thread 2, Exit
Wait for Thread N, Exit
Master Thread: Get file to proccess
Master Thread: Thread controller
Worker Thread 1: Start
Worker Thread N: Start
Worker Thread 1: Pre-processing
Worker Thread N: Pre-processing
Worker Thread 1: CNN inference
Worker Thread N: CNN inference
Worker Thread 1: Post-processing
Worker Thread N: Post-processing
Worker Thread 1: Exit
Worker Thread N: Exit
Master Thread: Gather results
Master Thread: Exit

Figure 6.2: Multi-threading diagram of CNN inference with ASDEC. solid lines represent control, and dotted lines represent data-dependencies between threads, and blocks of threads (represent in blocks in dotted lines). In blue dotted lines present the possibility to start a specified number of threads given by the user. CNN inference is performed on the user's selected hardware platform (GPU, CPU)

$$step = floor(P/T) \tag{6.1}$$

$$extraPop = P - stepSize * T \tag{6.2}$$

## 6.2 Profiling

When either performing training or inference with ASDEC timing reports are created to show the total execution time in combination with a breakdown of all separate processes. The training and inference are both performed on a CPU (Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz), where single-core, and multi-core (5 threads) parameters are provided to ASDEC, the number of threads is arbitrarily selected. Data generation was not profiled, because while ASDEC supports data generation it is not subject to hardware acceleration, and ASDEC is only responsible for starting the coalescent simulation software with the correct parameters.

### 6.2.1 Training

For training, data-sets 1 until and including 5 are used (see table 4.1), which all have 3000 populations. All data-sets were pre-trained to avoid differences in input data between runs, therefore data-generation is 0 seconds for all runs. A chromosome length of 100.000 in combination with extraction at the centre region (location of the sweep in each population, position 50.000), and the extent of the extraction is 28 SNPs in both directions. The window size was 50 SNPs, and the step size between windows was 1 SNP. This leads to a final number of 35.000 images, where 80% are used for training and 20% are used for validation. Finally, the CNN architecture of SweepNet was used to define the model in Keras.

When performing training with ASDEC, a breakdown is provided for all processes taking more than 1% of the execution time (Fig.6.3). From Fig.6.3, it can be concluded that around 96% and 90% of the execution time is used for training on Tensorflow 2 for multi-core and single-core respectively. Other tasks that take less than 1% of the execution time are initialization tasks and clean up after execution tasks.

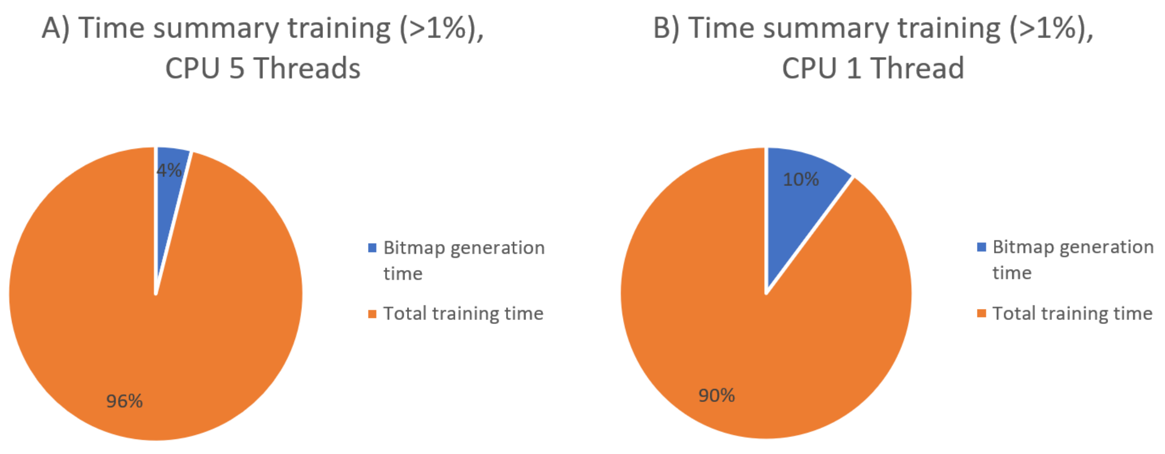


Figure 6.3: Execution time breakdown training with ASDEC

### 6.2.2 Inference

For inference data-set 1, was used, which consists of 10 populations and is pre-trained to avoid differences in input data between runs. A chromosome length of 100.000, where the complete genome was taken (no extraction). The window size was 50 SNPs, and the step size between windows was 1 SNP. The trained model created from run 5 was used for profiling the inference. Lastly, a population batch size of 2 was used, meaning that each file only consists of 1 full run of a thread (as illustrated in Fig.6.2 (image generation, inference, post-processing)), and a total of two files were provided (one neutral, one selective) so each thread completes two runs, one for each file. The number of images created for inference is around 13.000 per thread per file, resulting in approximate 130.000 images to create, and process.

In Fig.6.4 the results are provided and show all processes taking more than 1% of the execution time. Fig.6.4 shows two CNN executions (multi-core execution and the single-core execution) of the neutral file and the selective file of data-set 1. Then 46% of the execution time is required for processing the selective file (TEST), and 54% of the execution time is required for processing the neutral file (BASE). The small difference is because the selective file contains more SNPs, and therefore more images are required. When observing the log files of ASDEC, it was observed that around 99% of the time per thread is spent on the prediction performance within TensorFlow 2. The image generation and post-processing are together only responsible for around 0.6% of the total execution time. The other tasks that require less than 1% of the execution time are initialization and clean up processes.

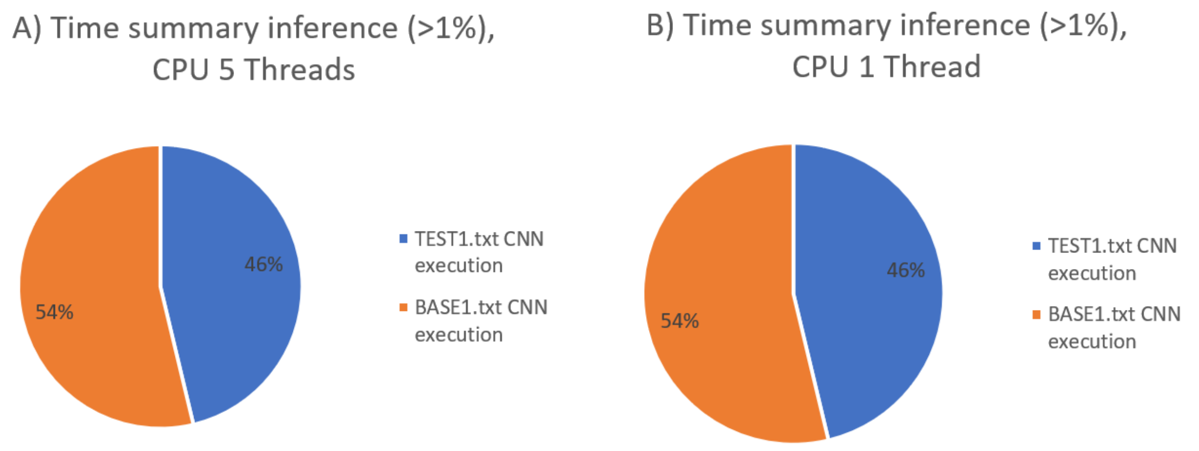


Figure 6.4: Execution time breakdown inference with ASDEC

## 6.3 DPU hardware acceleration

While the current implementation of ASDEC already provides a vast array of supported hardware a significant speedup could still yield importance to compete with the low execution times of some signature-based methods and tools. While it is not achievable to surpass or achieve comparable execution times with tools such as RAiSD, a significant speedup increases the usability of ASDEC. To achieve this hardware acceleration with the help of FPGAs and DPUs (Alveo cards) using Vitis AI is explored. Deep-Learning Processor Unit (DPU) hardware acceleration with Vitis AI provides inference acceleration, where still for example a GPU can be used for training in the earlier discussed way. The version of Vitis AI used in this research is 1.3.

Vitis AI is used for inference acceleration because of the use of only predefined optimized IP cores, and its design aimed towards high efficiency and ease of use[3]. Also earlier performed research showed great performance increases by using Xilinx FPGA boards such as the ZCU102, and ZCU104 in combination with Vitis AI as presented by Wang and Gu[17], and Qasaimeh et al. [56]. The acceleration with the help of Vitis AI was considered during the complete research, wherein steps such as the NAS represented in Chapter 5 considered Vitis AI support.

To integrate the workflow of both ASDEC and Vitis AI more work is performed by addressing quantization, evaluation, and compilation of the model. In Fig.6.6 the combination of the tooling is shown here first of all the setup of the used tooling is required. While it is possible to only use pre-synthesized hardware configurations the options for synthesizing parameter driving hardware designs is possible and shown by the yellow blocks in Fig.6.6. The green blocks in Fig.6.6 show the steps already implemented in ASDEC, this consists of steps such as training a model and performing an evaluation. In the blue blocks in Fig.6.6 the steps performed by Vitis AI are shown. Notice that up until the compilation step all models are independent of a hardware platform. In Vitis AI the hardware platform is introduced by the synthesized model delivering a hardware architecture configuration in the form of .json file, and by combining this file with the quantized model a platform-specific model is created. The cross-compilation of Vitis AI is performed on a device (FPGA, Alveo card, etc.) or a host, and generates the final executable.

After evaluation it is possible to further fine-tune the quantized model, while this possibility is provided by Vitis AI, it is not used and/or implemented during this research. Nonetheless the step is provided in Fig.6.6 for completeness.

Two different DPUs are considered within this research, namely the DPUCZDX8G [5] and DPUCAHX8H [4]. Both DPUs are programmable engines optimized for deep neural networks but offer different optimized platforms. The DPUs are elaborated more in the following sub-sections. To be able to estimate a theoretical throughput of a DPU the number of model operations is defined by the number of multiplies and accumulate (MAC) operations. The DPUs will only be used for inference, and still need a host system for image generation, image pre-processing, and post-processing. In Fig.6.5 the relation between the host system and the accelerator is illustrated. The communication between the host system and the accelerator can be made using different standards such as Peripheral Component Interconnect Express (PCIe) (Alveo U50), Universal Serial Bus (USB), or Ethernet. A pre-trained SweepNet model created with ASDEC was used for this flow and resulted in a final quantized .xmodel for a SweepNet model. The steps present in Fig.6.6 were followed, and the following chapter will go deeper into the error created by the quantization, and two DPUCZDX8G and DPUCAHX8H.

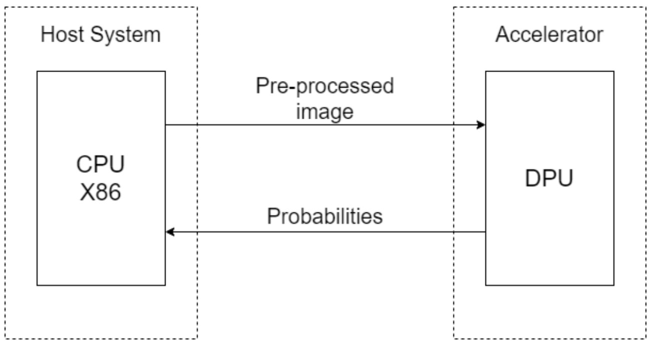
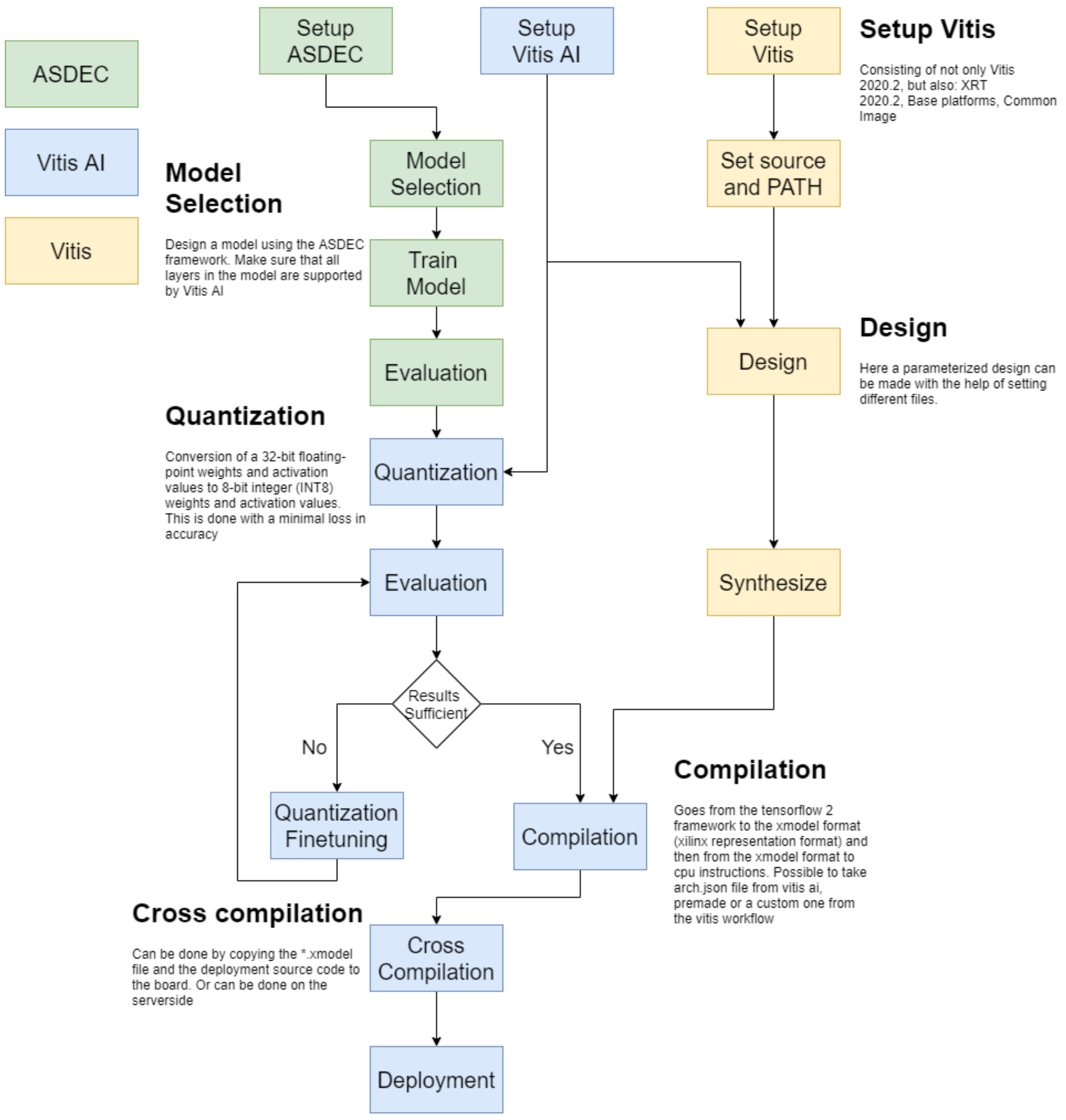


Figure 6.5: Host system and accelerator were the host systems that performs image generation, image pre-processing, and post-processing and the accelerator performs the inference



ASDEC
Vitis AI
Vitis
Setup ASDEC
Setup Vitis AI
Setup Vitis
Setup Vitis
Consisting of not only Vitis 2020.2, but also: XRT 2020.2, Base platforms, Common Image
Model Selection
Design a model using the ASDEC framework. Make sure that all layers in the model are supported by Vitis AI
Model Selection
Set source and PATH
Train Model
Evaluation
Design
Design
Here a parameterized design can be made with the help of setting different files.
Quantization
Conversion of a 32-bit floating-point weights and activation values to 8-bit integer (INT8) weights and activation values. This is done with a minimal loss in accuracy
Quantization
Evaluation
Synthesize
Results Sufficient
No
Yes
Compilation
Goes from the tensorflow 2 framework to the xmodel format (xilinx representation format) and then from the xmodel format to cpu instructions. Possible to take arch.json file from vitis ai, premade or a custom one from the vitis workflow
Quantization Finetuning
Compilation
Cross compilation
Can be done by copying the *.xmodel file and the deployment source code to the board. Or can be done on the serverside
Cross Compilation
Deployment

Figure 6.6: Overview of the acceleration workflow by the usage of ASDEC, Vitis, and Vitis AI

### 6.3.1 Size of SweepNet

For the exact specification defined by Xilinx, and possibly efficiency numbers Xilinx was contacted using their forum[72]. It was determined that a CNN architecture is defined by the number of model operations, and these model operations (flops, in TensorFlow terminology) are defined by their number of MACs. As instructed by Xilinx the number of MACs was multiplied by a factor of 2 to account for the DPUs counting the multiply and accumulate as 2 separate operations. Flops are based on the number of floating-point operations performed by hardware, after quantization floating-point operations are no longer performed, and therefore it is assumed that floating-point operations and 8-bit integer operations are equal, or less resulting in a more conservative estimate. The output of the determination of the flops for SweepNet is provided in listing 6.1 and uses a window size of 50. The final output presented in the listing 6.1 assumes non-fused MACs as also done by Xilinx.

Listing 6.1: Output flops determination

```
==================Model Analysis Report======================

Doc:
scope: The nodes in the model graph are organized by their
names, which is hierarchical like filesystem.
flops: Number of float operations.
Note: Please read the implementation for the math behind it.

Profile:
node name | # float_ops
_TFProfRoot (--/13.94m flops)
  conv2d_1/Conv2D (6.55m/6.55m flops)
  conv2d_2/Conv2D (5.53m/5.53m flops)
  dense/MatMul (1.26m/1.26m flops)
  conv2d/Conv2D (238.34k/238.34k flops)
  max_pooling2d/MaxPool (110.59k/110.59k flops)
  max_pooling2d_1/MaxPool (94.21k/94.21k flops)
  max_pooling2d_2/MaxPool (78.85k/78.85k flops)
  conv2d/BiasAdd (29.79k/29.79k flops)
  conv2d_1/BiasAdd (25.57k/25.57k flops)
  conv2d_2/BiasAdd (21.60k/21.60k flops)
  dense_1/MatMul (128/128 flops)
  dense/BiasAdd (32/32 flops)
  dense_1/Softmax (10/10 flops)
  dense_1/BiasAdd (2/2 flops)

======================End of Report==========================
Flops: 13,935,692
```

### 6.3.2 DPUCAHX8H

The DPUCAHX8H DPU is provided within Xilinx Vitis AI inference acceleration software framework and offers programmable core optimized for high throughput application, and CNNs [4]. The DPU cores on the programmable logic (PL) of various Alveo cards (U280, U50, U50LV (Low Voltage)), these cards are aimed towards data centre application. The Alveo U50 is optimized for workloads in financial computing, machine learning, computational storage, and data-science application. The U50 is built on Xilinx UltraScale+ architecture, and offers a maximum power draw of 75Watts, for more information see the data-sheet of the Alveo U50 [73].

In Fig.6.7, a top-level block diagram of the DPUCAHX8H is given, within this top-level diagram, various parameters can be used to control the speed, and therefore the resource requirements. The parameters of the DPUCAHX8H are the number of DPU cores which can vary between 1 to 3, and each DPU core can contain five processing engines (PE) [4]. The Alveo cards also have a High Bandwith Memory (HBM) interface, and this HBM is divided by the DPUCAHX8H for both the storage of temporary data (virtual banks), and the storage of instructions, input images, output results, and user data (system memory) [4].

The instructions for the DPU are created in the compilation step illustrated in Fig.6.6 and are ran on the host server (present on system memory of the host system). Lastly, the on-chip memory consists of private local memory for each PE, and the global memory is shared between all PEs per DPU core [4].

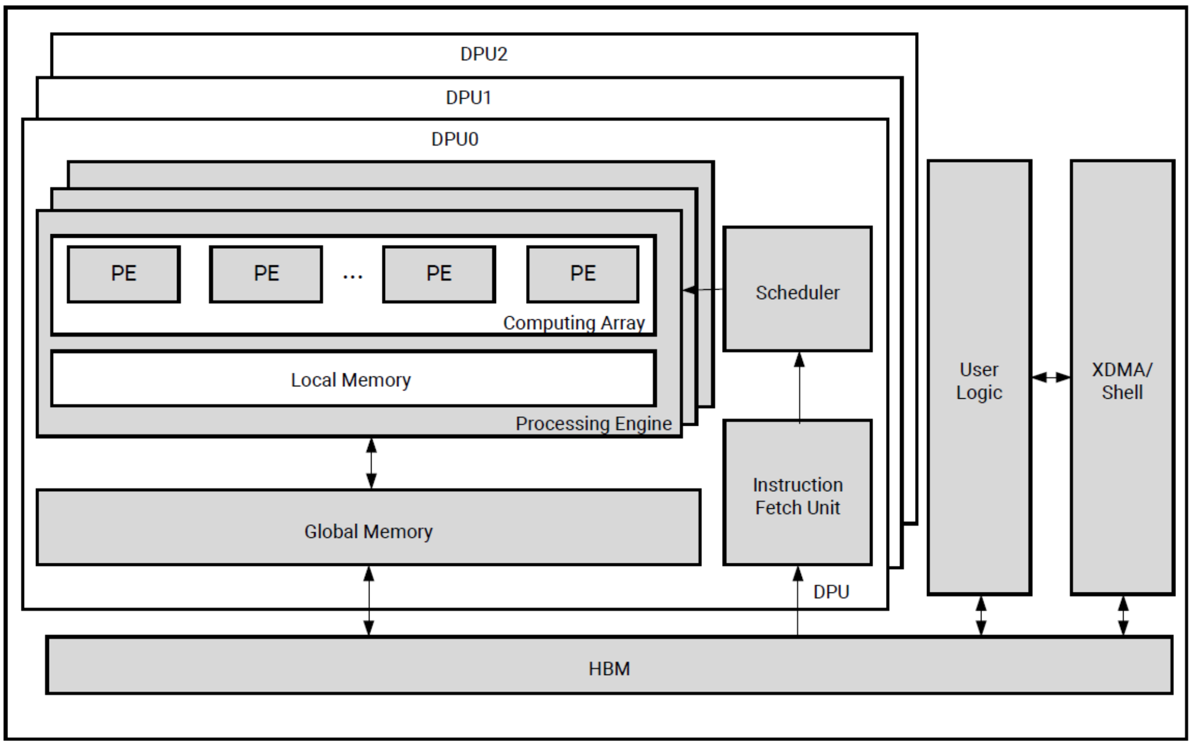
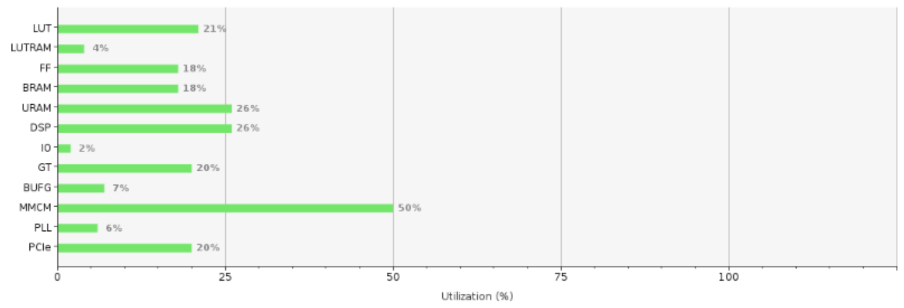
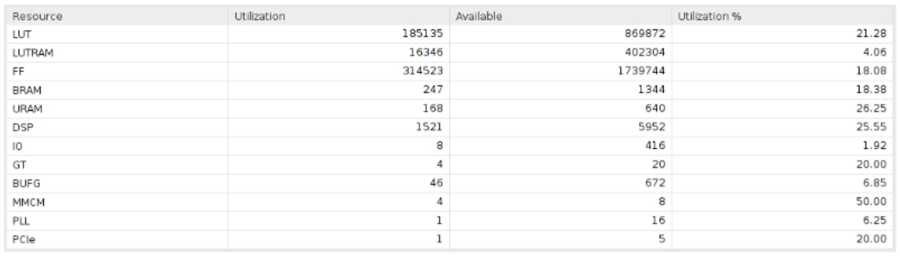


Figure 6.7: Top-level DPUCAHX8H[4]

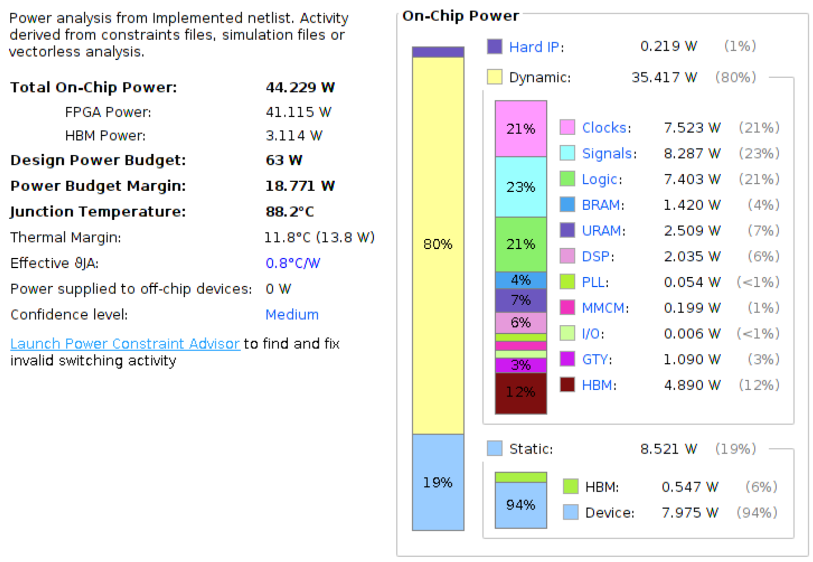


A



| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 185135 | 869872 | 21.28 |
| LUTRAM | 16346 | 402304 | 4.06 |
| FF | 314523 | 1739744 | 18.08 |
| BRAM | 247 | 1344 | 18.38 |
| URAM | 168 | 640 | 26.25 |
| DSP | 1521 | 5952 | 25.55 |
| IO | 8 | 416 | 1.92 |
| GT | 4 | 20 | 20.00 |
| BUFG | 46 | 672 | 6.85 |
| MMCM | 4 | 8 | 50.00 |
| PLL | 1 | 16 | 6.25 |
| PCIe | 1 | 5 | 20.00 |

B



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

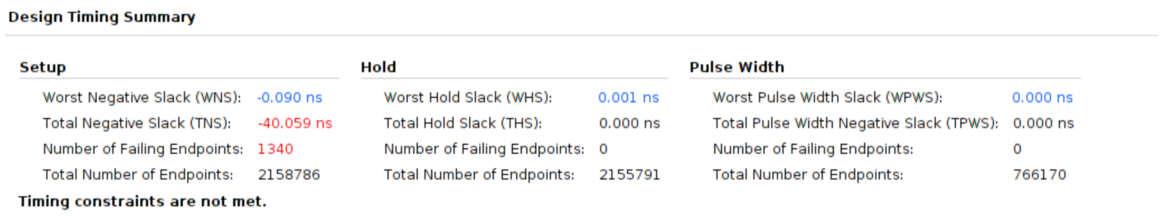
| | |
|---|---|
| **Total On-Chip Power:** | **44.229 W** |
| FPGA Power: | 41.115 W |
| HBM Power: | 3.114 W |
| **Design Power Budget:** | **63 W** |
| **Power Budget Margin:** | **18.771 W** |
| **Junction Temperature:** | **88.2°C** |
| Thermal Margin: | 11.8°C (13.8 W) |
| Effective ϑJA: | 0.8°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Medium |

Launch Power Constraint Advisor to find and fix invalid switching activity

**On-Chip Power**

| | | |
|---|---|---|
| Hard IP: | 0.219 W | (1%) |
| Dynamic: | 35.417 W | (80%) |
| Clocks: | 7.523 W | (21%) |
| Signals: | 8.287 W | (23%) |
| Logic: | 7.403 W | (21%) |
| BRAM: | 1.420 W | (4%) |
| URAM: | 2.509 W | (7%) |
| DSP: | 2.035 W | (6%) |
| PLL: | 0.054 W | (<1%) |
| MMCM: | 0.199 W | (1%) |
| I/O: | 0.006 W | (<1%) |
| GTY: | 1.090 W | (3%) |
| HBM: | 4.890 W | (12%) |
| Static: | 8.521 W | (19%) |
| HBM: | 0.547 W | (6%) |
| Device: | 7.975 W | (94%) |

C



**Design Timing Summary**

| **Setup** | | **Hold** | | **Pulse Width** | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | -0.090 ns | Worst Hold Slack (WHS): | 0.001 ns | Worst Pulse Width Slack (WPWS): | 0.000 ns |
| Total Negative Slack (TNS): | -40.059 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 1340 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 2158786 | Total Number of Endpoints: | 2155791 | Total Number of Endpoints: | 766170 |

**Timing constraints are not met.**

Figure 6.8: Vitis Analyzer DPUCAHX8H on an Alveo U50 System Info, two DPU core with three PEs, running on 300MHz, synthesized, A shows resource usage, B shows timing summary, and C shows power summary

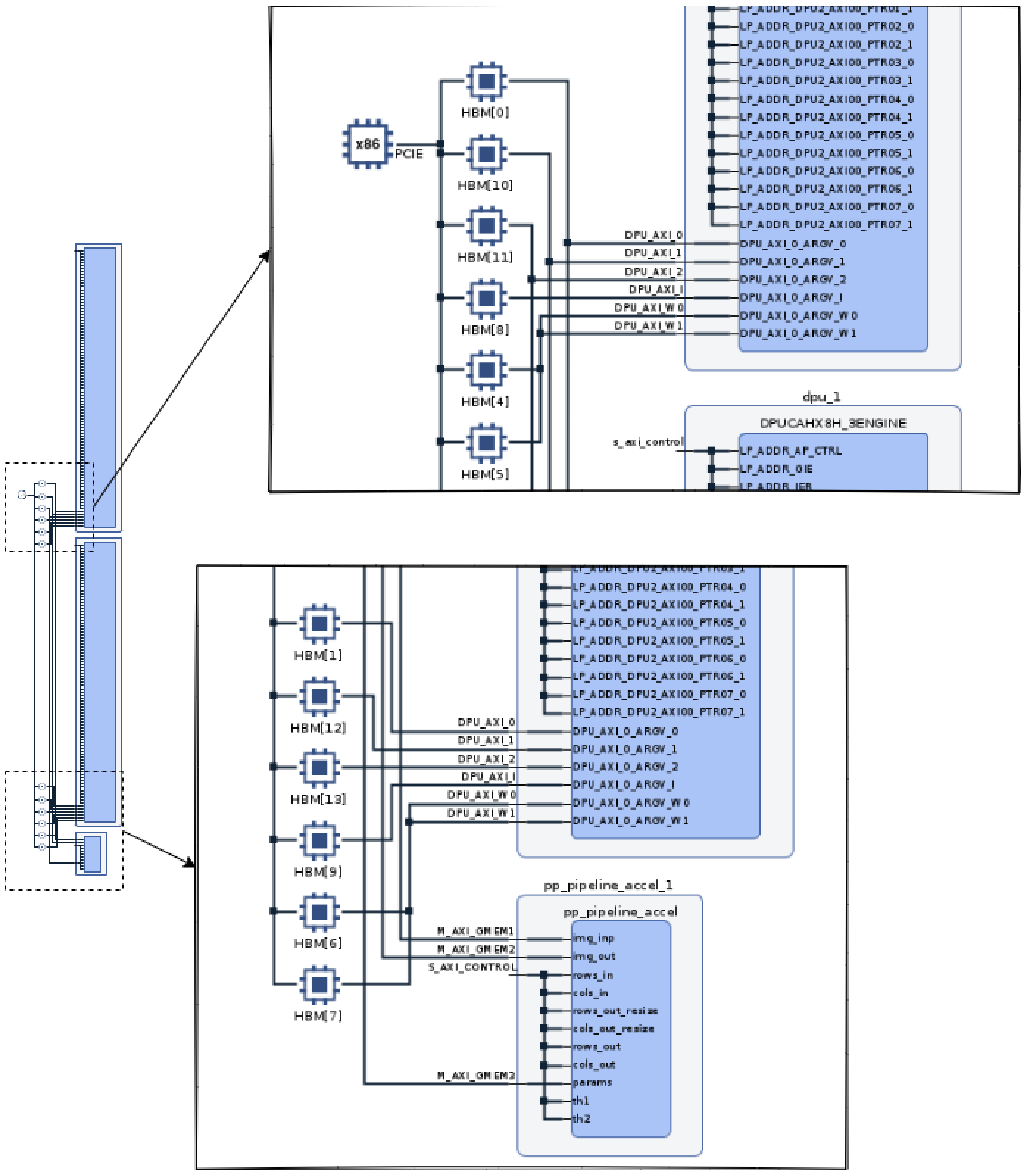


Figure 6.9: Vitis Analyzer DPUCAHX8H on an Alveo U50 System Diagram, two DPU cores with three PEs, running on 300MHz

From the documentation presented in [4] the peak performance defined by the number of Giga Operations Per Second (GOPs) on the Alveo U50, running a frequency of 300MHz is 7373 GOPs, and the DPU architecture contains 2 DPU cores both running 3 PEs. To account for losses, an efficiency range of between 15% and 70% is used, with 30% as a conservative estimate [72] (kindly provided by Xilinx during personal contact, provided for the DPUCZDX8G). The final formula for estimation of the number of operations is given in Equation.6.3.

$$OpsEstimation = (NumberModelOps/DPUOps) * Efficiency \tag{6.3}$$

For compilation, the provided DPUCAHX8H arch.json can be used to adjust the generic .xmodel for the Alveo U50. The cross-compilation is performed on the host machine because no Alveo card is available the deployment is skipped, but the theoretical throughput provides the information for a comparison with the other hardware implementations. In Fig.6.8 the synthesized design is shown for the Alveo U50 card and shows the resources used, as recommended in the manual [4].

While the suggested design by Vitis AI, consisting of 2 DPU cores both running 3 PEs should meet all constraints. After synthesizing problems concerning timing constraints that were not met, according to Vivado were observed. Because a pre-built was used to generate the bit-stream there is no easy ability for this version of Vitis AI (1.3) to decrease the number of PEs, DPU cores or clock frequency. Future versions of Vitis AI are going to add support for this, therefore for now the execution numbers (theoretical throughput) are used with a side note that timing constraints could be a problem for this design. See Fig.6.9 for the System diagram, and Fig.6.8 for the system info implemented (generating bit-stream, not deployed) design using Vitis Analyzer and Vivado, and Vivado.

### 6.3.3 DPUCZDX8G

The DPUCZDX8G architecture is provided within the Xilinx Vitis AI inference acceleration software framework and is designed for the Zynq UltraScale+ MPSoC [5]. Equally to the DPU-CAHX8H, the DPUCZDX8G architecture offers a configurable computation engine optimized for CNN inference. The FPGA used in combination with this architecture is a general-purpose evaluation board. The FPGA is the Xilinx ZCU 102, containing a Zynq UltraScale+ XCZU9EG-2FFVB1156E MPSoC (multiprocessor system-on-chip), with high-speed DDR4 SODIMM and component memory interfaces, and various features were most importantly the FPGA posses programmable logic for implementation a DPU [74]. More information on the Xilinx ZCU102 is provided in the manual [74].

Just like the DPUCAHX8H DPU instructions are stored on off-chip memory, and the DPU instructions controlling the execution of the ZCU102 are generated by the compiler step shown in Fig.6.6. Buffers are implemented on on-chip memory for fast access, and data is reused as much as possible to reduce the number of calls to external memory[5].

Usage of resources such as digital signal processing (DSP) slices, look up tables (LUT), block random-access memory (RAM), and UltraRam can be controlled by parameters presented in the DPUCZDX8G architecture, and are based on the programmable logic resources. The parameters include various DPUCZDX8G architectures as shown in table 6.1, and multiple DPUCZDX8G can be combined together [5]. Each DPUCZDX8G architecture achieves a different level of parallelism, parallelism is divided into pixel parallelism (PP), input channel parallelism (ICP) (depth of parallelism on input matrix), and output channel parallelism (OCP) (depth of parallelism on output matrix). The final peak performance per clock cycle is calculated with the help of Equation.6.4 [5].

$$PeakPerformance/perClock = PP * ICP * OCP * 2 \tag{6.4}$$

Each DPU has its global memory pool, and the various DPUs share a high-speed data tube as shown in Fig.6.10. Besides controlling the number, and size of the DPU it is possible to set extra parameters based on the use case options include: Average pooling, various activation functions, Element-wise multiplication, Low power mode, random-access memory (RAM) usage, and much more as described in [5]. To account for losses an efficiency range of between 15% and 70% is used, with 30% as a conservative estimate[72].

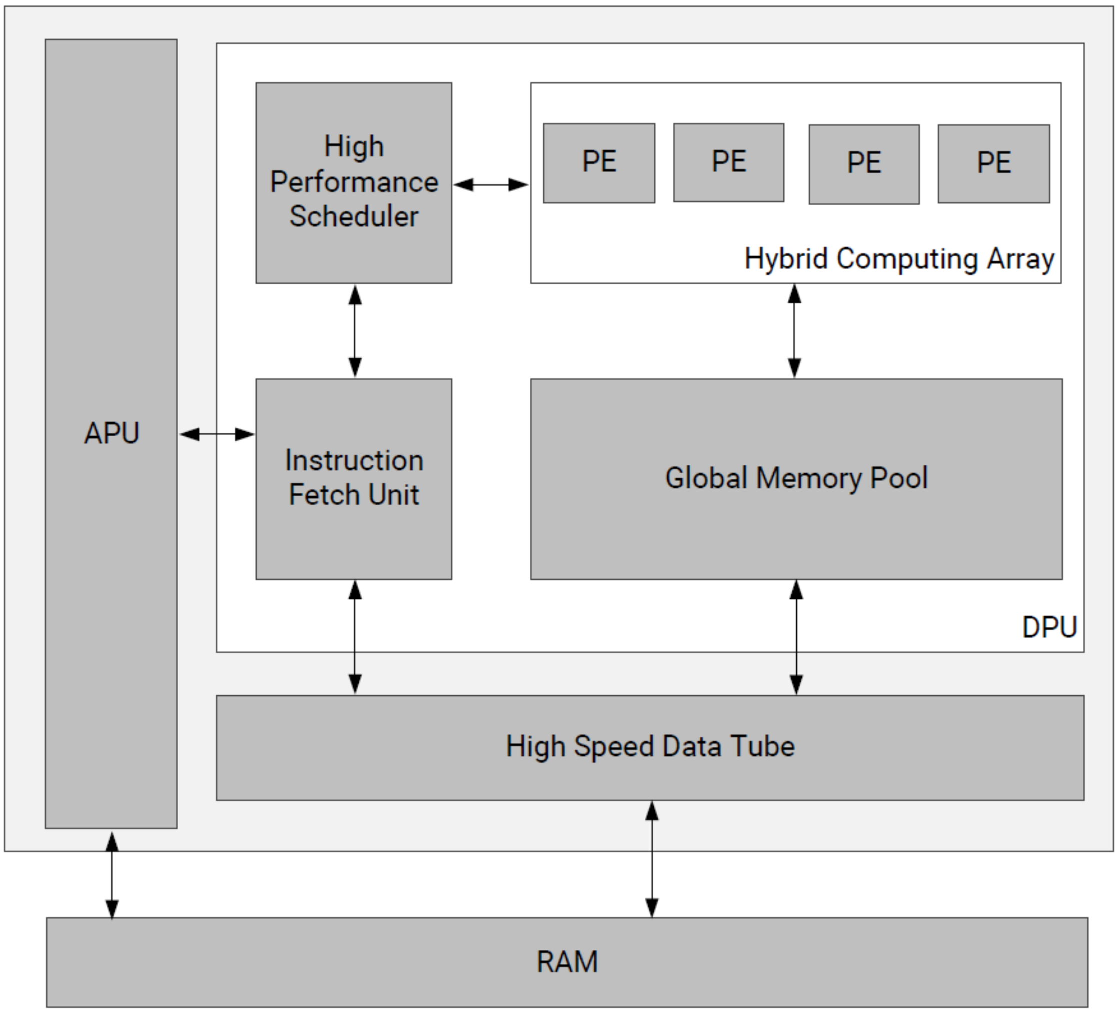


Figure 6.10: Top-level DPUCZDX8G[5]

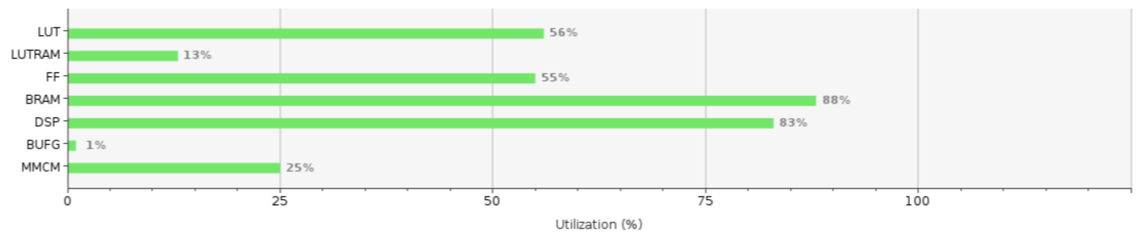
Table 6.1: DPUCZDX8G configuration settings[5]

| **DPUCZDX8G Architecture** | **Pixel Parallelism (PP)** | **Input Channel Parallelism (ICP)** | **Output Channel Parallelism(OCP)** | **Peak(operations /per clock)** |
|---|---|---|---|---|
| B512 | 4 | 8 | 8 | 512 |
| B800 | 4 | 10 | 10 | 800 |
| B1024 | 8 | 8 | 8 | 1024 |
| B1152 | 4 | 12 | 12 | 1152 |
| B1600 | 8 | 10 | 10 | 1600 |
| B2304 | 8 | 12 | 12 | 2304 |
| B3136 | 8 | 14 | 14 | 3136 |
| B4096 | 8 | 16 | 16 | 4096 |

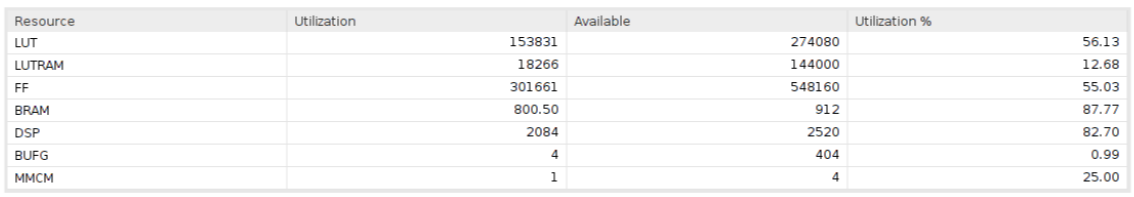
Two different designs were tested using the ZCU102 FPGA, one of them showing a really small design running one B512 DPUCZDX8G core, and another showing a large design running 3 B4096 DPUCZDX8G cores. Both designs are synthesized, and implemented (generating bit-stream, not deployed), and are used in comparison to more conventional hardware such as CPU and GPU. The summaries of the one DPUCZDX8G B512 is shown in Fig.6.13, and the summaries of the design with 3 DPUCZDX8G B4096 is shown in Fig.6.11 and Fig.6.12. The final results of both designs is a folder containing the files to flash to the ZCU102 FPGA and an arch.json file used for the compilation step in Fig.6.6.

The final throughput is calculated using Equation.6.5 of the one B512 DPUCZDX8G core design on the ZCU102 (running on 150MHz) is this 1 image per $6.05 \times 10^{-4}$ seconds for efficiency of 30%. For the 3 B4096 DPUCZDX8G cores design (running on 150MHz) this is 1 image per $2.52 \times 10^{-5}$ seconds for an efficiency of 30%. Equation.6.5 and the efficiency percentages were kindly provided during personal contact with Xilinx [72].

$$Throughput = NumberModelOps/(DPUOps * Efficiency * ClockFrequency) \tag{6.5}$$

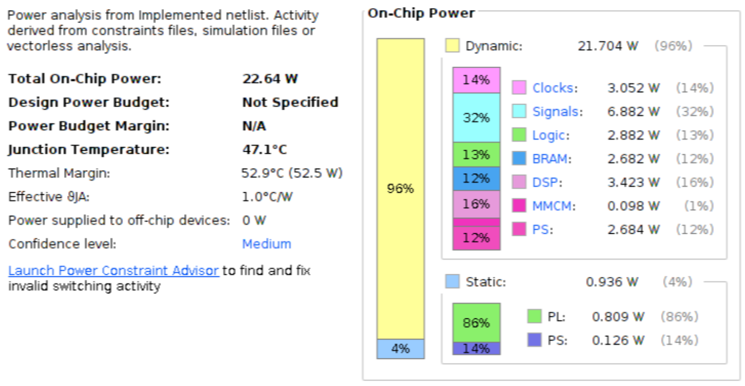


A



| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 153831 | 274080 | 56.13 |
| LUTRAM | 18266 | 144000 | 12.68 |
| FF | 301661 | 548160 | 55.03 |
| BRAM | 800.50 | 912 | 87.77 |
| DSP | 2084 | 2520 | 82.70 |
| BUFG | 4 | 404 | 0.99 |
| MMCM | 1 | 4 | 25.00 |

B



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

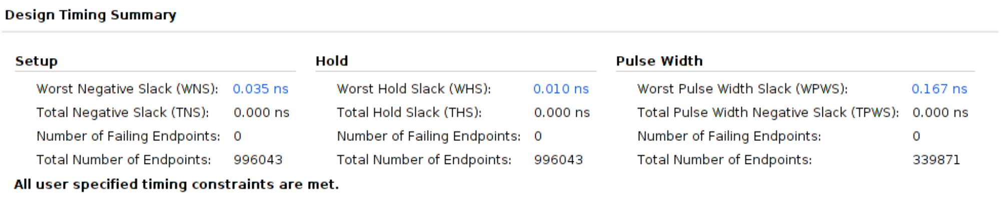
| | |
|---|---|
| **Total On-Chip Power:** | **22.64 W** |
| **Design Power Budget:** | **Not Specified** |
| **Power Budget Margin:** | **N/A** |
| **Junction Temperature:** | **47.1°C** |
| Thermal Margin: | 52.9°C (52.5 W) |
| Effective ϑJA: | 1.0°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Medium |

Launch Power Constraint Advisor to find and fix invalid switching activity

**On-Chip Power**

| | | |
|---|---|---|
| Dynamic: | 21.704 W | (96%) |
| Clocks: | 3.052 W | (14%) |
| Signals: | 6.882 W | (32%) |
| Logic: | 2.882 W | (13%) |
| BRAM: | 2.682 W | (12%) |
| DSP: | 3.423 W | (16%) |
| MMCM: | 0.098 W | (1%) |
| PS: | 2.684 W | (12%) |
| Static: | 0.936 W | (4%) |
| PL: | 0.809 W | (86%) |
| PS: | 0.126 W | (14%) |

C



**Design Timing Summary**

| **Setup** | | **Hold** | | **Pulse Width** | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 0.035 ns | Worst Hold Slack (WHS): | 0.010 ns | Worst Pulse Width Slack (WPWS): | 0.167 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 996043 | Total Number of Endpoints: | 996043 | Total Number of Endpoints: | 339871 |

**All user specified timing constraints are met.**

Figure 6.11: Vitis Analyzer DPUCZDX8G on a ZCU102, 3 DPU core B4096, running on 150MHz, synthesized, A shows resource usage, B shows timing summary, and C shows power summary

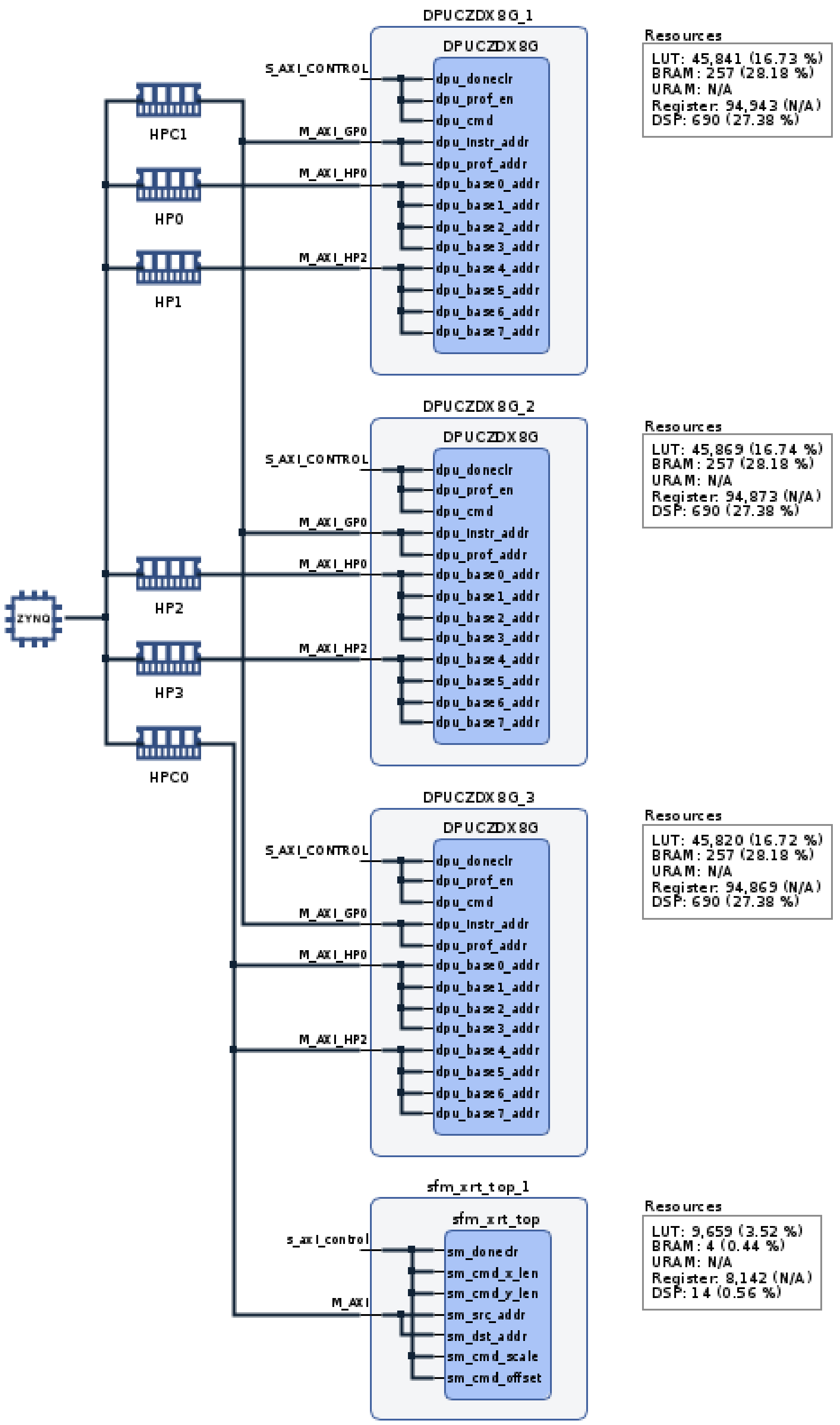
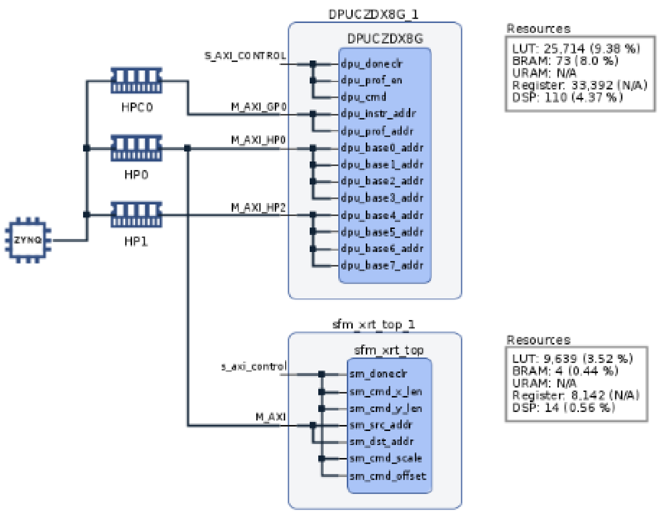


Figure 6.12: Vitis Analyzer DPUCZDX8G on a ZCU102, 3 DPU core B4096, running on 150MHz, synthesized System Diagram

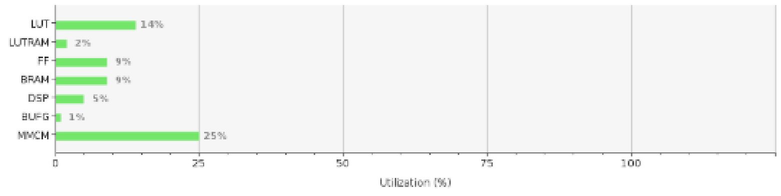
A



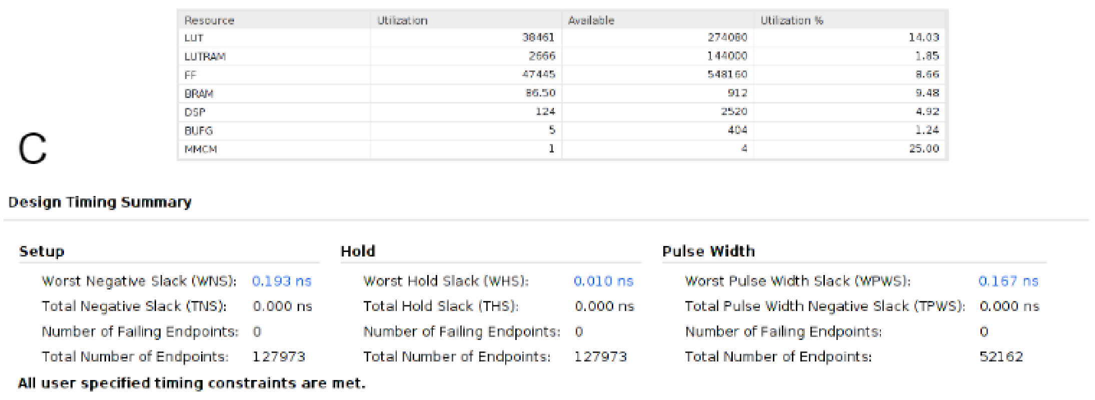
Resources
LUT: 25,714 (9.38 %)
BRAM: 73 (8.0 %)
URAM: N/A
Register: 33,392 (N/A)
DSP: 110 (4.37 %)

Resources
LUT: 9,639 (3.52 %)
BRAM: 4 (0.44 %)
URAM: N/A
Register: 8,142 (N/A)
DSP: 14 (0.56 %)

B



C



| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 38461 | 274080 | 14.03 |
| LUTRAM | 2666 | 144000 | 1.85 |
| FF | 47445 | 548160 | 8.66 |
| BRAM | 86.50 | 912 | 9.48 |
| DSP | 124 | 2520 | 4.92 |
| BUFG | 5 | 404 | 1.24 |
| MMCM | 1 | 4 | 25.00 |

**Design Timing Summary**

**Setup**

| | |
|---|---|
| Worst Negative Slack (WNS): | 0.193 ns |
| Total Negative Slack (TNS): | 0.000 ns |
| Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 127973 |

**Hold**

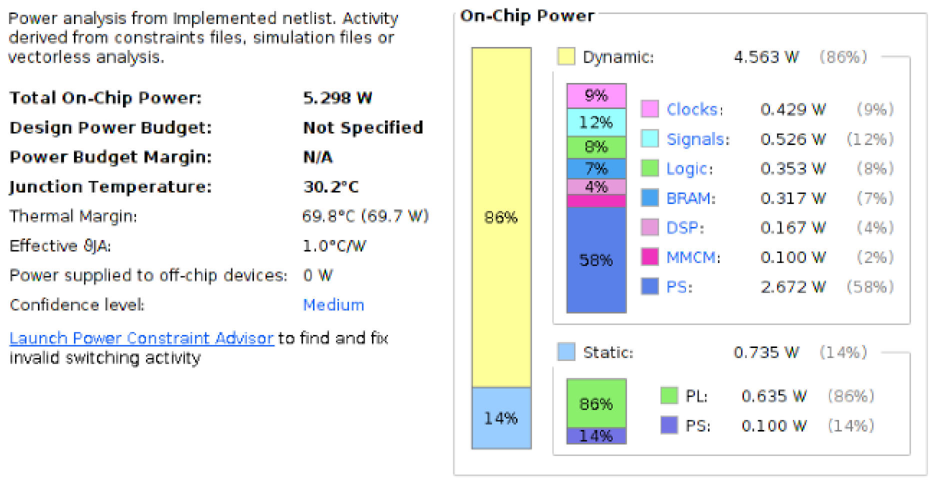
| | |
|---|---|
| Worst Hold Slack (WHS): | 0.010 ns |
| Total Hold Slack (THS): | 0.000 ns |
| Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 127973 |

**Pulse Width**

| | |
|---|---|
| Worst Pulse Width Slack (WPWS): | 0.167 ns |
| Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 52162 |

**All user specified timing constraints are met.**

D



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

| | |
|---|---|
| **Total On-Chip Power:** | **5.298 W** |
| **Design Power Budget:** | **Not Specified** |
| **Power Budget Margin:** | **N/A** |
| **Junction Temperature:** | **30.2°C** |
| Thermal Margin: | 69.8°C (69.7 W) |
| Effective ϑJA: | 1.0°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Medium |

Launch Power Constraint Advisor to find and fix invalid switching activity

**On-Chip Power**

| | | |
|---|---|---|
| Dynamic: | 4.563 W | (86%) |
| Clocks: | 0.429 W | (9%) |
| Signals: | 0.526 W | (12%) |
| Logic: | 0.353 W | (8%) |
| BRAM: | 0.317 W | (7%) |
| DSP: | 0.167 W | (4%) |
| MMCM: | 0.100 W | (2%) |
| PS: | 2.672 W | (58%) |
| Static: | 0.735 W | (14%) |
| PL: | 0.635 W | (86%) |
| PS: | 0.100 W | (14%) |

Figure 6.13: Vitis Analyzer DPUCZDX8G on a ZCU102, 1 DPU core B512, running on 150MHz, synthesized, A shows a top-level overview, B shows resource usage, C shows timing summary, and D shows power summary

### 6.3.4 Quantization error

As stated in the manual of Vitis AI (version 1.3) [3], a drop in accuracy is possible when performing quantization. To compare the performance of the 32-bit floating-point (FP32), and 8-bit integer (INT8) SweepNet models, 9 different FP32 models are trained using data-sets 1 to 60, data-set 1, data-set 9, data-sets 61 to 70, data-set 61, data-set 70, data-sets 97 to 101, data-set 97, and data-set 101. After training, these models are quantized using the quantization provided by Vitis AI. For quantization, an evaluation data-set created from completely new coalescent simulation files equal to the data-set(s) used for training in ASDEC (around 2500 images per data-set) are used. After quantization, each model has two versions one INT8, and one FP32. Next, a completely new data-set created from completely new coalescent simulation files equal to the data-set(s) used for training in ASDEC (around 2500 images per data-set) are used to evaluate the performance of each model. Both for quantization and evaluation the default batch size of 50 was used. The final results are presented in table 6.2, while some data-sets show a small increase in accuracy (negative difference in table 6.2) after quantization for the other data-sets a decrease in accuracy is observed. While there is a loss in accuracy for some of the tested data-sets the loss is within (around 2% to 4%) margins, compared with quantization losses presented by Qasaimeh, et al.[56].

Table 6.2: Quantization results

| **Training data-set** | **32-bit Floating Point** | | **8-bit Integer** | | **Deviation** | |
|---|---|---|---|---|---|---|
| | accuracy | loss | accuracy | loss | accuracy | loss |
| 01 | 98.36% | 0.3315 | 98.00% | 0.332 | 0.37% | 0.15% |
| 60 | 89.21% | 0.4413 | 88.93% | 0.4455 | 0.31% | 0.95% |
| 01-60 | 92.78% | 0.3927 | 93.41% | 0.3854 | -0.67% | -1.86% |
| 61 | 96.96% | 0.3478 | 97.04% | 0.3498 | -0.08% | 0.58% |
| 70 | 83.39% | 0.4888 | 80.29% | 0.5047 | 3.86% | 3.25% |
| 61-70 | 82.96% | 0.5051 | 82.93% | 0.5086 | 0.04% | 0.69% |
| 97 | 89.00% | 0.4199 | 88.93% | 0.4203 | 0.08% | 0.10% |
| 101 | 83.21% | 0.4685 | 83.50% | 0.4674 | -0.35% | -0.23% |
| 97-101 | 91.04% | 0.4121 | 90.82% | 0.412 | 0.24% | -0.02% |

# 7 RESULTS AND DISCUSSION

This chapter focuses on the final evaluation of ASDEC in comparison with the signature-based methods and tools for the defined metrics presented in Chapter 5 and execution time. Next to the comparison with signature-based tooling, also a comparison between the different hardware implementations of ASDEC is provided. These include CPU (multi-core), GPU, DPU (FPGA), DPU (Alveo U50). Lastly a scan of the first human chromosome from the Yoruba population, 1000Genomes project [67] is run to identify possible candidate genes for selection.

## 7.1 Experimental setup

The first experimental setup is deployed for comparison between the ASDEC framework and the whole-genome signature-based methods and tools. To keep execution times reasonable a separate experimental setup is created for the ASDEC acceleration comparison. Lastly, a setup is created for running the human chromosome.

### 7.1.1 Quality of ASDEC

To compare ASDEC with signature-based methods and tools, a large scale comparison is created, which includes all data-sets implemented within the ASDEC framework (see table 4.1). For training of the various CNNs required for inference, equal parameters are used consisting of the SweepNet CNN architecture, a window size of 50, step size of 1, extraction mode with an extent of 28 SNPs, batch size of 1, and 6 epochs. Besides training equal parameters are deployed for inference consisting of window size of 50, step size of 1, population batch size of 5, and post-processing mode ”Grid mode normal” with a grid size of 4000, and a grid range of 1% of the chromosome length in both directions(see Chapter 4).

The corresponding metrics are computed as presented in Chapter 5, and the results from signature-based tools as presented in [21] are used. One major difference between the signature-based tools, and ASDEC is the number of populations in ASDEC are limited to 100, while in [21] a population size of 1000 was used. The reduction in population size was needed due to the great number of experiments to perform, and the given time-frame (multiple weeks of computation required over multiple machines).

The experiment is divided within the different confounding factors present in table 4.1. Starting with the bottleneck confounding factor the following data-sets were used for both training (extraction point 500.000 bp, chromosome length 1.000.000), and inference (chromosome length 1.000.000): 1, 9, 12, 23, 24, 36, 45, 48, 57, 60, and 1-60 (all data-sets). For the Migration confounding factor for training (extraction point 50.000 bp, chromosome length 100.000), and inference (chromosome length 100.000) the following data-sets were used: 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 61-70 (all data-sets). For migration, three different scenarios are present in data-set 92-97 both the recombination, and the sweep is present in the centre of the genome, ASDEC was unable to train with these data-sets (under-fitting). Migration data-sets 97-101 contained the migration region again in the centre of the genome, but the sweep was located at 30% of the genome. It was concluded to train the CNN with an extraction point of 30.000 bp with a chromosome length of 100.000 on data-sets: 97, 98, 99, 100, 101, and 97-101. For migration inference (chromosome length 100.000) the following data-sets were used: 92, 93, 94, 95, 96, 97, 98, 99, 100, 101. In the final migration scenario, no sweep was present meaning that no training data exists for this scenario. The earlier trained migration models were used (data-set 97 to 101), and for inference (chromosome length of 100.000) the following data-sets were used: 71, 74, 77, 78, 81, 84, 85, 88, and 91.

### 7.1.2 ASDEC hardware performance

To compare the various hardware platforms where ASDEC could be run on, a small test is designed with a run-time of around 10 minutes including both training, and inference. The experiment consists of the training of pre-trained coalescent simulation files from data-set 1 to 5 (see table 4.1), and inference on data-set 1. For the comparison between CPU, and GPU inference only 1 population is used, because of timing constraints of the used Google Colab service, and for a comparison between threads a local machine is used and the number of population was increased to 10. The results are only used for the comparison of execution times concerning the comparison of other metrics the test does not contain enough complexity. To compare the execution time of ASDEC with the signature-based methods and tools a linear relation between execution time and the number of populations is assumed. Considering ASDEC this assumption is tested by running with 1 population, and running 10 populations and determining the throughput of 24.6 and 24.8 images per second respectively. The small difference is due to the variation in the size of the input data. For the signature-based methods and tools, the assumption is justified because each population is considered as a separate data-set.

During training, a chromosome length of 100.000 is provided, extraction is used on 50.000 positions of base pairs (bp), the extent of the extraction is 28 SNPs in both directions, a window size of 50 SNPs, step size of 1 SNP, 17.500 neutral images, 17.500 selective images, and a split of 80% used for training, and 20% used for validation. For inference only data-set 1 is used consisting of 10 or 1 population(s). The number of populations in data-set 1 is dependent on the test being performed, for Google Colab only 1 population is used due to the timing constraint of the Google Colab service. Lastly, a chromosome length of 10.000, no extraction, a window size of 50 SNPs, and a step size of 1 is used.

### 7.1.3 Human Genome

To demonstrate the handling of real data with ASDEC a scan is performed of the first chromosome of the human genome (Yoruba population, 1000Genomes data-set [68]). The scan was used to identify several candidate genes (top 0.05%). ASDEC with SweepNet was trained with all evolutionary models that include a sweep, and a window size of 50 in combination with a step size of 1. A total number of 800.000 data points were generated using post-processing mode ”grid position normal” (post-processing based on grid-points and a range from each grid-point as discussed in Chapter 4), with an extent of 2.500 positions of base pairs (bp) in both directions of each of the 800.00 grid points.

## 7.2 Quality of ASDEC

The quality of ASDEC in comparison with signature-based methods and tools such as SweepFinder2, SweeD, OmegaPlus, and RAiSD is considered for 5 scenarios. The various scenarios are defined by the confounding factor (definition of the confounding factors presented in Chapter 4) and the location of the sweep in combination with the confounding factor.

The first confounding factor is the bottleneck (data-sets 1 (weak bottleneck) to 60 (severe bottleneck) in table 4.1) where the sweep is located at the centre of the genome. In Fig.7.1A,B,D no model misspecification is considered. In Fig.7.1A the TPR rates are provided for an FPR of 5%, for weak to mild bottlenecks (data-sets 1, 9, 12, and 45) equally high TPR values for all tools are observed. When considering more severe bottlenecks ASDEC achieves a higher degree of sensitivity in comparison with the signature-based methods and tools RAiSD, OmegaPlus, SweeD, and SweepFinder2. For the most severe bottleneck (data-set 60) ASDEC achieves 4.1x, 4.7x, 5.9x, and 5.9x higher TPR values respectively. Generally, ASDEC considerably outperforms the signature-based methods and tools for the more severe bottleneck cases and achieves a similar TPR value for the mild bottleneck scenarios.

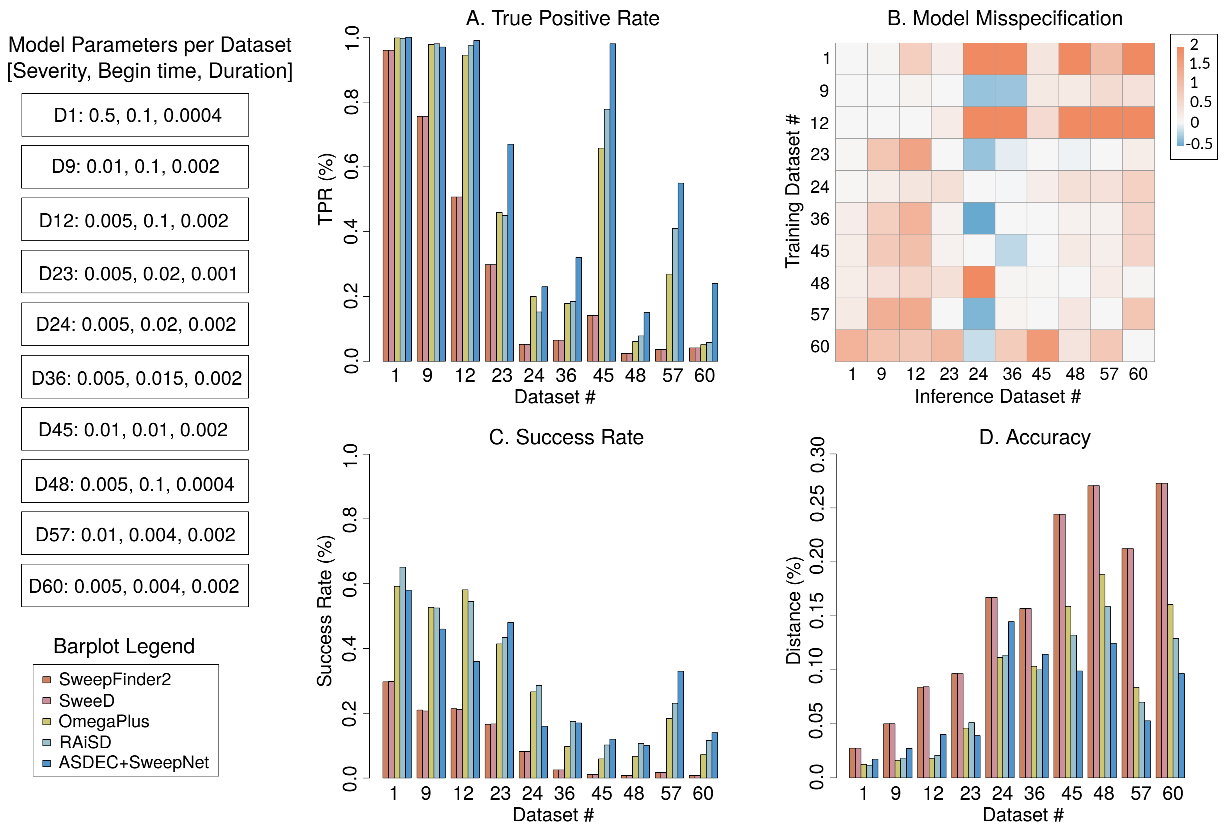


Figure 7.1: Evaluation of the bottleneck confounding factor with varying severity, begin time, and duration. A shows the TPR (by a 5% FPR), B shows evaluation with model misspecification (color represents $log10(\frac{TPR_{ii}}{TPR_{ij}})$ were $i$ is the training data-set, and $j$ is the evaluation data-set), C shows success rate ($e = \%1 \times L$), and D shows the distance from the real selection target as a percentage of the total genome length.

Fig.7.1B presents the TPR values when model misspecification is considered, where the diagonal correspond to the correct model specification. Each cell is defined by its training train data-set ($i$), and its testing data-set ($j$), were the value of each cell corresponds to $log10(TPR_{ii}/TPR_{ij})$. Notice that $TPR_{ii}$ corresponds to the correctly specified models. The heat map shows that ASDEC is relatively robust to model misspecification, with an observable decrease in TPR values when increasing the deviation between the training data-set ($i$) and the testing data-set ($j$).

Fig.7.1C provides the success rate, and Fig.7.1D provides the distance error. Bot the success rate and distance error show similar results as were observed for the TPR values presented in Fig.7.1A. Were for the mild bottleneck scenarios, equal performance to a small decrease in performance is observed (some data-sets for example data-set 24 a decrease in performance is observed). Considering the more severe bottleneck scenarios a clear increase in performance for both the success rate and the distance error is observed. When the performance of ASDEC is compared with the signature-based methods and tools for data-set 57 an increase of 1.4x, 1.8x, 19.4x, and 19.4x for the success rate, and a decrease of 1.3x, 1.6x, 4.0x, and 4.0x is observed for the distance error (lower is better) compared with RAiSD, OmegaPlus, SweeD, and SweepFinder2 respectively.

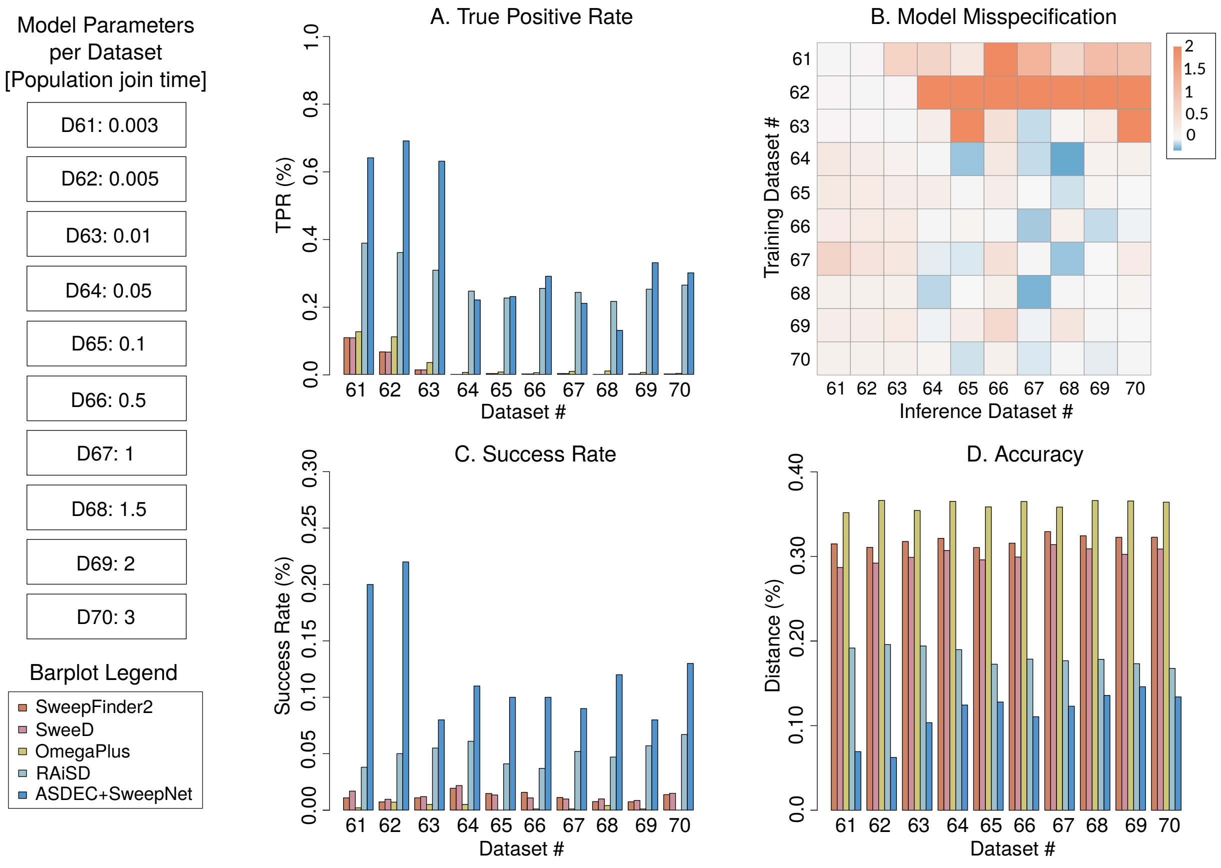


Figure 7.2: Evaluation of the migration confounding factor with varying population join time. A shows the TPR (by a 5% FPR), B shows evaluation with model misspecification (color represents $log10(\frac{TPR_{ii}}{TPR_{ij}})$ were $i$ is the training data-set, and $j$ is the evaluation data-set), C shows success rate ($e = 1\% \times L$), and D shows the distance from the real selection target as a percentage of the total genome length.

Fig.7.2 presents the migration confounding factor. An equal comparison as presented for the bottleneck confounding factor (Fig.7.1) is used for the migration confounding factor. In Fig.7.2A the TPR values by an FPR of 5% are presented and show that the performance of ASDEC is comparable with the results of RAiSD. The tools based on only a single signature (either SFS or LD) perform poorly for all migration cases. For the hardest migration scenario (data-set 70) ASDEC achieves 1.1x, 100x, 250x, and 250x the TPR value of RAiSD, OmegaPlus, SweeD, and SweepFinder2 respectively. Fig.7.2B presents the model misspecification with the correct model specification on the diagonal it can be observed that ASDEC is relatively robust to model misspecification, but is more affected more by an increasing population join time of the training data-set to the evaluation data-set.

Fig.7.2C, and Fig.7.2D show that for both the success rate and distance error ASDEC outperforms all signature-based methods and tools for all data-sets. Achieving a 5.26x higher success rate than the previously highest achieved success rate, which was achieved by RAiSD. While ASDEC outperforms all signature-based methods and tools for all data-sets it is still confounded by migration, as observed by the relatively low TPR and success rate, and relatively high distance error.

A neutral evolutionary model with single 5-kb or 10-kb recombination regions, and a model with three 5-kb recombination regions in combination with an intensity range from 2 to 100, relative to the rest of the genome. No sweep is present within all data-sets and the difference between the neutral data, and selective data is the presence of recombination. ASDEC was trained with recombination scenarios where a sweep is present outside of the recombination region (data-sets 97 to 101). In the assessment of the effect of recombination heterogeneity (false specifying recombination as a selective sweep) on the FPR, the cutoff frequency was set on the 95th percentile. Comparing the neutral data without recombination and the neutral data with recombination it was concluded that there is a negligible effect on the FPR as shown in table 7.1 where the expected value is 5% for all methods and tools, while ASDEC is slightly more sensitive than the signature-based methods and tools. The average values over all selected runs are as followed: SweepFinder2: 3.81%, SweeD: 3.78%, OmegaPlus: 5.84%, RAiSD: 4.93 and ASDEC+SweepNet: 6.44%.

Table 7.1: The effect of recombination on the FPR

| **Data-set #** | **71** | **74** | **77** | **78** | **81** | **84** | **85** | **88** | **91** |
|---|---|---|---|---|---|---|---|---|---|
| Recombination region size | 5kb | | | 10kb | | | 5kb×3 | | |
| Recombination intensity | 2 | 10 | 100 | 2 | 10 | 100 | 2 | 10 | 100 |
| SweepFinder2 | 3.9 | 4.4 | 3.8 | 3.8 | 3.6 | 4.2 | 3.7 | 3.6 | 3.3 |
| SweeD | 3.9 | 4.4 | 3.7 | 3.7 | 3.5 | 4.2 | 3.7 | 3.6 | 3.3 |
| OmegaPlus | 5.9 | 5.4 | 7.0 | 5.1 | 6.2 | 5.9 | 5.6 | 5.4 | 6.1 |
| RAiSD | 5.1 | 5.5 | 5.6 | 6.4 | 4.0 | 5.2 | 4.9 | 3.1 | 4.6 |
| ASDEC+SweepNet | 6.0 | 11.0 | 10.0 | 7.0 | 4.0 | 5.0 | 3.0 | 4.0 | 8.0 |

For the other recombination scenarios (data-sets 92-101), a sweep is present. The location of the sweep is equal to the location of the recombination hotspot at 50-kb for data-set 92 to 96, and for data-set 97 to 101 the sweep is at 30-kb outside of the recombination hotspot. Concerning the overlapping recombination hotspot and selective sweep scenarios, no ASDEC models could be trained due to the accuracy during training not achieving a value higher than 50%. Therefore Fig.7.3A, Fig.7.3C, Fig.7.3D for data-sets 92 to 96 are based upon a model misspecification scenario where the training data-sets are 97 to 101 (no overlap). In Fig.7.3A a TPR value by a 5% FPR value, showing that when the recombination hotspot and sweep region are overlapping ASDEC in combination with the other tools struggle to correctly classify the locus of the selective sweep. The performance impact of an overlapping recombination hotspot and selective region is also reflected in the low success rates, and high distance error provided in Fig.7.3C and Fig.7.3D.

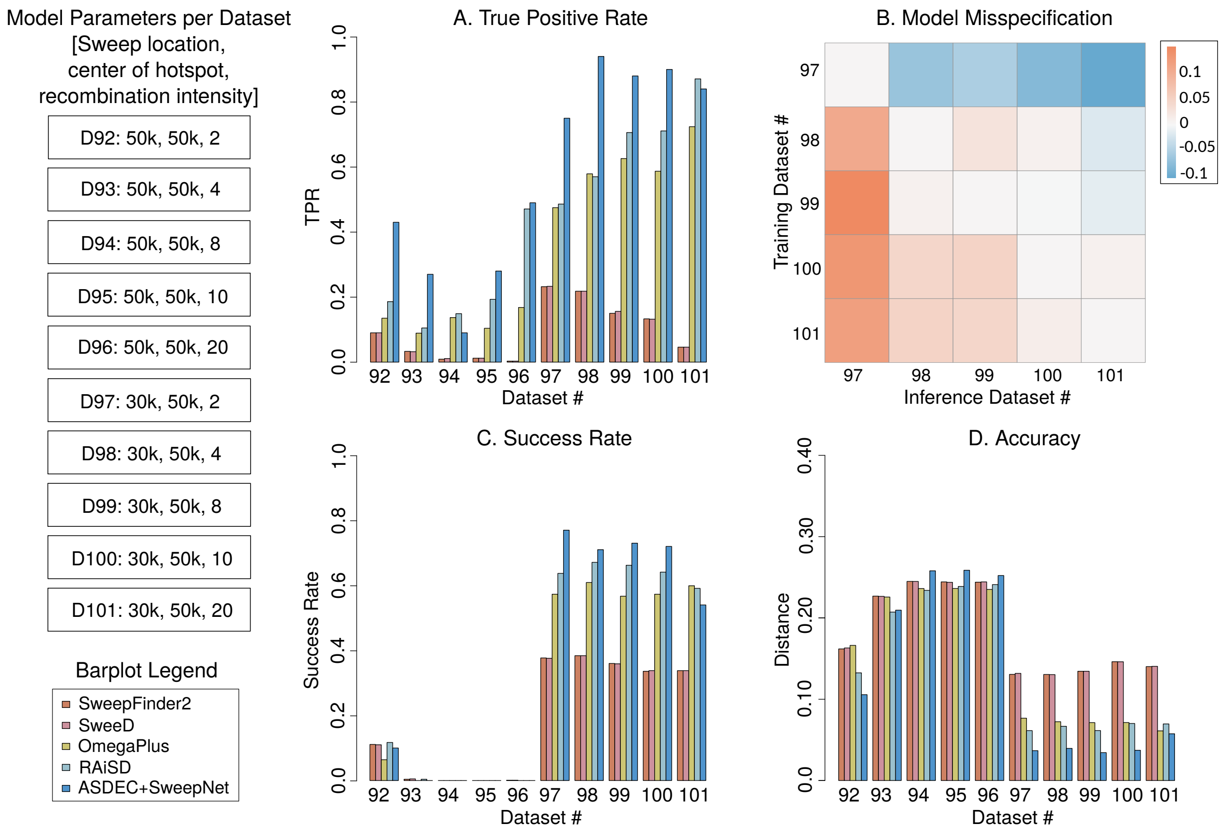


Figure 7.3: Evaluation of the recombination confounding factor with varying recombination intensity, and a selective sweep within the recombination region (data-set 92 to 96) and outside the recombination region (data-sets 97 to 101). A shows the TPR (by a 5% FPR), B shows evaluation with model misspecification (color represents $log10(\frac{TPR_{ii}}{TPR_{ij}})$ were $i$ is the training data-set, and $j$ is the evaluation data-set), C shows success rate ($e = 1\% \times L$), and D shows the distance from the real selection target as a percentage of the total genome length.

For the data-sets that have no overlap in the recombination hotspot, and a selective sweep the TPR value by a 5% FPR value are provided in Fig.7.3A. For the obtained TPR value ASDEC outperforms or matches the performance of the top-performing signature-based methods and tools, where for example for data-set 100 an increase of 1.3x, 1.5x, 6.8x, and 6.8x compared with RAiSD, OmegaPlus, SweeD, and SweepFinder2 respectively. Fig.7.3B provides the model miss specification cases equally as elaborated before, the effect of model misspecification with recombination hotspots with a lower recombination intensity than the correct recombination model tends to mostly overestimate the TPR. When a higher recombination intensity than the correct recombination model is considered then the TPR is mostly underestimated.

Fig.7.3C and Fig.7.3D provide results in line with the results observed in Fig.7.3A, where ASDEC outperforms signature-based methods and tools. ASDEC only takes a slight performance hit when considering data-set 101. When considering for example data-set 100 a success rate of 1.1x, 1.3x, 2.1x, and 2.1x and a distance error decrease of 1.9x, 1.9x, 3.9x, and 3.9x for ASDEC compared with RAiSD, OmegaPlus, SweeD, and SweepFinder2 respectively.

## 7.3 ASDEC hardware performance

In table 7.2 training in combination with inference is executed for multiple hardware platforms using Google Colab. For inference, only 1 population in data-set 1 was used, and therefore a thread count of 1 is used for all runs because a higher thread count will still default to the number of populations as discussed in Chapter 6.

Table 7.2: Execution time (in seconds) of different hardware architectures for both inference and training using ASDEC (* denotes not meeting a time constraint). Times of DPUs include required CPU execution time and therefore provide the complete required time/throughput.

| **Hardware** | **Thread Count** | **Total Number of Training Images** | **Total ASDEC Training Time** | **Total Number of Inference Images** | **Total ASDEC Inference Time** | **Images Per Second Inference** |
|---|---|---|---|---|---|---|
| LOCAL: AMD FX 6300 6 cores | 1 | 35000 | 1092.71579 | 12911 | 524.4974 | 24.6159 |
| Google Colab CPU Virtual CPU | 1 | 35000 | 1044.257 | 12911 | 588.2169 | 21.9493 |
| Google Colab Tesla P4 GPU | 1 | 35000 | 409.2712 | 12911 | 405.3848 | 31.8488 |
| ZCU102 DPUCZDX8G 3 DPU CORES, B4096, 150MHz | 1 | - | - | 12911 | 15%: 17.7934 30%: 17.4680 70%: 17.2821 | 15%: 725.6061 30%: 739.1187 70%: 747.0736 |
| ZCU102 DPUCZDX8G 1 DPU CORES, B512, 150MHz | 1 | - | - | 12911 | 15%: 32.7610 30%: 24.9518 70%: 20.4894 | 15%: 394.0966 30%: 517.4376 70%: 630.1307 |
| Alveo U50* DPUCAHX8H 2 DPU CORES, 3PEs, 300MHz | 1 | - | - | 12911 | 15%: 17.3053 30%: 17.2240 70%: 17.1775 | 15%: 746.0720 30%: 749.5935 70%: 751.6228 |

Table 7.2 shows a vast improvement between running ASDEC on for example a CPU (Google Colab), and a GPU (Google Colab) with a 1.45x more images per second for inference, and an improvement of 2.55x faster execution for training. To determine the estimate for the throughput of the various DPUs first of all a run is performed where the image generation, pre-processing for the CNN (normalization of the input matrix), and post-processing is measured on the Google Colab CPU coming on an average time of 3 runs of 17.14264 seconds. The time required for the DPU to run is added to this average time giving a final estimate for running the complete ASDEC framework on a DPU.

Table 7.3: Inference on multiple hardware platforms considering power consumption measurements or estimation in watts, and timing information in seconds as throughput(* denotes not meeting a time constraint). Execution times are given in seconds. Times of DPUs include required CPU execution time and therefore provide the complete required time.

| **Hardware** | **Thread Count** | **Total Number of Images** | **Total Inference Time** | **Images Per Second** | **Total Average Power Consumption** | **Power Host** | **Power Accelerator** |
|---|---|---|---|---|---|---|---|
| CPU AMD FX6300 | 1 | 130343 | 5252.536 | 24.81525 | 66.23368 | 66.23368 | - |
| | 2 | 130343 | 2753.697 | 47.33382 | 90.40422 | 90.40422 | - |
| | 3 | 130343 | 2292.204 | 56.86361 | 90.49029 | 90.49029 | - |
| | 4 | 130343 | 1837.615 | 70.93054 | 92.21419 | 92.21419 | - |
| | 5 | 130343 | 1410.788 | 92.39018 | 93.98667 | 93.98667 | - |
| | 6 | 130343 | 1414.861 | 92.12424 | 93.78833 | 93.78833 | - |
| ZCU102 DPUCZDX8G 3 DPU CORES, B4096, 150MHz | 1 | 130343 | 15%: 135.9067 30%: 132.6218 70%: 130.7447 | 15%: 959.0624 30%: 982.8173 70%: 996.9276 | 48.4728 | 74.3055 | 22.640 |
| ZCU102 DPUCZDX8G 1 DPU CORES, B512, 150MHz | 1 | 130343 | 15%: 287.0122 30%: 208.1746 70%: 163.1245 | 15%: 454.1375 30%: 626.1235 70%: 799.0400 | 39.8018 | 74.3055 | 5.298 |
| Alveo U50* DPUCAHX8H 2 DPU CORES, 3PEs, 300MHz | 1 | 130343 | 15%: 130.9793 30%: 130.1581 70%: 129.6888 | 15%: 995.1420 30%: 1001.4206 70%: 1005.0444 | 59.2673 | 74.3055 | 44.229 |

Another feature of the ASDEC framework is the CPU multi-threading for all stages of the framework. To show the scaling of the CPU multi-threading inference on 10 populations of data-sets 1 is performed. Besides the CPU multi-threading, the DPUs are considered for a comparison to the multi-core performance. The DPU throughput is calculated with the theoretical throughput given in Chapter 6, and CPU execution times for image generation, pre-processing for the CNN (normalization of the input matrix), and post-processing. The final CPU time required by the DPU is measured on the AMD FX 6300 CPU for one thread with a resulting time of 129.3369 seconds (average of 3 runs). For the various DPUs, no multi-threading is used, since the host system delivers the pre-processed images to the design. Therefore having multiple threads means that there should be a solution to known which image corresponds to which thread, and this case is not considered within this research. Besides the correct correlation of images and threads, sufficient bandwidth between the CPU and DPU should be ensured to obtain stable execution. In table 7.3 the results of the various runs are provided, the idle power consumption of the AMD FX 6300 is measured and determined to be around 41.178 Watts. The idle power consumption was measured by reading the power sensor of the CPU with the lm_sensors software. The measurement was done after a reboot and after the completion of all startup processes. The power was measured over a period of multiple minutes and averaged to reduce the effects of possible outliers. Table 7.3 shows that increasing the number of threads yields significant performance gains while the scaling between different thread counts is not linear. The non-linear property only holds when the number of populations (in this experiment 10) modulo of the number of threads equals zero. When the modulo is not zero there exists an imbalance between the threads by the number of populations assigned to them (see Chapter 6). While the throughput can be greatly increased by multi-threading a DPU implementation provides the most throughput with lower power consumption as shown in table 7.3.

Table 7.4: Comparison between signature-based tools and ASDEC on various different hardware platforms(* denotes not meeting a time constraint) for data-set 1 (see table 4.1). Data-set 1 contained an average population size for both the neutral and selective file of 7000 SNPs and 6000 SNPs respectively. Times of DPUs include required CPU execution time and therefore provide the complete required time.

| **Tool** | **Thread Count** | **Populations per second** |
|---|---|---|
| RAiSD | - | 2.8583 |
| OmegaPlus | - | 0.2644 |
| SweeD | - | 0.03699 |
| SweepFinder2 | - | 0.02293 |
| ASDEC CPU: AMD FX 6300 | 1 | 0.001904 |
| ASDEC CPU: AMD FX 6300 | 6 | 0.007068 |
| ASDEC Google Colab GPU: Tesla P4 | 1 | 0.002467 |
| ASDEC ZCU102 DPUCZDX8G 3 DPU CORES, B4096, 150MHz | 1 | 15%: 0.07358 30%: 0.07540 70%: 0.07648 |
| ASDEC ZCU102 DPUCZDX8G 1 DPU CORES, B512, 150MHz | 1 | 15%: 0.03484 30%: 0.04804 70%: 0.06130 |
| ASDEC Alveo U50* DPUCAHX8H 2 DPU CORES, 3PEs, 300MHz | 1 | 15%: 0.07635 30%: 0.07683 70%: 0.07711 |

In table 7.4 the results are provided, where there is assumed that there is a linear relationship between the number of populations and the required execution time (as stated at the start of this chapter). The results in table 7.4 are obtained by the execution of data-set 1 (see table 4.1). Data-set 1 contained an average population size for both the neutral and selective file of 7000 SNPs and 6000 SNPs respectively. ASDEC on a ZCU102 (efficiency of 30%) in comparison with signature-based tools RAiSD, OmegaPlus, SweeD, and SweepFinder2 performs 0.026x, 0.285x, 2.039x, and 3.289x respectively. ASDEC in combination with a ZCU102 running 3 DPUCZDX8G B4096 cores can outperform SFS-based tools SweepFinder2 and SweeD, while the LD, and composite based tools OmegaPlus and RAiSD still outperform ASDEC. The usage of an accelerator such as the ZCU102 (3 DPUCZDX8G B4096 cores (efficiency of 30%)) leads to a performance increase of 10.7x over CPU AMD FX 6300 running 6 threads.

## 7.4 Human Genome

In table 7.5 the final results of the scan of the first chromosome of the human genome (Yoruba population, 1000Genomes data-set[68]) are shown. The scan consists of 800.000 data entries, were the top 0.05% of the outliers have been selected resulting in a final number of 500 data-points. The exact parameters used during the scan are provided at the start of this chapter. Within the scan 9 different candidate genes (top 0.05%) were identified. All 9 candidate genes located by ASDEC have already been identified previously by other methods as targets of positive selection [75],[76],[77],[78],[79],[80],[81],[82],[83].

Table 7.5: 9 candidate genes in human chromosome 1 of the 1000Genomes project, Yoruba population (GRCh37/hg19assembly) performed with ASDEC (scores of top 0.05%)

| Gene name | Position (start-end, in bp) | Region identified by ASDEC | Previous candidate gene identification and method used |
|---|---|---|---|
| SCMH1 | 41,492,871-41,627,104 | 41,570,580-41,625,444 | Johnson and Voigh[75], iHS [84] |
| VAV3 | 108,113,782-108,507,545 | 108,244,669-108,652,426 | Scheinfeldt et al[76], iHS [84] |
| MAGI3 | 113,933,475-114,224,924 | 114,013,341-114,284,515 | Ji et al [77], iHS [84] |
| SPAG17 | 118,496,288-118,727,848 | 118,716,729-118,742,182 | Kuhlwilm and Boeckx[78], n/a |
| FCRL2 | 157,715,523-157,746,922 | 157,570,369-157,926,03 | Matos et al. [79], dN/dS [85] |
| ALDH9A1 | 165,631,449-165,667,900 | 165,085,221-165,724,772 | Landini et al [80], nSL [86] |
| DISP1 | 222,988,431-223,179,337 | 223,016,450-223,038,632 | Wagner [81], dN/dS [85] |
| CDC42BPA | 227,177,566-227,505,826 | 227,177,271-227,199,703 | Grossman et al[82], CMS |
| SLC35F3 | 234,040,679-234,460,262 | 234,368,359-234,383,563 | Refoyo-Martínez et al.[83], GRoSS |

Johnson and Voight [75] showed in their research that by identifying shared selective events genomic loci (including selective sweeps) could be identified. Scheinfeldt et al. [76] presented a genomic analysis of high-altitude populations, and showed sweep detection resulted in targets present for adaption for high altitude. Ji et al. [77] showed adaptions to various solar energy-related traits, resulting in a genome-wide adaptation study concerning the presented solar energy-related traits concerning climate ambient temperature (CAT), ultraviolet radiation (UVR), and sunlight duration (SD). Kuhlwilm and Boeckx [78] identified 571 genes with a nucleotide mutation that changed a corresponding amino acid in the protein at high frequency. Matos et al. [79] showed the presence of a specific gene (FCRLS) is more widespread and older than was previously considered across mammals. The presence of this gene is stated by this research to be functional (under positive selection). Landini et al. [80] present the changes based on the adoption of cereal and rice-based diets. This research showed selective sweeps in 2379 individuals from 124 East Asian and South Asian populations. Wagner [81] proposed a simple and rapid sweep detection test, and applied this test to several thousand human-chimpanzee genes in different species (orthologs). Grossman et al. [82] developed a method to detect underlying genes and the advantageous mutations, combining multiple signals of selection. The method (dubbed Composite of Multiple Signals (CMS)) was used to identify candidate regions from the international haplotype map. Refoyo-Martínez et al. [83] present a method that can detect signatures of strong local adaptation across the genome. The presented method graph-aware Retrieval of Selective Sweeps (GRoSS) is used on bovine, codfish, and human population genomic data and new genes were discovered.

Various methods are used by earlier research for the identification of targets of positive selection. Johnson and Voight [75], Scheinfeldt et al. [76], and Ji et al. [77] used the method intergrated haplotype score (iHS) [84] to identify their candidate genes. Landini et al. [80] performed sweep detection with the haplotype-based statistic (nSL) developed by Ferrer-Admetlla et al. [86]. Both Refoyo-Martínez et al. [83] and Grossman et al. [82] deployed their own methods named GRoSS, and CMS respectively to identify their candidate genes. Lastly Matos et al. [79] and Wagner [81] used dN/dS [85] to identify their candidate genes for targets of positive selection.

# 8 CONCLUSION AND FUTURE WORK

This chapter provides the conclusion of the presented research and also provides various possible improvements that could be made in the future.

## 8.1 Conclusion

This work presented and accelerated ASDEC a framework for whole-genome sweep detection. The ASDEC framework showed superior performance over the current state of the art signature-based methods and tools. ASDEC can be deployed by other researchers for whole-genome sweep detection research. Other researchers should still consider appropriate parameters for their data when using ASDEC. ASDEC still required long execution times on conventional hardware to alleviate this ASDEC accelerated was introduced. ASDEC accelerated was designed and evaluated using ASDEC and Vitis AI in combination with Xilinx hardware (FPGA, data-centre accelerator card). Overall the presented work delivered an impressive result for the use of CNN based sweep detection methods. To conclude this work the various research (sub-)questions are answered. Before answering the main research question all research sub-questions are answered.

The first research sub-question provides the basic question on how to perform sweep detection. This question provides the initial starting point of the presented work.

”Is it possible to design a CNN that can be used in the detection of selective sweeps?”

To answer this question first of all the ASDEC framework was developed to be able to bring the data to an input that can be processed by a CNN. ASDEC also includes post-processing to reduce the granularity of the raw probabilities outputted by the CNN. With the help of the ASDEC framework, it now is possible to perform sweep detection with a CNN based method. For more information on how to perform sweep detection with ASDEC see Chapter 4.

To determine a sufficient CNN architecture for sweep detection with the ASDEC framework the next research sub-question was formulated.

”How should the CNN be modelled to achieve sufficient results with regards to the qualitative evaluation?”

Firstly the qualitative evaluation is defined by the sensitivity, accuracy, and success rate of each model. Next, a hand-designed NAS is used given a variable number of 2D-convolutional layers, max-pooling layers, 2D-convolutional filter sizes, dense layers, and the size of each dense layer. Within the NAS various input data-sets were used in combination with varying image numbers. The NAS resulted in a final CNN architecture dubbed SweepNet which is considered to be sufficient given the qualitative evaluation. The complete NAS is formulated in Chapter 5.

Before implementing a CNN accelerator it should be known which part should be accelerated, therefore the following research sub-question is formulated.

”Which parts of the model should be performed on the accelerator?”

The use of acceleration was only researched for CNN inference because training is only performed once while inference needs to be done multiple times for a single trained model and input data-set. To be able to answer the question of which part of ASDEC should be accelerated, profiling embedded in ASDEC was used. The ASDEC profiling showed that around 99% of the execution time of ASDEC was used for CNN inference (pre-and post-processing are responsible for around 0.3% of the execution time). To reduce the execution time of the ASDEC framework it was determined to accelerate the complete inference on an FPGA or Alveo card with the help of Vitis AI, with the exception of image normalization which was done on the host system. Furthermore, pre-and post-processing is still done on the host system. In Chapter 6 the profiling of ASDEC is presented.

After deciding which part of the CNN should be accelerated the performance potential should be considered therefore the following research sub-question is formulated.

”What is the performance potential for acceleration of the developed sweep detection method?”

Comparing ASDEC on a general-purpose CPU (AMD-FX 6300 (6 threads)) and a general-purpose CPU (AMD-FX 6300 (1 thread)) in combination with Xilinx ZCU102 FPGA (3 DPUCZDX8G B4096 cores) showed a theoretical decrease of 10.7x in execution time when using an FPGA accelerator. The performance potential is elaborated in Chapter 7. Furthermore, when executing ASDEC on a GPU (Tesla P4), the execution time is reduced by 1.3x over a single-core CPU (AMD FX 6300). The Tesla P4 was available from Google Colab, greater improvements when using local hardware can be observed. The improvement with local hardware is due to constraints of cloud services such as queuing of hardware.

The various research sub-questions led up to the main research question presented at the beginning of this document is.

”Can an accelerated convolutional neural network outperform currently available methods and tooling, regarding sensitivity, accuracy, success rate, and execution times for the detection of a selective sweep?”

It can be concluded that the ASDEC framework which is an accelerated convolutional neural network can outperform the top-performing signature-based methods and tools concerning sensitivity, accuracy, and success rate. ASDEC (accelerated) is not able to compete with the top-performing signature-based method and tool (for example RAiSD) considering execution times. Considering the execution time, ASDEC is outperforming the SFS-based methods and tools (SweepFinder2 and SweeD), because of their computational complexity and being outperformed by the composite- (RAiSD) and LD-based (OmegaPlus). Overall it is concluded that ASDEC can outperform or match the performance of currently available methods and tools.

## 8.2 Future work

The current state of ASDEC is a fully usable and configurable framework for CNN based sweep detection. ASDEC in its current state can process ms formatted files, VCF formatted files, and FASTA formatted files (FASTA formatted files should first be converted by RAiSD). Besides data-sets generated by coalescent simulation software, ASDEC is also able to process real genomic data. ASDEC combined with SweepNet already showed great performance compared against signature-based methods. ASDEC is implemented on various hardware platforms consisting of CPU multi-core, GPU, FPGA (theoretical), and Alveo (DPU) (theoretical). ASDEC in its current state already shows promising performance and usability further improvements could build upon the existing ASDEC framework. The remainder of this section provides further possible improvements to the ASDEC framework.

A completely new feature for the ASDEC framework that can be implemented is the sorting of rows within images on genetic similarity. This feature was already discussed in research of Torada et al. [24] and Flagel et al. [14]. Within these two studies ([24] and [14]), it was discussed that sorting of rows within images based on genetic similarity can greatly improve sweep detection performance.

ASDEC could also benefit from a more extensive NAS. In this case, the NAS should not necessarily be restricted by the operations supported by Vitis AI. By not restricting the NAS to only Vitis AI-supported operations, a larger set of operations can be researched such as the drop out layer, L1 regularization, L2 regularization, and various activation functions. This could lead to more CNN architectures, and possibly CNN architectures with vastly better performance than the CNN architecture presented in this research (SweepNet). Besides the NAS itself, more extensive research can be put into the parameters used during pre-and post-processing which was for this research empirically determined. Determining improved parameters for both pre- and post-processing could greatly improve the performance of ASDEC and maybe even tailor pre-and post-processing for specific confounding factors.

Another improvement to the ASDEC framework could be the scaling of the CPU inference multi-threading. The scaling of the CPU inference multi-threading is now optimized for scenarios where a large number of populations are present within a single file. Optimization for the case when only one population is present (for now ASDEC will default to one thread) could yield greatly improved execution times for files with a single population. This optimization could for example be done internally in ASDEC. ASDEC then should be able to divide one large population into multiple smaller populations, and later combine the results on the correct positions.

Also, the prepared deployment on for example the Xilinx ZCU 102 FPGA can be performed providing real-world performance. This real-world performance should be compared with the theoretical performance presented in this research. Besides the implementation of the DPU also the multi-threading of the DPUs can be research possibly leading to great improvements in execution times as shown by Wang and Gu [17].

All the mentioned future improvements build upon the already existing ASDEC framework and should increase the usability of ASDEC further. While in its current state ASDEC is already a completely functional sweep detection method.

# REFERENCES

[1] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[2] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[3] Xilinx. *Vitis AI User Guide*, ug1414 (v1.3) edition, 2021.

[4] Xilinx. *DPUCAHX8H for Convolutional Neural Networks*, pg367 (v1.0) edition, 2021.

[5] Xilinx. *DPUCZDX8G for Zynq UltraScale+ MPSoCs*, pg338 (v3.3) edition, 2021.

[6] Julien Y Dutheil. *Statistical Population Genomics*. Springer Nature, 2020.

[7] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. 2001.

[8] John Frank Charles Kingman. The coalescent. *Stochastic processes and their applications*, 13(3):235–248, 1982.

[9] Richard R Hudson. Testing the constant-rate neutral allele model with protein sequence data. *Evolution*, pages 203–217, 1983.

[10] Richard Hudson. ms a program for generating samples under neutral models. 01 2002.

[11] Lefteris Koumakis. Deep learning models in genomics; are we there yet? *Computational and Structural Biotechnology Journal*, 2020.

[12] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] 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, 2019.

[15] Qianru Zhang, Meng Zhang, Tinghuan Chen, Zhifei Sun, Yuzhe Ma, and Bei Yu. Recent advances in convolutional neural network acceleration. *Neurocomputing*, 323:37–51, 2019.

[16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[17] Jin Wang and Shenshen Gu. Fpga implementation of object detection accelerator based on vitis-ai. In *2021 11th International Conference on Information Science and Technology (ICIST)*, pages 571–577. IEEE, 2021.

[18] Joseph J Vitti, Sharon R Grossman, and Pardis C Sabeti. Detecting natural selection in genomic data. *Annual review of genetics*, 47:97–120, 2013.

[19] Maxwell W Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321–332, 2015.

[20] John Maynard Smith and John Haigh. The hitch-hiking effect of a favourable gene. *Genetics Research*, 23(1):23–35, 1974.

[21] Nikolaos Alachiotis and Pavlos Pavlidis. Raisd detects positive selection based on multiple signatures of a selective sweep and snp vectors. *Communications biology*, 1(1):1–11, 2018.

[22] Kevin Y Yip, Chao Cheng, and Mark Gerstein. Machine learning and genome annotation: a match meant to be? *Genome biology*, 14(5):1–10, 2013.

[23] 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.

[24] 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):1–12, 2019.

[25] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[26] François Chollet et al. Keras.

[27] University of Leicester. Genetics for higher education.

[28] Eva Vallejos-Vidal, Sebastián Reyes-Cerpa, Jaime Andrés Rivas-Pardo, Kevin Maisey, José M Yáñez, Hector Valenzuela, Pablo A Cea, Victor Castro-Fernandez, Lluis Tort, Ana M Sandino, et al. Single-nucleotide polymorphisms (snp) mining and their effect on the tridimensional protein structure prediction in a set of immunity-related expressed sequence tags (est) in atlantic salmon (salmo salar). *Frontiers in genetics*, 10:1406, 2020.

[29] Brian Charlesworth and D Charlesworth. Population genetics from 1966 to 2016. *Heredity*, 118(1):2–9, 2017.

[30] Scott D Putney, Walter C Herlihy, and Paul Schimmel. A new troponin t and cdna clones for 13 different muscle proteins, found by shotgun sequencing. *Nature*, 302(5910):718–721, 1983.

[31] Motoo Kimura. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics*, 61(4):893, 1969.

[32] Tomoko Ohta. The neutral theory is dead. the current significance and standing of neutral and nearly neutral theories. *BioEssays*, 18(8):673–677, 1996.

[33] 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):1–15, 2013.

[34] Md Tauqeer Alam, Dziedzom K De Souza, Sumiti Vinayak, Sean M Griffing, Amanda C Poe, Nancy O Duah, Anita Ghansah, Kwame Asamoa, Laurence Slutsker, Michael D Wilson, et al. Selective sweeps and genetic lineages of plasmodium falciparum drug-resistant alleles in ghana. *Journal of Infectious Diseases*, 203(2):220–227, 2011.

[35] Pavlos Pavlidis and Nikolaos Alachiotis. A survey of methods and tools to detect recent and strong positive selection. *Journal of Biological Research-Thessaloniki*, 24(1):1–17, 2017.

[36] John M Braverman, Richard R Hudson, Norman L Kaplan, Charles H Langley, and Wolfgang Stephan. The hitchhiking effect on the site frequency spectrum of dna polymorphisms. *Genetics*, 140(2):783–796, 1995.

[37] Justin C Fay and Chung-I Wu. Hitchhiking under positive darwinian selection. *Genetics*, 155(3):1405–1413, 2000.

[38] Montgomery Slatkin. Linkage disequilibrium—understanding the evolutionary past and mapping the medical future. *Nature Reviews Genetics*, 9(6):477–485, 2008.

[39] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[41] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[42] Konstantinos Fatseas. Embedded neural network design on the zybo fpga for vision based object localization. Master's thesis, University of Twente, 2018.

[43] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[44] Mina Khoshdeli, Richard Cong, and Bahram Parvin. Detection of nuclei in h&e stained sections using convolutional neural networks. In *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 105–108. IEEE, 2017.

[45] Niels ten Dijke. Convolutional neural networks for regulatory genomics. Master's thesis, University Leiden, 2017.

[46] Hamed Habibi Aghdam and Elnaz Jahani Heravi. Convolutional neural networks. In *Guide to convolutional neural networks*. Springer, 2017.

[47] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[48] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[50] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[51] Xilinx. *Vivado Design Suite User Guide*, ug904 (v2020.2) edition, 2021.

[52] Xilinx. *Vitis High-Level Synthesis User Guide*, ug1399 (v2020.2) edition, 2021.

[53] Siemens. High-level synthesis & verification.

[54] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 33–36, 2011.

[55] Xilinx. *DNNDK User Guide*, ug1327 (v1.6) edition, 2019.

[56] Murad Qasaimeh, Kristof Denolf, Alireza Khodamoradi, Michaela Blott, Jack Lo, Lisa Halder, Kees Vissers, Joseph Zambreno, and Phillip H Jones. Benchmarking vision kernels and neural network inference accelerators on embedded platforms. *Journal of Systems Architecture*, 113:101896, 2021.

[57] Michael DeGiorgio, Christian D Huber, Melissa J Hubisz, Ines Hellmann, and Rasmus Nielsen. Sweepfinder2: increased sensitivity, robustness and flexibility. *Bioinformatics*, 32(12):1895–1897, 2016.

[58] Pavlos Pavlidis, Daniel Živković, Alexandros Stamatakis, and Nikolaos Alachiotis. Sweed: likelihood-based detection of selective sweeps in thousands of genomes. *Molecular biology and evolution*, 30(9):2224–2234, 2013.

[59] Nikolaos Alachiotis, Alexandros Stamatakis, and Pavlos Pavlidis. Omegaplus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinformatics*, 28(17):2274–2275, 2012.

[60] Rasmus Nielsen, Scott Williamson, Yuseob Kim, Melissa J Hubisz, Andrew G Clark, and Carlos Bustamante. Genomic scans for selective sweeps using snp data. *Genome research*, 15(11):1566–1575, 2005.

[61] Yuseob Kim and Wolfgang Stephan. Detecting a local signature of genetic hitchhiking along a recombining chromosome. *Genetics*, 160(2):765–777, 2002.

[62] Yuseob Kim and Rasmus Nielsen. Linkage disequilibrium as a signature of selective sweeps. *Genetics*, 167(3):1513–1524, 2004.

[63] Gregory Ewing and Joachim Hermisson. Msms: a coalescent simulation program including recombination, demographic structure and selection at a single locus. *Bioinformatics*, 26(16):2064–2065, 2010.

[64] Daniel R Schrider and Andrew D Kern. S/hic: robust identification of soft and hard sweeps using machine learning. *PLoS genetics*, 12(3):e1005928, 2016.

[65] Daniel R Schrider and Andrew D Kern. Soft sweeps are the dominant mode of adaptation in the human genome. *Molecular biology and evolution*, 34(8):1863–1877, 2017.

[66] Andrew D Kern and Daniel R Schrider. Discoal: flexible coalescent simulations with selection. *Bioinformatics*, 32(24):3839–3841, 2016.

[67] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[68] Peter H Sudmant, Tobias Rausch, Eugene J Gardner, Robert E Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, et al. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015.

[69] Garrett Hellenthal and Matthew Stephens. mshot: modifying hudson's ms simulator to incorporate crossover and gene conversion. 2006.

[70] 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):1–4, 2009.

[71] Unknown. Tensorboard: Tensorflow's visualization toolkit.

[72] Matthijs University of Twente and jheaton Xilinx Employee. Estimate throughput of a model, Jun 2021.

[73] Xilinx. *Alveo U50 Data Center Accelerator Card Data Sheet*, ds965 (v1.7.1) edition, 2020.

[74] Xilinx. *ZCU102 Evaluation Board User Guide*, ug1182 (v1.6) edition, 2019.

[75] Kelsey Elizabeth Johnson and Benjamin F Voight. Patterns of shared signatures of recent positive selection across human populations. *Nature ecology & evolution*, 2(4):713–720, 2018.

[76] Laura B Scheinfeldt, Sameer Soi, Simon Thompson, Alessia Ranciaro, Dawit Woldemeskel, William Beggs, Charla Lambert, Joseph P Jarvis, Dawit Abate, Gurja Belay, et al. Genetic adaptation to high altitude in the ethiopian highlands. *Genome biology*, 13(1):1–9, 2012.

[77] Lindan Ji, Dongdong Wu, Haibing Xie, Binbin Yao, Yanming Chen, David M Irwin, Dan Huang, Jin Xu, Nelson LS Tang, and Yaping Zhang. Ambient temperature is a strong selective factor influencing human development and immunity. *Genomics, Proteomics & Bioinformatics*, 2020.

[78] Martin Kuhlwilm and Cedric Boeckx. A catalog of single nucleotide changes distinguishing modern humans from archaic hominins. *Scientific Reports*, 9(1):1–14, 2019.

[79] Maria Carolina Matos, Ana Pinheiro, José Melo-Ferreira, Randall S Davis, and Pedro José Esteves. Evolution of fc receptor-like scavenger in mammals. *Frontiers in immunology*, 11:3937, 2021.

[80] Arianna Landini, Shaobo Yu, Guido Alberto Gnecchi-Ruscone, Paolo Abondio, Claudia Ojeda-Granados, Stefania Sarno, Sara De Fanti, Davide Gentilini, Anna Maria Di Blasio, Hanjun Jin, et al. Genomic adaptations to cereal-based diets contribute to mitigate metabolic risk in some human populations of east asian ancestry. *Evolutionary applications*, 14(2):297–313, 2021.

[81] Andreas Wagner. Rapid detection of positive selection in genes and genomes through variation clusters. *Genetics*, 176(4):2451–2463, 2007.

[82] Sharon R Grossman, Ilya Shylakhter, Elinor K Karlsson, Elizabeth H Byrne, Shannon Morales, Gabriel Frieden, Elizabeth Hostetter, Elaine Angelino, Manuel Garber, Or Zuk, et al. A composite of multiple signals distinguishes causal variants in regions of positive selection. *Science*, 327(5967):883–886, 2010.

[83] Alba Refoyo-Martínez, Rute R da Fonseca, Katrín Halldórsdóttir, Einar Árnason, Thomas Mailund, and Fernando Racimo. Identifying loci under positive selection in complex population histories. *Genome research*, 29(9):1506–1520, 2019.

[84] Benjamin F Voight, Sridhar Kudaravalli, Xiaoquan Wen, and Jonathan K Pritchard. A map of recent positive selection in the human genome. *PLoS biology*, 4(3):e72, 2006.

[85] Sergey Kryazhimskiy and Joshua B Plotkin. The population genetics of dn/ds. *PLoS genetics*, 4(12):e1000304, 2008.

[86] Anna Ferrer-Admetlla, Mason Liang, Thorfinn Korneliussen, and Rasmus Nielsen. On detecting incomplete soft or hard selective sweeps using haplotype structure. *Molecular biology and evolution*, 31(5):1275–1291, 2014.