Deep learning scans for selective sweeps using RAiSD-AI

Nikolaos Alachiotis[1]*, Prodromos Papadopoulos[2], Hanqing Zhao[1], and Pavlos Pavlidis[2]

[1] University of Twente, The Netherlands

[2] University of Crete, Greece

Abstract. Recent advances in method and software development for selective sweep detection focus on using deep learning to improve detection performance. However, the adoption of deep learning in real-world analyses is slow, hindered by the lack of reusable tools that alleviate the interdisciplinary friction of integrating such methods for practical deployment. This chapter walks the reader through the basics of using RAiSD-AI for selective sweep analysis. RAiSD-AI is a recently introduced tool that can train and test Convolutional Neural Networks (CNNs), and subsequently deploy them for genomic scans for selective sweeps.

Keywords: Selective Sweep, Deep Learning, Convolutional Neural Network

Introduction

The detection of natural selection in genomic data is a key objective in population genetics, providing insights into evolutionary processes and the genetic basis of adaptation. Traditional methods for identifying selective sweeps often rely on summary statistics that capture specific data features, such as changes in allele

* Corresponding author: n.alachiotis@utwente.nl

frequencies [1], haplotype structures [2], or patterns of linkage disequilibrium [3]. However, designing appropriate summary statistics can be challenging, especially when dealing with complex demographic histories and heterogeneous genomic landscapes. Alternatively, model-based approaches, such as SweepFinder [4] and SweeD [5], infer selective sweep parameters using a maximum-likelihood framework. In simple demographic scenarios, such as recent and mild bottlenecks, the performance of such tools is high. In more complex demographic models, however, these model-based approaches have lower efficiency because the underlying mathematical models assume simpler evolutionary models (e.g., constant population).

Recent advances in deep learning have introduced powerful new methods for inferring selective events directly from genetic data. Specifically, Convolutional Neural Networks (CNNs) [6], initially developed for image recognition, have demonstrated strong performance in detecting subtle patterns indicative of selective sweeps. By treating genotype matrices or summary statistic profiles as input "images", CNNs can learn hierarchical representations that integrate local and global signals of selection. This chapter presents a practical, step-by-step workflow for using RAiSD-AI (Raised Accuracy in Sweep Detection using Artificial Intelligence), covering the training and testing of a CNN, and its application to scan genomic data.

RAiSD-AI abstracts away the complexity of pre- and post-processing data for CNNs, allowing users to perform genomic scans for selective sweeps without requiring in-depth knowledge of deep learning theory or frameworks, such as TensorFlow [7] or PyTorch [8]. RAiSD-AI seamlessly integrates CNN-based processing with widely used genomic scan algorithms, such as linear grid [4], and implements advanced methods for processing large-scale datasets with minimal memory requirements, such as out-of-core processing [9]. This chapter provides

a series of command-line examples that demonstrate how to use RAiSD-AI to train a CNN, optionally evaluate its classification performance, and apply the trained model to perform genomic scans. The source code of RAiSD-AI can be downloaded from the RAiSD-AI repository, while additional scripts used in this chapter are available at the RAiSD-AI Guide repository.

Environment and compilation

RAiSD-AI is implemented in C, and uses PyTorch for CNN training and inference. To streamline the installation process, which involves compiling the source code, the RAiSD-AI Guide repository contains a script named `install.sh`. This script installs all required dependencies, compiles RAiSD-AI, and adds the executable to the system's `PATH`, allowing it to be executed from any directory. For consistency and reproducibility, the setup uses a custom Anaconda environment based on Python 3.8. The full list of required packages is defined in the `raisd-ai.yml` file, which is also included in the repository and created by the `install.sh` script. The bash commands to download the repository, compile RAiSD-AI, and create and activate the Anaconda environment are provided below.

```
# Clone the repository and enter the directory
git clone https://github.com/StatisticalPopulationGenomics-2ndEd/RAiSD-AI
cd RAiSD-AI/

# Run the installation script
bash install.sh

# Activate the RAiSD-AI conda environment
conda activate raisd-ai
```

The following command automates all the steps presented and described in this chapter; it is advisable to go through the complete tutorial step by step for a comprehensive understanding of the basic functionalities of RAiSD-AI.

```
bash quick_run.sh
```

Simulate genetic data for training (Step 1)

RAiSD-AI relies on a CNN classifier. CNNs need to be trained before used either for classification or for scanning. Given a dataset to be scanned for selective sweeps, its demographic parameters need to be determined first. Various methods can be used to infer the demographic model of a population or a group of populations. For example, dadi [10, 11] exploits the Site Frequency Spectrum (SFS) using a diffusion-based approximation over multiple populations. Sequentially Markovian Coalescent (SMC) methods, such as PSMC [12] and MSMC2 [13], use coalescent approximations to infer population sizes via heterozygosity. Approximate Bayesian Computation (ABC) [14], such as the `abc` package [15] in R, or `pyABC` [16] in Python exploit the distribution of summary statistics values. Then simulations are performed under the obtained demographic parameters using tools such as msprime [17] or SLiM [18].

Since the goal is to detect selective sweeps, the CNN is used to distinguish between regions representative of a selective sweep and neutral regions. Therefore, the complete training set must comprise two groups of simulations, both under the same demographic model, i.e., neutral simulations and simulations with a selective sweep. The selective sweep needs to be simulated at the same position in all different training simulations. This is needed because RAiSD-AI receives one sweep position as an input parameter to properly extract windows of Single Nucleotide Polymorphism (SNP) data from every simulation and use them for training the CNN in the next step.

We briefly describe two ways to generate simulated data under a specific demographic scenario; this step is not necessary for following this RAiSD-AI guide as the next section provides a link to download example datasets. The first approach uses ABC through the Python library `pyABC` [16], while the second one relies on `stdpopsim` [19][20], which generates synthetic data based on verified demographic models found in literature.

The pyABC framework facilitates the estimation of parameter values that best fit the data by sampling from a simulator instead of directly computing likelihood values. In this iterative procedure, parameter sets are initially proposed and used to generate simulations. The similarity between each simulation and the observed data is assessed, and parameter values that fit the data are inferred. Through parallel execution and iterative refinement of saved parameter sets across successive cycles, the workflow enables systematic convergence toward the target posterior distribution. An additional enhancement involves incorporating prior information about the population under study, which improves the accuracy of inferential results and enables the generation of synthetic datasets that more closely reflect the characteristics of real data. After population parameter estimation, these parameters are used as input arguments to a simulator (e.g., msprime [17]) to generate tailored synthetic training datasets with and without a selective sweep.

Another way to generate simulations for training relies on the standardized population simulation library stdpopsim [19, 20], which allows researchers to generate realistic synthetic datasets derived from well-established demographic models for a variety of species. stdpopsim deploys SLiM4 [18] for forward-time simulations, and msprime [17] for coalescent-based simulations. Based on the properties of the empirical data, the user can choose the most suitable demographic model from the predefined options available in the `stdpopsim` library

to generate neutral simulated datasets and simulated datasets with a selective sweep in the Variant Call Format (VCF) format, which can be directly parsed by RAiSD-AI.

We describe a basic approach to use `stdpopsim` with `tskit`[21, 22, 23] to create ms-format files that can be used to train/test a RAiSD-AI model. Neutral data are simulated with `msprime`, while selective sweep data are simulated with `SLiM`, which are the engines used by `stdpopsim`. Before examining the simulation code, it is important to understand the `collapse_ts` function and why it is necessary. When `stdpopsim` generates tree sequences, the site and mutation data may contain complex ancestral and derived state information that is not always in the expected format for ms-format output. The following `collapse_ts` function standardizes this format by converting all ancestral states to "0" and all derived states to "1", ensuring compatibility with downstream analysis tools that expect this binary representation.

```
def collapse_ts(ts):
    t = ts.dump_tables()
    t.sites.packset_ancestral_state(["0"] * len(t.sites))
    t.mutations.packset_derived_state(["1"] * len(t.mutations))
    return t.tree_sequence()
```

The neutral simulations provide baseline data without selection and serve as negative examples for detecting selective sweeps. The following code generates 100 neutral replicates using the `msprime` simulator.

```
import stdpopsim, tskit

species = stdpopsim.get_species("HomSap")
model = species.get_demographic_model("OutOfAfrica_2T12")
```

```
contig = species.get_contig("chr22", right=1e5)
with open("neutral.ms", "w") as f:
    for i in range(100):
        ts = stdpopsim.get_engine("msprime").simulate(
            model, contig, {"AFR": 100})
        tskit.write_ms(collapse_ts(ts), f)
```

This generates 100 replicates of a 100,000 bp region from chromosome 22, each with 100 samples from the African (AFR) population under the Out-of-Africa demographic model [24].

The selective sweep simulations provide positive examples where a beneficial mutation has increased in frequency. The following code generates 100 simulations with a selective sweep centered at position 50,000 bp using the `SLiM` simulator.

```
contig_sweep = species.get_contig("chr22", right=1e5)
contig_sweep.add_single_site(id="sweep", coordinate=5e4)
sweep = stdpopsim.selective_sweep(
    "sweep", "AFR", mutation_generation_ago=2000, selection_coeff=0.1
)
with open("selsweep.ms", "w") as f:
    for i in range(100):
        ts = stdpopsim.get_engine("slim").simulate(
            model,
            contig_sweep,
            {"AFR": 100},
            extended_events=sweep,
            slim_scaling_factor=10)
```

```
        tskit.write_ms(collapse_ts(ts), f)
```

This generates 100 replicates with the same regional and population parameters as the neutral simulations, but includes a selective sweep with a selection coefficient of 0.1 that arose 2,000 generations ago.

Prepare data for CNN training (Step 2)

Once the simulations are generated, RAiSD-AI can extract relevant information from them to create input data for the CNN, and then use PyTorch to train the CNN model. Because the training process depends on known simulation parameters, it must be performed using simulated data to ensure that the location of the selective sweep is precisely defined. We provide an example dataset that will be used for training in this chapter; it can be downloaded with the following command line:

```
wget https://zenodo.org/records/18841026/files/Mild-bottleneck-training.tar.xz
tar -xvJf Mild-bottleneck-training.tar.xz
```

This dataset simulates a recent and mild bottleneck scenario, and includes 1,000 neutral simulations and 1,000 selective sweep simulations. The sample size is 128 sequences. It is generally recommended to use a large training dataset size that includes thousands of training data points per CNN class. A training data point is a single element from the training dataset used to train the CNN, for example an image representing a window of 100 SNPs. By default, RAiSD-AI extracts a single training data point from each simulation.

The following commands parse the two input dataset files, i.e., the `neutral.ms` file that contains $1,000$ neutral simulations, and the `selsweep.ms` file that contains $1,000$ simulations with a selective sweep. RAiSD-AI is informed about the

length of the simulated region through `-L`, and extracts one training data point from position $50,000$ in this example, from each of the $1,000$ simulations.

```
RAiSD-AI -n TrainingData
         -I Mild-bottleneck-training/neutral.ms
         -w 128
         -L 100000
         -its 50000
         -op IMG-GEN
         -icl neutralTR
         -bin
         -typ 1
         -frm

RAiSD-AI -n TrainingData
         -I Mild-bottleneck-training/selsweep.ms
         -w 128
         -L 100000
         -its 50000
         -op IMG-GEN
         -icl sweepTR
         -bin
         -typ 1
```

These commands receive the same run name (`TrainingData`) through `-n`. The first command line additionally includes the `-frm` (force remove) parameter. Because of the same run name, both commands will generate output in the same directory (`RAiSD_Images.TrainingData`) but in different subfolders (`neutralTR` and `sweepTR`) as both outputs are going to be used together to train the CNN.

The `-frm` parameter indicates the first class to be generated (and therefore removes the output directory if it already exists before generating new data). The second command line (which does not include the `-frm` parameter) first checks the correctness and compatibility of the parameters with the previous run. For example, the data type is set to `-bin`, which indicates that the generated files will be binary. Also, the data type (`-typ`) must be the same in both runs for the second command to run correctly. The data type indicates what information is encoded in the training data points ("`-typ 1`" indicates that allele frequencies and relative SNP distances will be encoded and used for training). These commands generate training data points in binary format, and each data point represents a SNP window of 128 SNPs (specified with `-w`). A summary of all the parameters used in the two command lines is provided below.

- The `-n` parameter specifies the run name
- The `-I` parameter specifies the input file
- The `-L` parameter specifies the length of the simulated region
- The `-w` parameter indicates the width of the window
- The `-its` parameter points to the extraction location for training data points per simulation
- The `-op` parameter (set to `IMG-GEN` in these runs) indicates the operating mode of RAiSD-AI, i.e., the run will generate training data points for later use in CNN model generation (training)
- the `-icl` parameter provides the name of the subdirectory that is generated for each class in the main RAiSD_Images.runname directory
- the `-typ` parameter is set to 1 to encode derived allele frequencies along with relative SNP positions

A large and balanced training dataset—consisting of an equal number of data points for each class—is generally recommended to enable the CNN to generalize

well and minimize the risk of overfitting. RAiSD-AI can extract multiple training data points from a single simulation. However, it is generally recommended to use a larger number of independent simulations as well. Based on extensive experiments across diverse demographic models, we recommend to use between $5,000$ and $10,000$ simulations (i.e., training data points) per class.

Training the CNN classifier (Step 3)

After generating the necessary training data points, RAiSD-AI can be used to train the CNN. The tool contains CNN implementations of various CNN architectures, such as SweepNet [25] and FAST-NN [26], which are designed for classification tasks, and FASTER-NN [27], which is designed for detection tasks. The default architecture is FASTER-NN, which is also used in the example we present in this chapter. The following command line trains FASTER-NN:

```
RAiSD-AI -n Model
         -I RAiSD_Images.TrainingData
         -op MDL-GEN
         -e 100
```

The CNN will be trained for 100 epochs (`-e` parameter) using the data in directory `RAiSD_Images.TrainingData` (training data generated in the previous step) as input (`-I` parameter). Notice that the mode of operation for RAiSD-AI in this command line is set to `MDL-GEN` for generating a CNN model (training). The user can specify which CNN architecture to use with `-arc` using one of the following arguments: SweepNet, FAST-NN, or FASTER-NN. Because not all combinations of training data point configurations are supported by every CNN architecture, it is recommended to check the in-tool help menu of RAiSD-AI (`-h` command-line parameter) for compatibility options.

To ensure effective training of the CNN, it is recommended to use 50-100 training epochs, based on empirical observations. RAiSD-AI incorporates an early-stopping mechanism that does not terminate training before the specified number of epochs is reached, but retains the model achieving the highest validation accuracy across all epochs. This best-performing model is saved in a folder named `RAiSD_Model.Model`, while training details—including validation accuracy progression over epochs—are recorded in the file `RAiSD_Info.Model`. An example of the output of the run that shows the result of training after the first 4 epochs, and when the best model is saved, is shown in Figure 1.

Using the CNN classifier for classification (Step 4)

When training is finished, the trained CNN can be used by RAiSD-AI for classification or, in combination with a scan algorithm (e.g., linear grid), for detection. Since RAiSD-AI is primarily designed for detection, employing the CNN classifier only for classification serves as a way to test how the classifier performs on previously unseen data. In this step, the classifier from Step 3 is tested using new data that were not involved in the training process. The test dataset we use in this example contains $1,000$ neutral simulations and $1,000$ simulations with a selective sweep, both generated under the same demographic-model parameters as the training data. The following commands can be used to download the test dataset.

```
wget https://zenodo.org/records/18841026/files/Mild-bottleneck-testing.tar.xz
tar -xvJf Mild-bottleneck-testing.tar.xz
```

The provided files (ms file format) for both the neutral and the selective sweep test datasets are first converted into data points using RAiSD-AI in the same way as training data points were created in Step 2. The following command was used for the neutral class:

```
RAiSD-AI -n TestData
         -I Mild-bottleneck-testing/neutral.ms
         -w 128
         -L 100000
         -its 50000
         -op IMG-GEN
         -icl neutralTE
         -bin
         -typ 1
         -frm
```

The following command was used for the selective-sweep class:

```
RAiSD-AI -n TestData
         -I Mild-bottleneck-testing/selsweep.ms
         -w 128
         -L 100000
         -its 50000
         -op IMG-GEN
         -icl sweepTE
         -bin
         -typ 1
```

It should be noted that the same image-generation configuration as for the training data points should be used (`-w`, `-bin`, and `-typ`). RAiSD-AI performs internal checks for compatibility across runs and notifies the user for parameter mismatches. The output of these two commands is stored in a directory named `RAiSD_Images.TestData`. After generating the test data points, the CNN classifier can be tested with the following command, which deploys the CNN for inference:

```
RAiSD-AI -n Test
         -mdl RAiSD_Model.Model
         -I RAiSD_Images.TestData
         -op MDL-TST
         -clp 2 neutralTR=neutralTE sweepTR=sweepTE
```

This command line allows the user to specify a name for the run using `-n`, specify the path to the directory where the CNN model is stored via `-mdl`, provide the path to the input test data with `-I`, and define the structure of the inference/classification tests to gather performance statistics through `-clp`. Notice that the output of previous RAiSD-AI calls (`RAiSD_Model.Model` and `RAiSD_Images.TestData`) is provided directly as input here, which simplifies the test process and reduces the risk of mistakes.

In this run, the mode of operation for RAiSD-AI is set to `MDL-TST` to indicate that the CNN will only be used for classification/inference. The `-clp` parameter provides the number of classes to be tested, followed by an equal number of pairings (using '=') between the class labels and the corresponding folder names in the `RAiSD_Images.TestData` directory. The names of the individual folders in the directory used for training the CNN model are used as the names of the model classes, i.e., `neutralTR` and `sweepTR`. The respective folders in the directory used for testing are `neutralTE` and `sweepTE`.

An example of the test results saved in the `RAiSD_Info.Test` file is shown in Figure 2. The results contain precision, recall, and f1-scores for each class, as well as the overall classification accuracy of the CNN model. It should be noted that the example dataset represents a rather straightforward case for selective sweep detection, which explains the perfect classification performance results. In general, lower classification numbers are obtained for more challenging demographic models, such as severe bottlenecks. Based on a wide range of experimental runs

with simulations with mild bottlenecks, severe bottlenecks, recent migration, ancient migration, and recombination hotspots with low and high intensity, we have observed that the CNNs implemented in RAiSD-AI achieve classification accuracies that typically exceed 90%.

Using the CNN classifier for genomic scans (Step 5)

Given a trained CNN model (stored in directory `RAiSD_Model.Model` from Step 3), RAiSD-AI can be used to scan a dataset for selective sweeps. In this step, we demonstrate the use of RAiSD-AI for CNN-based genomic scans by first scanning a neutral dataset to obtain cut-off thresholds; they can either be applied to score distributions obtained from scans on empirical data, to determine candidate sweep regions, or to score distributions obtained from simulated datasets with selective sweeps, to calculate detection accuracy and sensitivity (true positive rate). For convenience, we provide the command lines to perform a statistical analysis to assess detection accuracy and sensitivity using the same simulated datasets that were previously used in Step 4 to assess classification accuracy. It is important to note that RAiSD-AI computes the μ statistic [28] in parallel with applying the CNN. Moreover, RAiSD-AI can be used without any CNN, relying solely on the μ statistic [28], thus eliminating the need for model training.

Cut-off thresholds based on neutral simulations

The following command scans the `neutral.ms` test dataset using a linear grid to obtain cut-off thresholds based on neutral simulations.

```
RAiSD-AI -n neutral
         -mdl RAiSD_Model.Model
         -op SWP-SCN
```

```
-I Mild-bottleneck-testing/neutral.ms
-L 100000
-k 0.05
-G 100
-pci 1 1
```

The mode of operation (`-op`) is set to `SWP-SCN` (scanning), and the size of the linear grid is defined by `-G` (grid). This command creates a linear grid of 100 equidistant positions along the region to be scanned. The length of the region is defined by `-L`, and is only required with simulated datasets in ms-like format. When VCF files are used, RAiSD-AI obtains the region length from the input file. Because the classification scores of the CNN model are used to create a score distribution across the linear grid, the `-pci` parameter is required to provide the number of "positive" classes to be used (the first '1' after `-pci` in this example) followed by their indices in the CNN model (the second '1' after `-pci` in this example). For all CNN architectures currently implemented in RAiSD-AI, the number of positive classes is '1', while the index of the positive class can be found in the `classLabels.txt` file that is generated in the `RAiSD_Model.Model` directory after training. An example of the content in this file is shown below, where the index of the positive class (`sweepTR`) is the number in parentheses after the class name.

```
***DO_NOT_REMOVE_OR_EDIT_THIS_FILE***
2
neutralTR (0)
sweepTR (1)
***DO_NOT_REMOVE_OR_EDIT_THIS_FILE***
```

To direct RAiSD-AI to report cut-off thresholds based on the neutral simulations, the `-k` parameter is needed. It provides the false positive rate (e.g., 0.05

in this example) to report the corresponding reported score after sorting the highest-score sweep locations reported for all simulations in the input dataset. An example of the relevant output is shown below.

```
FPR threshold for FPR 0.050000 (sorted vector index 50)

muVar               : 1.57210 (min 1.22456, max 1.81900)
muSFS               : 2.11927 (min 1.39872, max 2.88221)
muLD                : 3.46960 (min 2.63990, max 4.01455)
mu                  : 5.54402 (min 3.36891, max 7.27077)
sweepTR             : 0.04163 (min 0.00032, max 0.74320)
```

RAiSD-AI reports cut-off thresholds for the μ statistic (`mu`), each of its three factors separately (`muVar`, `muSFS`, and `muLD`), and the CNN inference scores for the positive class (`sweepTR`).

Scan empirical or simulated data

The following command line scans the `selsweep.ms` test dataset using a linear grid of the same size as in the previous scan.

```
RAiSD-AI -n sweep
         -mdl RAiSD_Model.Model
         -op SWP-SCN
         -I Mild-bottleneck-testing/selsweep.ms
         -L 100000
         -T 50000
         -d 2500
         -G 100
```

```
-pci 1 1
-l 5 var=1.57210 sfs=2.11927 ld=3.46960 mu=5.54402 pcl0=0.04163
```

The cut-off thresholds reported in the previous scan (`neutral.ms`) are passed as input arguments through the `-l` command-line flag, based on which, RAiSD-AI reports sensitivity (true positive rate) for each of the statistics and the CNN scores. The general syntax of `-l` is the following: "-l Number_of_Thresholds label1=thres1 label2=thres2 ...", where label is one of the following: "var", "sfs", "ld", "mu", "pcl0", "pcl1". In addition to the `-l` parameter to report TPRs, two additional parameters can be optionally used, i.e., `-T` and `-d`. The `-T` flag specifies the selection target in base pairs and calculates the average distance (across all datasets in the input file) between the selection target and the reported locations, while the `-d` flag provides a maximum distance from the selection target (in base pairs) to calculate success rates. Using `-T`, `-d`, and `-l` is mainly meaningful when simulated data with selective sweeps are scanned, to obtain detection performance numbers given the known selective sweep locations. When empirical data is scanned, the cut-off thresholds can be applied to the score distributions in a post-processing step, to determine candidate selective sweep regions.

An example of the output generated when using `-T`, `-d`, and `-l` is provided in Figure 3. These detection-performance statistics are calculated based on the list of highest scores and respective positions over all simulations. Because the cut-off thresholds were obtained with "`-k 0.05`", the reported TPR scores correspond to FPR of 5%.

RAiSD-AI can additionally generate plots for all statistics and CNN scores separately for each dataset/simulation scanned. This can be achieved by adding the `-P` flag (command lines not shown). Note that this will slow down overall execution because of the plot generation and saving to the disk overhead, with the execution time increase being more profound when RAiSD-AI only uses the

μ statistic and not a CNN; RAiSD-AI execution is equivalent to RAiSD when a CNN is not used. Examples of the generated plots are provided in Figures 4A and 4B for the first neutral and selective-sweep simulations in the test dataset. Figure 4A shows the μ statistic score distributions, while Figure 4B plots the CNN inference scores for the `sweepTR` class. The test dataset simulates a selective sweep at the center of the region, which is shown in both the μ statistic and the CNN inference plots. Notice also the profound difference in score ranges between the scan of the neutral dataset and the scan of the selective-sweep dataset, with the μ statistic assuming values between 1.0 and 4.5 in the neutral scan, and up to 21 in the selective-sweep scan. The CNN scores are all close to 0.0 in the neutral scan, while a distinct region with scores as high as 1.0 is observed in the selective-sweep scan.

Conclusion

This chapter provided a comprehensive, step-by-step guide to use `RAiSD-AI` for training CNNs, and apply the trained models to detect selective sweeps in genomic data. By following the outlined workflow—Steps 2 through 5—users can train a `FASTER-NN` CNN using simulated datasets (e.g., a mild bottleneck scenario) and subsequently apply the model to classification and detection tasks in population-genomic datasets.

`RAiSD-AI` offers flexible parameterization, enabling users to tailor deep learning models for selective sweep detection under diverse evolutionary scenarios. Based on prior empirical work, we recommend the following best practices for optimizing `RAiSD-AI` model performance:

- Use **large and balanced training datasets** ($\geq$ 5,000 training samples per class) to reduce the risk of overfitting.

- Train models for **several epochs** ($\geq$ 25) to ensure adequate learning and convergence.
- For sweep detection based on the μ-statistic alone, use **small window sizes** ($\leq$ 50 SNPs).
- For CNN-based scans, employ **larger windows** ($\geq$ 250 SNPs) to enable the CNN to capture broader genomic context.

As with most deep learning approaches for selective sweep detection, this chapter demonstrated the use of fully simulated datasets to train the `RAiSD-AI` CNN classifier. Simulation-based training offers control over evolutionary parameters, such as the sweep location, the demographic history, and the sample size, but entails a critical limitation as CNNs trained exclusively on synthetic data may learn simulator-specific artifacts rather than biologically meaningful patterns. As noted by Trost et al. [29], this can lead to severe "out-of-distribution" effects, where models perform well on synthetic datasets but not on empirical data. Because real demographic histories are complex and often unknown, careful tuning and iterative exploration of `RAiSD-AI` settings are essential to develop models with strong generalization capabilities. Future development will focus on hybrid training strategies that incorporate both simulated and empirical data to enable `RAiSD-AI` and similar tools to yield robust, generalizable insights into the genomic signatures of natural selection.

References

[1] J. C. Fay et al. “Hitchhiking under positive Darwinian selection”. In: *Genetics* 155.3 (2000), pp. 1405–1413.

[2] A. Ferrer-Admetlla et al. “On detecting incomplete soft or hard selective sweeps using haplotype structure”. In: *Molecular biology and evolution* 31.5 (2014), pp. 1275–1291.

[3] Y. Kim et al. “Linkage disequilibrium as a signature of selective sweeps”. In: *Genetics* 167.3 (2004), pp. 1513–1524.

[4] R. Nielsen et al. “Genomic scans for selective sweeps using SNP data”. In: *Genome research* 15.11 (2005), pp. 1566–1575.

[5] P. Pavlidis et al. “SweeD: likelihood-based detection of selective sweeps in thousands of genomes”. In: *Molecular biology and evolution* 30.9 (2013), pp. 2224–2234.

[6] Y. LeCun et al. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[7] M. Abadi et al. “Tensorflow: a system for large-scale machine learning.” In: *Osdi*. Vol. 16. 2016. Savannah, GA, USA. 2016, pp. 265–283.

[8] M. Joseph. “Pytorch tabular: A framework for deep learning with tabular data”. In: *arXiv preprint arXiv:2104.13638* (2021).

[9] M. Drozdowski et al. “Out-of-core divisible load processing”. In: *IEEE Transactions on Parallel and Distributed Systems* 14.10 (2003), pp. 1048–1056.

[10] R. N. Gutenkunst et al. “Inferring the joint demographic history of multiple populations from multidimensional SNP frequency data”. In: *PLoS genetics* 5.10 (2009), e1000695.

[11] R. N. Gutenkunst. “dadi. CUDA: Accelerating population genetics inference with graphics processing units”. In: *Molecular biology and evolution* 38.5 (2021), pp. 2177–2178.

[12] H. Li et al. “Inference of human population history from individual whole-genome sequences”. In: *Nature* 475.7357 (2011), pp. 493–496.

[13] S. Schiffels et al. “Inferring human population size and separation history from multiple genome sequences”. In: *Nature genetics* 46.8 (2014), pp. 919–925.

[14] M. A. Beaumont et al. “Approximate Bayesian computation in population genetics”. In: *Genetics* 162.4 (2002), pp. 2025–2035.

[15] K. Csilléry et al. “Approximate Bayesian computation (ABC) in practice”. In: *Trends in ecology & evolution* 25.7 (2010), pp. 410–418.

[16] Y. Schälte et al. “pyABC: Efficient and robust easy-to-use approximate Bayesian computation”. In: *Journal of Open Source Software* 7.74 (2022), p. 4304. DOI: 10.21105/joss.04304. URL: https://doi.org/10.21105/joss.04304.

[17] J. Kelleher et al. “Efficient coalescent simulation and genealogical analysis for large sample sizes”. In: *PLoS computational biology* 12.5 (2016), e1004842.

[18] B. C. Haller et al. “SLiM 3: Forward Genetic Simulations Beyond the Wright–Fisher Model”. In: *Molecular Biology and Evolution* 36.3 (Jan. 2019), pp. 632–637. ISSN: 0737-4038. DOI: 10.1093/molbev/msy228. eprint: https://academic.oup.com/mbe/article-pdf/36/3/632/27980602/msy228.pdf. URL: https://doi.org/10.1093/molbev/msy228.

[19] J. R. Adrion et al. “A community-maintained standard library of population genetic models”. In: *eLife* 9 (June 2020). Ed. by G. Coop et al., e54967. ISSN: 2050-084X. DOI: 10.7554/eLife.54967. URL: https://doi.org/10.7554/eLife.54967.

[20] M. E. Lauterbur et al. “Expanding the stdpopsim species catalog, and lessons learned for realistic genome simulations”. In: *eLife* 12 (May 2023),

RP84874. DOI: 10.7554/elife.84874. URL: https://doi.org/10.7554/elife.84874.

[21] Y. Wong et al. "A general and efficient representation of ancestral recombination graphs". In: *Genetics* 228.1 (2024), iyae100. DOI: 10.1093/genetics/iyae100.

[22] J. Kelleher et al. "Efficient coalescent simulation and genealogical analysis for large sample sizes". In: *PLoS Computational Biology* 12.5 (2016), e1004842.

[23] P. Ralph et al. "Efficiently Summarizing Relationships in Large Samples: A General Duality Between Statistics of Genealogies and Genomes". In: *Genetics* 215.3 (2020), pp. 779–797. DOI: 10.1534/genetics.120.303253.

[24] J. A. Tennessen et al. "Evolution and functional impact of rare coding variation from deep sequencing of human exomes". In: *science* 337.6090 (2012), pp. 64–69.

[25] H. Zhao et al. "SweepNet: a lightweight CNN architecture for the classification of adaptive genomic regions". In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. 2023, pp. 1–10.

[26] S. van den Belt et al. "Scalable CNN-based classification of selective sweeps using derived allele frequencies". In: *Bioinformatics* 40.Supplement_2 (2024), pp. ii29–ii36.

[27] S. van den Belt et al. "Fast and accurate deep learning scans for signatures of natural selection in genomes using FASTER-NN". In: *Communications Biology* 8.1 (2025), p. 58.

[28] N. Alachiotis et al. "RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors". In: *Communications biology* 1.1 (2018), p. 79.

[29] J. Trost et al. “Simulations of sequence evolution: how (un) realistic they are and why”. In: *Molecular biology and evolution* 41.1 (2024), msad277.

```
CNN training (FASTER-NN) ...


Epoch 0: loss 3.9974163943593988, acc 0.8729411959648132, val_acc

0.9966666666666667 [SAVING BEST MODEL, ACCURACY: 0.9966666666666667]

Epoch 1: loss 0.15314856308136118, acc 0.9958823323249817, val_acc 1.0

[SAVING BEST MODEL, ACCURACY: 1.0]

Epoch 2: loss 0.025837054590640816, acc 0.9994117617607117, val_acc

0.9966666666666667

Epoch 3: loss 0.005785707562722564, acc 1.0, val_acc 1.0
```

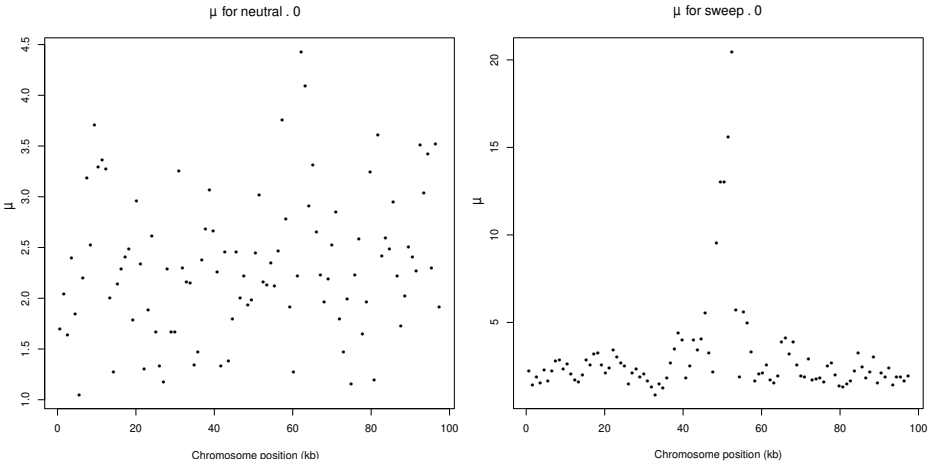
Fig. 1: CNN training (FASTER-NN) output. For each epoch, RAiSD-AI prints the validation loss, the training accurary, and the validation accuracy. It is also shown when the best model (in terms of validation accuracy) is saved.

```
True class: neutralTR | Test data: RAiSD_Images.TestData/neutralTE (1000
samples/images) | Predicted class: 1000 neutralTR - 0 sweepTR
True class: sweepTR | Test data: RAiSD_Images.TestData/sweepTE (1000
samples/images) | Predicted class: 0 neutralTR - 1000 sweepTR


Precision for class neutralTR : 100.000 %
Precision for class sweepTR : 100.000 %

Recall for class neutralTR : 100.000 %
Recall for class sweepTR : 100.000 %

F1-score for class neutralTR : 100.000 %
F1-score for class sweepTR : 100.000 %

Accuracy : 100.000 %
```

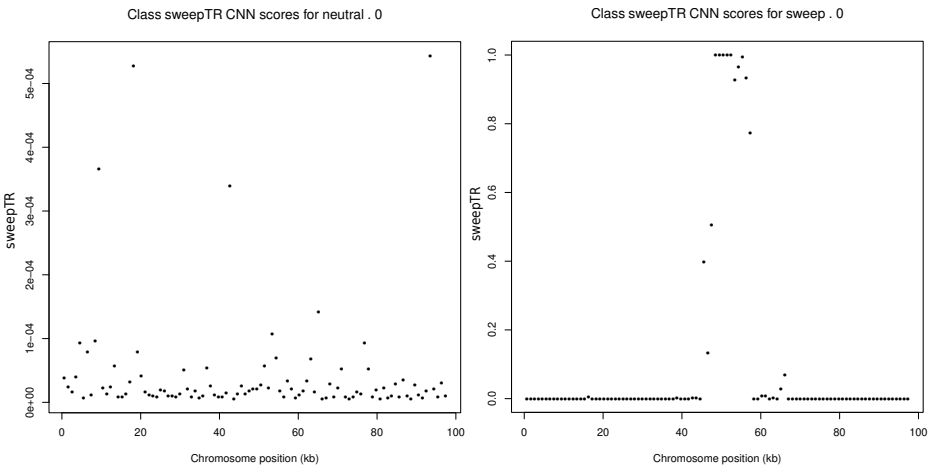
Fig. 2: CNN testing output. RAiSD-AI reports precision, recall, and F1-score for each class, as well as the overall accuracy of the CNN classifier. The suffix "TR" denotes the class name used during training, while "TE" refers to the corresponding class in the test dataset.

```
Average distance from target site position: 50000

muVar : 897.9

muSFS : 3762.4

muLD : 10306.7

mu : 1000.8

sweepTR : 803.1

Success rate based on max distance of 2500 from target site:

muVar : 0.97800

muSFS : 0.71400

muLD : 0.26700

mu : 0.94400

sweepTR : 0.97700

TPR score(s) for FPR threshold(s):

muVar : 1.00000 (FPR threshold 1.57210)

muSFS : 0.84400 (FPR threshold 2.11927)

muLD : 0.49700 (FPR threshold 3.46960)

mu : 1.00000 (FPR threshold 5.54402)

sweepTR : 1.00000 (FPR threshold 0.04163)
```

Fig. 3: RAiSD-AI detection performance. RAiSD-AI reports metrics such as the average distance from the selection target, the success rate based on a maximum distance from the target site, and the true positive rate based on a false positive rate cut-off threshold.



(a) μ statistic score distributions of a RAiSD-AI scan of a neutral dataset (left) and a dataset with a selective sweep at the center of the simulated region (right). The '0' in the plot headers is the simulation index (numbering starts from 0).



(b) CNN inference scores of RAiSD-AI scans of the same neutral dataset (left) and the same dataset with a selective sweep at the center of the simulated region (right). Recall that "sweepTR" is the name of the training class that corresponds to the sweep data.

Fig. 4: Plots of score distributions of the μ statistic (a) and CNN scores (b) generated by RAiSD-AI when the `-P` parameter is used.