Exploration of machine learning methods for genomic selection in cattle

Auteur : Hervers, Florent
Promoteur(s) : Geurts, Pierre; Druet, Tom
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2024-2025
URI/URL : https://github.com/Florent-Hervers/master-thesis-code; http://hdl.handle.net/2268.2/23245

UNIVERSITE DE LIEGE
FACULTE DES SCIENCES APPLIQUEES

Exploration of machine learning methods for genomic selection in cattle

Hervers Florent

Thesis presented to obtain the degree of :
Master of Science in Computer Science Engineering

Thesis supervisors:
Geurts Pierre
Druet Tom

Academic year: **2024 - 2025**

Acknowledgments

I would like to thank my supervisors, Pierre Geurts and Tom Druet, for enabling me to work on this subject. I am very thankful for their availability, patience, analysis, and suggestions that they have made during the experiments and redaction. Their help was a major factor in the success of this thesis, thanks to their contributions. I would like to thank Vân Anh Huynh-Thu for taking the time to follow this thesis. I am grateful for the suggestions made during the meetings and the report writing that helped me achieve better work.

I also want to express my gratitude to Frederik Van De Velde and Ariane Wuidard for taking the time to read this report thoroughly. Their valuable feedback enabled me to make this thesis more complete and understandable.

I thank the Walloon breeding association for enabling the use of their dataset during this thesis.

Finally, I would like to thank my family and my friends for their support during this thesis and during the last five years in general. I would not have been able to achieve everything without their help.

Summary

Genomic selection is a method developed to help breeders select the best parents to have a progeny that exhibits good targeted characteristics. This selection is performed based on the genotype of the animals. The field of genomic selection consists of designing a model that can be trained based on a training population that is able to predict the chosen phenotypes based only on the genotypes.

Machine learning is a field that has become more and more popular during the last decade. These algorithms are used to build complex models based on a large amount of data. These methods have provided very powerful models in a large variety of fields, from image segmentation to the processing of natural language.

As more and more animals are genotyped, the amount of available data becomes big enough to be able to consider machine learning methods. Some architectures trained on cattle datasets were proposed in the literature, but there are still many possible algorithms to be evaluated. In this thesis, we will explore a wide range of machine learning methods from the linear Ridge regression or XGBoost to more complex neural networks, like convolutional neural networks or transformers. The objective is to find models that perform better than the state-of-the-art GBLUP linear model, and perform some experiments on the models to better understand how to apply the machine learning methods for genomic selection.

From all the models we have trained, no model had better performance than the GBLUP models, but some models, the MLP, the ridge regression, and the SVM, reached similar performances. The experiments also show interesting results. The usage of maximum or average pooling layers seems to decrease the performance of the convolutional neural network. Models trained to predict several phenotypes at the same time do reach the same performance as the models trained on a single phenotype, which can reduce the time required to develop a model. Indeed, only a single architecture would have to be tuned instead of one for every phenotype. The impact of the size of the training set was studied. It shows that using more data could increase the performance of the different architectures. Finally, some suggestions are proposed for future research on the usage of machine learning methods for genomic selection.

Contents

List of Figures

List of Tables

Introduction

When breeding dairy cattle, reproduction is a mandatory step. Without it, no cows would produce milk, and there would be no next generation to take over when the previous one got sold for their meat. In an economic world where competitiveness is really important, having to rely on a random operation such as reproduction is not very reassuring. Owning a cow that produces less milk or that has an underdeveloped musculature reduces the profit of the farmers. As they do not already make a lot of profit, too many losses due to these problems may put their whole business at risk. In the opposite case, having a productive generation may increase the profit of the farmers, allowing them to invest in new equipment or secure the future of the farm. All these examples show the importance of reproduction for farmers, and a better understanding of the process could have a massive effect in this sector.

Genomic selection is a field that was developed thanks to technological improvements of the last twenty years. The goal is to predict certain characteristics called phenotypes based on the genotype of an individual. Based on this tool, it would become possible to estimate the performance of the progeny using the information contained in DNA, which will reduce the randomness of the reproduction process by using a harmless procedure. Several methods were developed that performed sufficiently well to enable the use of genomic selection all around the world.

Machine learning is a field of computer science that studies algorithms that use data to learn a model to classify or predict a value from the input. That field has become very popular in the last ten years, with the rise of deep learning, a subfield of machine learning that focuses on neural network-based models. Deep learning methods were applied to a wide range of problems, ranging from image classification to text translation or image generation, with very good results. However, while deep learning has achieved great success in some domains, other application domains, including the field of genomic selection, have not yet fully benefited from deep learning advances.

One state-of-the-art model in genomic selection is the GBLUP, which is a linear model. Some machine learning algorithms have been proposed in order to find a better model by using nonlinearity, but those models failed to perform better than the GBLUP. This thesis will test several machine learning algorithms with a special focus on deep learning algorithms due to their recent success in other fields. The goal is to perform a wide evaluation in order to determine which model is suitable for genomic selection and if they can match or improve the quality of the prediction provided by existing state-of-the-art models. As the literature is not very developed, a wide range of methods will be evaluated to provide an overview of the possibilities. In addition to the evaluation of the model, some experiments will be performed to evaluate the relevance of specific hyperparameters or methods. This will help to better understand how the algorithm can be applied in genomic selection.

Objective and Outline

The objective of this thesis is to evaluate the possible machine learning methods that can be applied based on the given dataset to search for an alternative to the linear GBLUP model that is widely used in genomic selection. As the literature contains only a few papers using a cattle dataset of this size, a wide range of methods is defined in order to cover all the main families of methods.

The second objective of this thesis is to perform some experiments on these architectures. As the dataset is uncommon in machine learning, the usual practice of the field may not be adapted to genomic selection. In this paper, experiments were designed for every architecture to test a wide range of hyperparameters. From the training pipeline to the usage of certain layers, these experiments aim to determine what is a good practice from what decreases the prediction of the model.

To summarize, the goal of this thesis is to provide a useful basis for any future research done with the provided dataset or more globally in the field of genomic selection. To do so, this report was written to be as complete as possible to ensure the experiments are reproducible and could be used as the basis for more advanced research, even if they do not use the same dataset as the one from this work. All scripts used during the project are available on GitHub at `https://github.com/Florent-Hervers/master-thesis-code` to enable in-depth analysis of every experiment presented in this document.

As this work touches on several domains, it is important to provide the basis from biology and computer science, to make the content of the document understandable to anyone, regardless of their specialty. Chapter 1 will introduce the relevant biology concepts and the dataset used in the thesis. After that, GBLUP, the state-of-the-art model, will be explained in detail, as it is used as the baseline in this thesis. Finally, an overview of the existing models in genomic selection will be performed. The next two chapters, Chapter 2 and Chapter 3, will explain, model by model, the algorithms and the experiments evaluated during this thesis. To avoid confusion between the different models, the relevant comparisons between the models will be discussed in Chapter 4. Finally, Chapter 5 will conclude this thesis and discuss potential directions for future works.

Chapter 1

Background

1.1 Terminology and essential concepts

This section aims to introduce the main terms and concepts mentioned in this thesis. Only the biological concepts will be explained here, while all computer science-related concepts will be explained in the following sections, where they will make more sense. The goal of this section is to give any reader with an engineering background the necessary knowledge to be able to fully understand what will be addressed in the rest of the thesis. For the sake of clarity, this section is not a complete description of the field of genomic selection, but rather an introduction to the field to ensure the understanding of this thesis.

1.1.1 DNA

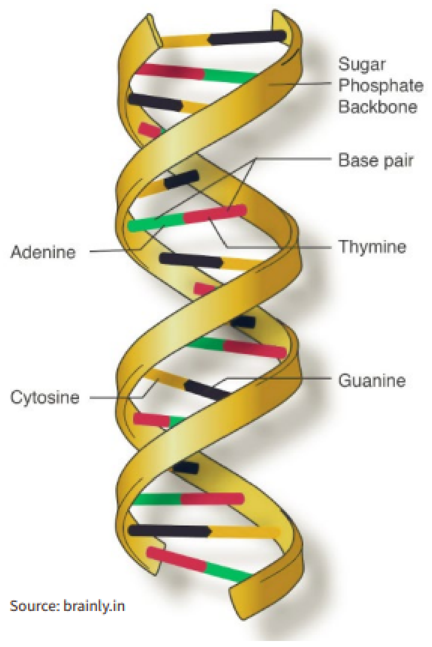


Figure 1.1: Schematic of the structure of the DNA molecule. (Taken from [1])

The deoxyribonucleic acid (abbreviated to DNA) is present in almost all organic organisms. Found in the nucleus of every cell, DNA contains the entirety of genetic information. This molecule is a polymer, and its building block, the monomer, is called a nucleotide. A monomer is composed of three molecules: a sugar molecule (2-deoxyribose), a phosphate molecule, and a nitrogen base. Because the sugar and phosphate molecules are common to all nucleotides, we usually describe the nucleotides by their nitrogen base. There are four different nucleotides in the DNA: Adenine, Thymine, Guanine, and Cytosine. To simplify the notation, only the first letter of the base is used when speaking about nucleotides. As shown in Figure 1.1, the DNA structure is a double helix with a matching of the bases between the two DNA strands: A is always matched with T, and C is always matched with G. Due to this matching, we only need the sequence of nitrogen bases of one DNA strain to describe the full DNA molecule. The second strain can always be inferred from the first one and thus does not carry any additional information.

1.1.2 Chromosomes, SNP and polyploidy

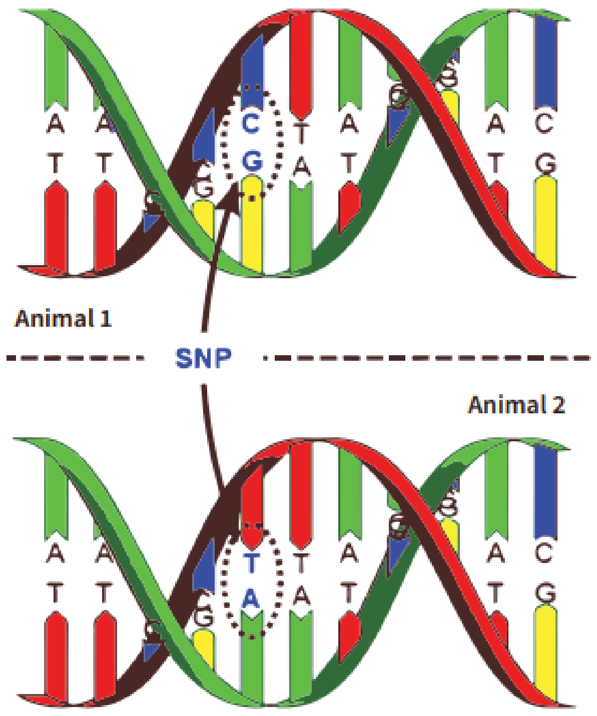


Figure 1.2: Illustration of a SNP between two individuals (taken from [1])

In this field, we are interested in the differences between individuals in order to choose the individuals that will perform better in the criterion of interest. The simplest modification in the DNA of an individual is a different nucleotide for a given locus (position in the chromosome or a gene). In the jargon, biologists call that a single-nucleotide polymorphism (for the sake of brevity, the abbreviation SNP will be used in the remainder of this document). Figure 1.2 shows a selected locus in the DNA molecule of two different individuals. We can see that all nucleotides are identical except for one single position. This is the SNP where we have an allele represented by the nucleotide C and the second by the nucleotide T. SNPs are created by mutations. Mutations are relatively rare, as it is uncommon. These mutations are transmitted from parents to progeny across generations. To give an idea of how unlikely mutations are, the number of new mutations in humans is between 50 and 90 per generation [12]. Tri-allelic SNP would require two mutations at the same position. Given the size of the cattle genome (2.7 billion base pairs) [13], it is very unlikely that this happens. Even if we detect a tri-allelic SNP, it is cleared from the data as it is too rare to be meaningful.

The next concept relevant to this project is the way the DNA is structured within an individual. Most of the time, DNA molecules are mixed inside a cell's nucleus without any structure, like a plate of tangled spaghetti made with DNA molecules. The term used to describe the DNA is chromatin. When cellular division occurs, DNA folds itself to form what is called chromosomes.
Figure 1.3 shows, on the left, the chromatin, which is the default state of the DNA, compared to the condensed version, the chromosomes, on the right of the figure. These chromosomes are visible using a microscope, and it is possible to assign a number to each chromosome to structure the genetic code via this ordering of the chromosomes. Humans are diploid organisms, meaning that each cell contains two sets of every chromosome. There also exist haploid organisms (with one set of chromosomes), such as the fire ant, or triploid (with three sets of chromosomes), such as the seedless watermelon or the Jonagold

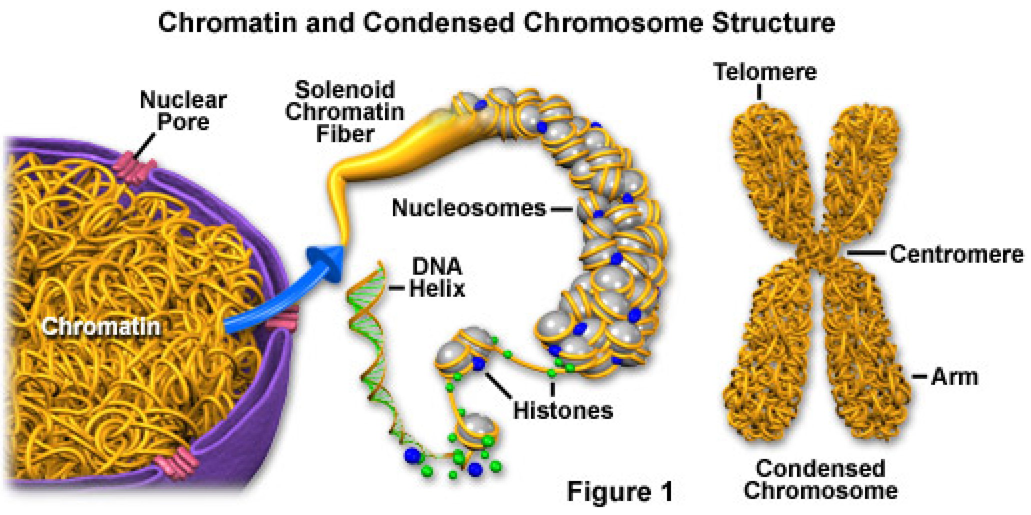


Figure 1.3: Comparison of chromatin and chromosomes (taken from [2])

apple [14, 15]. The number of sets of chromosomes for an individual is a very important property for genomic selection. For a given chromosome, the different replicas within an individual are not exactly the same: the nucleotide for a SNP may not have the same value for every replica. The higher the ploidy, the more the number of possible SNP combinations increases. For a diploid individual, there are three possible cases for a given position in DNA. Let A and a be different nucleotides. The possible combinations are the homozygous ones: AA and aa, and the heterozygous one: Aa (the order does not matter as we do not order the chromosomes within a set). In the case of a triploid individual, we would have four combinations: AAA, aaa, aaA, aAA. The ploidy impacts the number of possible combinations of the SNPs. All methods and models developed in this thesis will only be valid for diploid individuals, as the dataset used comes from cattle, which are diploid animals. Adapting the methods to non-diploid individuals will require modifications to take into account the increase in the number of possible combinations for a given position. Depending on the method, the modification may be straightforward or require major modifications to take into account the modified number of combinations.

1.1.3 Phenotypes and heritability

The next concepts to be explained are the phenotypes and their heritability. A phenotype can be defined as an observable or measurable characteristic. Some examples of this are the size, the eye color, or the blood group. There is a need to make a separation in the phenotypes. On one side, we have Mendelian traits. These traits are defined by a single gene, predicting the phenotype is thus as simple as finding only one gene that explains the trait [16]. The environment does not influence such phenotypes. Applying complicated methods on phenotypes that are defined by a single known gene does not make sense, as we already know where to look for improvement. The second type is complex traits. These are polygenic traits: they are influenced by many genes and the environment. This case is much more difficult, as there is no indication of how many genes are involved, which exponentially increases the number of possibilities, and some variations can be caused by the environment. It is impossible to predict a complex trait exactly using only the genotype of an individual, as the influence of the environment is not taken into account. This thesis and the field of genomic selection naturally focused on the complex traits. The model used to describe this is [17]:

$$P = G + E \tag{1.1}$$

The phenotype P is defined by the genotype G and the environment E. The effect of the environment is vague and widespread. Some examples of possible environmental effects are the temperatures, the diseases, the food, the interactions with other individuals, ... These effects can not be tracked completely and objectively. Thus, it can be considered as uncertainty on the phenotypic values. To quantify this uncertainty, researchers introduced the concept of heritability of a phenotype. It can be defined as the amount of variation in a trait that is due to variation in genetic factors [18]. To introduce the formula of heritability, we have to introduce some notations. From the definition of the variance and the equation (1.1) we have:

$$Var(P) = Var(G) + Var(E) + 2\,Cov(G, E)$$

Where $Var(G)$ denotes the genetic variance: the variance of the phenotypes that is caused by the genetic effects, and $Var(P)$ denotes the phenotypic variance: the classical variance of the values of the phenotypes. The formula of the broad sense heritability H^2 is $\frac{Var(G)}{Var(P)}$, the proportion of the variance in the phenotype due to the genotype. However, broad-sense heritability is not used in the field of genomic selection. The model used for the genetic effect is:

$$G = A + D + EP \tag{1.2}$$

Equation (1.2) indicates that the genetic effect is composed of three terms. The first one, A, represents the additive genetic effects (also called the breeding value). It is defined like this: "The additive genetic value of an individual is the sum of the average effects of all the alleles the individual carries" [18]. The second one, D, represents the dominance genetic effects: these are due to interactions between alleles at a single locus. The third and final one represents the epistasis genetic effects EP. It represents the effects due to the interaction of the alleles at different loci [19]. As the genome of a child will be built from a random combination of the genomes of the parents, they will therefore transmit half their additive effect to their progeny. Due to the randomness of the combination, the dominance and epistasis effects are less predictable.

This leads us to the definition of heritability we will use: the narrow-sense heritability h^2. The formula for this heritability is $h^2 = \frac{Var(A)}{Var(P)}$ where $Var(A)$ is the variance due to additive genetic effects. The focus on the additive genetic effect does not mean that the dominance and epistasis effects do not matter when predicting the phenotypes. The equation of the model (1.2) indicates that the dominance and the epistasis effect play a role that should not be neglected by the genomic selection to provide accurate predictions. However, these effects are difficult to predict from the parents to the progeny as they depend on the recombination of the chromosomes. As the parent will transmit half their additive effect to their progeny, this is why the narrow-sense heritability is preferred to the broad-sense heritability.

We can see that by definition, narrow-sense and broad-sense heritability are values between 0 and 1 as defined by a ratio of variances. These values are interesting as they quantify how useful the genotypes are for predicting a phenotype. A high value means that the genotypes impact the phenotype much more than the environment and may hint that the phenotype will be easier to predict using the genotypes than another.

1.1.4 Genomic selection

In 2001, [20] proposed a method where it would be possible to estimate phenotype values based on a dense marker map spread across the whole genome. This paper was the first one to use the genotypic data in order to predict the phenotypic values. It introduced a new field named genomic selection. To be able to perform genomic selection, we need to have a training population where the genotypes and the phenotypes are available. Based on this data, we use a model to predict the phenotypes from the genotypes. Once the best model is found, this model can be used to predict the phenotypes of individuals not in the training set. The usage of the genotype enables making more accurate predictions in cases where there is very little information about the parents and siblings, or when the animal is young. The two main components of the field are the data and the model. As the models are built using the data from the training population, the performances of genomic selection rely on the genotypes that are used. A model built based on ten sibling animals will not enable the construction of a good model, whatever the method used. The design of the model used to predict the phenotypes is also important for genomic selection. It needs to be able to extract from the genotypes the useful information and handle the influence of the environment on the phenotypes, as explained in the previous subsection. The objective of this thesis will consist of evaluating new models built using machine learning algorithms.

[20] introduced methods that were only demonstrated in simulated data. The technology of 2001 was not advanced enough to provide the required SNP markers. It was thus not possible to evaluate directly whether the usage of this new source of data can be used to improve the quality of the predictions. More or less ten years later, research has provided new methods to create the marker maps at a scale where they could be used in farms. Genomic selection for livestock animals was adopted all around the world: in France [21] or in the United States [22] and [23]. For example, milk, fat, and protein yields from dairy cattle nearly doubled between 1963 and 2013, and more than half of the improvement can be attributed to changes in genetics [22]. Genetics may have a big influence on chosen characteristics.

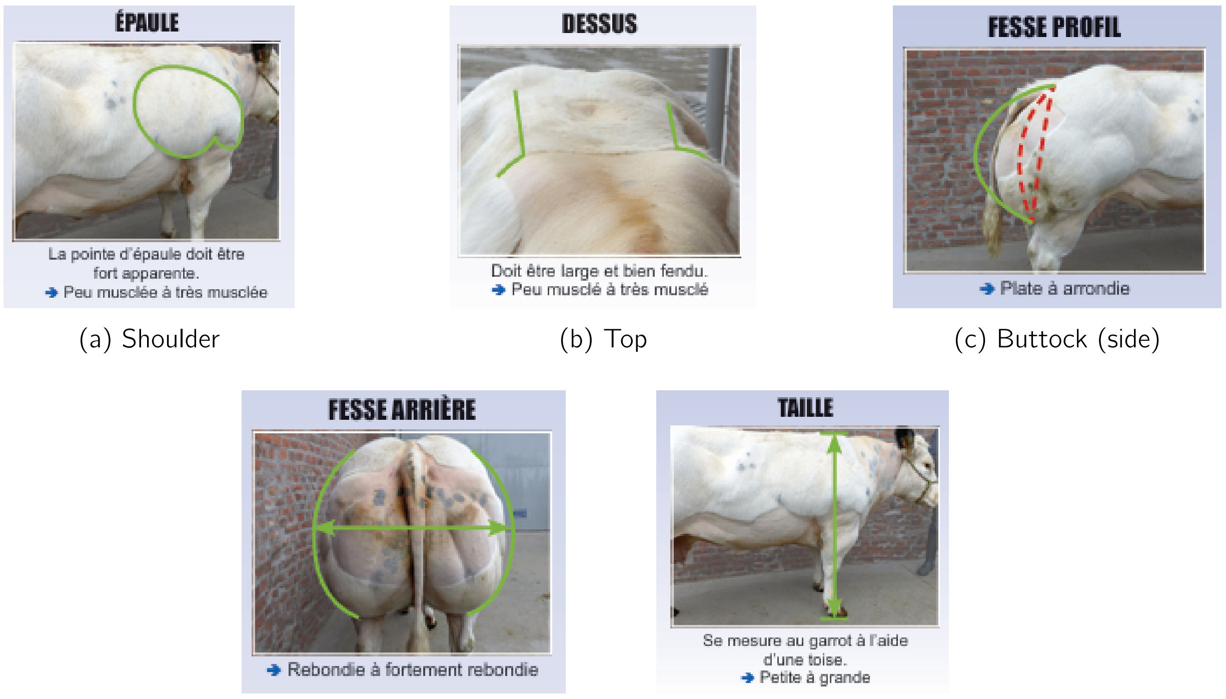
As the accuracy of the methods depends on the data used to train the model, the improvements due to genomic selection increase over time thanks to the addition of new data. The acquisition of new data is also sped up thanks to the technology improvements. The cost of genotyping a cow was reduced from 250$ in 2008 to between 40$ and 80$ in 2020, depending on the desired marker density [24]. For the same budget, it is possible to obtain 3 to 5 times more genotypes than in the early stages of the method, which helps to rapidly increase the databases for the different species. The implementation of international collaborations to bring together genotypes from different countries also helps to create big collections of genotypes that will contribute to improving the genomic selection methods and reducing the cost [24].

1.2 Dataset: definition and analysis

This section will introduce the dataset used during this study, where it came from, the species of cattle, the data format, and how the different sets were constructed. The dataset was provided by the Walloon breeding association (AWE). The association has two separate sections: the first one, called Elevéo, is a non-profit association that can provide specific services tailored to the needs of breeders. The second one, called Inovéo, is a cooperative that provides semen harvesting and genetics operations to breeders [25]. The association counts more than 4000 members and employs more than 200 people. The board of the company is composed only of members of that association, which ensures the association provides useful resources to the breeders in Wallonia. The provided dataset comes from Belgian Blue cattle. This breed is one of the most common in Belgium. Nearly 50% of the cows raised in Belgium are from the Belgian Blue breed. The original breed has been divided into two populations, one specialized in the meat and one dual-purpose for milk and meat production. The dataset used contains only animals from the meat population. It is also used in crossbreeding programs as it yields good results [26]. The data is not available publicly but was already used in a paper studying the possible improvement using the whole genome sequences instead of markers [27]. The dataset is composed of 36304 SNP markers for 18324 individuals, and the data of six different phenotypes were available.

Concerning the genotypes, they were provided in three files, one bed file, one bim file, and one fam file that follow the format used by the PLINK software [28]. For every individual, 36304 SNP markers were provided without any missing values. As explained in Section 1.1, there are three possibilities to encode a SNP for a given locus. Let a and A be two different alleles. The pair aa was encoded by 0, the pair aA (or Aa, the ordering does not matter) is encoded by 1, and the pair AA is encoded by 2. Another way to explain the encoding is that the value represents the number of allele A present in the individual. This encoding type is allele dosage and is used to model additive effects.

The first four phenotypes, named respectively shoulder, top, buttock (rear), and buttock (side), represent the muscular development for the shoulder, top, and two views for the buttock, the rear and side views. The fifth one, called size, describes the height of the animal's body. Figure 1.4 displays in green what these five phenotypes mean on a cow to help link the phenotype to the expression of these values. The last one, called musculature, is an artificial value obtained by summing all the muscular measures with different weights. The computation of the phenotypes was performed according to the following procedure. The values were first evaluated by specialized technicians on a 0-50 scale. The phenotypes were corrected for the effect due to the age of the animal using a quadratic regression. After that, the fixed effects were also taken into account, like the body condition, the date, or the place where it was performed. More details about the corrections can be found on the AWE website [29], and more details about the data sequencing can be found in these two papers [27] and [30]. Note that the value of the musculature phenotype is computed based on a weighted sum of raw values of the shoulder, top, and the two buttock phenotypes. The different corrections explain why the musculature is not perfectly correlated with those phenotypes.


ÉPAULE
La pointe d'épaule doit être fort apparente.
➔ Peu musclée à très musclée
DESSUS
Doit être large et bien fendu.
➔ Peu musclé à très musclé
FESSE PROFIL
➔ Plate à arrondie
FESSE ARRIÈRE
➔ Rebondie à fortement rebondie
TAILLE
Se mesure au garrot à l'aide d'une toise.
➔ Petite à grande

(a) Shoulder (b) Top (c) Buttock (side)

(d) Buttock (rear) (e) Size

Figure 1.4: Visualization of the phenotypes (taken from [3])

Machine learning methods are not methods that can be run directly on any dataset and provide the most accurate prediction possible. There are what we call hyperparameters: values that depend on the chosen algorithm or architectures that influence the quality of the results. Therefore, the most appropriate value for the parameters needs to be chosen. To avoid any selection bias, a separation is required between the data on which we will tune the hyperparameters, the validation set, from the data used to evaluate the best model, the test set. As the dataset size is relatively big, there is no need for cross-validation methods (that is used when the dataset is small to avoid selection bias without reducing the dataset size used for the training). The individuals to be used in the test set were defined with the data provided to be able to compare the results of this thesis with other experiments done with the same dataset. The size of the validation set was chosen to be 1000. All remaining samples available composed the training set on which we will construct all models. However, one last criterion has to be satisfied. To avoid bias in the model, the genotypes in the validation and test sets should be from different generations. If the progeny of an individual is found in the training dataset, it will be easier for the model to predict its phenotype. In practice, we want to use those models on young individuals that has no progeny yet. To have split the dataset coherently with the utilization in practice, the data is sorted by birth date. The individuals who composed the validation and test sets must be from younger generations. The test set already satisfies this constraint. The validation set is then built by taking the 1000 younger samples from the data not present in the test set.

Table 1.1: Dataset summary per phenotype

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Training set size	13300	13300	13300	13300	12533	12461
Validation set size	1000	1000	1000	1000	1000	1000
Test set size	3252	3252	3252	3252	3315	3047
Narrow-sense heritability h^2	0.304	0.339	0.375	0.327	0.434	0.381

Unfortunately, the provided dataset does not have all six phenotypes for every individual. There are some missing phenotypes. It means that the size of the different sets presented varies depending on the phenotype. Table 1.1 presents the size of the sets per phenotype together with the narrow-sense heritability estimated from the training set. As the first four phenotypes were probably measured by the same technician in a single visit, the individuals in the sets are identical. This interesting property from the dataset will be used later in the thesis. Note that for the Musculature phenotype that depends on the first four phenotypes, the sizes of the sets are different. It is caused by the fact that more traits are used to compute the Musculature phenotype that may be missing from some individuals, making the computation impossible. For the narrow-sense heritability, we can see that the values are moderate. For example, the heritability of the size for humans is 0.8 [31]. However, the phenotypes available are the one that has the biggest heritability from the ones listed on the website [29] and are thus the ones the most suitable to be used.

This dataset will be the only one used during this thesis. One particular property of our dataset is its size, which is quite big compared to the ones mentioned in the literature. As there were many candidate algorithms to evaluate, the training and evaluation of the models resulting from the algorithms will take a considerable amount of time, as there are six phenotypes to evaluate.

Table 1.2: Mean and variance of the phenotypes in function of the set

(a) Means

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Training set	3.388	5.086	3.312	3.244	-3.624	2.873
Validation set	3.637	5.442	3.360	3.323	-2.879	2.901
Test set	3.789	5.679	3.550	3.569	-3.121	3.051

(b) Variances

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Training set	4.206	14.865	2.748	2.584	59.318	3.009
Validation set	3.545	14.331	2.236	2.488	57.493	2.955
Test set	3.212	12.712	2.488	2.301	59.970	2.714

In order to give a bit more insight about the data from the different sets, Table 1.2 summarizes the means and the variances of the phenotypes for the different datasets. The first thing we can observe is that there are no statistics that are the same for the three sets. This can perturb the predictions of the machine learning model as it will assume that the new data comes from the same distribution, but from the statistics in Table 1.2, this does not seem to be the case here, which is something to keep in mind. When looking at every muscle-related phenotype, we can see that the mean of these phenotypes increases over time (and the different sets) while the variances decrease. The statistics for the size phenotype are very different than the ones for the muscle-related phenotypes: the mean is negative and the variance is much bigger.

1.3 Genomic best linear unbiased prediction (GBLUP) model

This section will explain and present one of the most used methods in genomic selection: the genomic best linear unbiased prediction model, or GBLUP. This model will be very important in this thesis as it will be used as the baseline to compare the results of the models developed in the following sections. The section will be divided into three sub-sections: the first will explain the mathematical model used in the GBLUP, together with the expression of the solutions of the model. The second part will explain how the model of the first section is applied to obtain the GBLUP model. In this section, an alternative formulation will also be introduced to show the two main approaches of the GBLUP model. The final section will explain how one can use the GBLUP model to make predictions on genotypes based on the available data.

1.3.1 Best unbiased linear estimator (BLUP)

This first section aims to provide a general introduction to the best linear unbiased model (also called BLUP) before covering models used in genomic selection. The goal of this subsection is to introduce the equations in a general way such that any reader can understand where the equations come from. As the concept of BLUP models is not specific to the genomic selection field, the presented equations in this section will be general.

The BLUP model uses a linear mixed model ([32–34]). The general expression of such a model is:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{V}\mathbf{u} + \mathbf{e} \tag{1.3}$$

where:

- $\mathbf{y} \in \mathbb{R}^n$ is the data vector to predict and n is the number of observations available.
- $\mathbf{b} \in \mathbb{R}^q$ the unknown vector of fixed effects.
- $\mathbf{u} \in \mathbb{R}^p$ the unknown vector of random effect that follows a normal distribution with a mean of 0 and a covariance matrix $\mathbf{U} \in \mathbb{R}^{p \times p}$.
- The matrices $\mathbf{X} \in \mathbb{R}^{n \times q}$ and $\mathbf{V} \in \mathbb{R}^{n \times p}$ are known matrices that links respectively $\mathbf{y}$ to $\mathbf{b}$ and $\mathbf{y}$ to $\mathbf{u}$ for every object.
- The vector $\mathbf{e} \in \mathbb{R}^n$ is a vector of random error assumed to be normally distributed with a mean of 0 and a covariance matrix $\mathbf{R} \in \mathbb{R}^{n \times n}$.

The variables of the model are the vectors $\mathbf{b}$ and $\mathbf{u}$, the fixed and random effects that are used to predict the value of the $\mathbf{y}$ vector. The name of the model can be explained by the structure of equation (1.3). The linear part is obvious from the structure of the equation. The model is mixed as it takes into account both fixed and random effects in its formulation. The solution of such a model is given by the Henderson mixed model equations ([32, 33]). The matrix form of these equations is as follows:

$$\begin{pmatrix} \mathbf{X}^T\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}^T\mathbf{R}^{-1}\mathbf{V} \\ \mathbf{V}^T\mathbf{R}^{-1}\mathbf{X} & \mathbf{V}^T\mathbf{R}^{-1}\mathbf{V} + \mathbf{U}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{b} \\ \mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{X}^T\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{V}^T\mathbf{R}^{-1}\mathbf{y} \end{pmatrix} \tag{1.4}$$

The main property of this formulation is that the solution of these equations will have the least mean squared error compared to all linear estimators. The "best" from the BLUP comes from there. The "unbiased" adjective comes from the fact that the average of the estimations will be equal to the average of what we wanted to estimate. The "predictor" term indicates that the model will estimate the random effects. The convention is to call estimators models that estimate the fixed effect, while predictors are models that estimate the random effects [32].

1.3.2 SNP-BLUP

The first formulation used in genomic selection that we will analyze is the SNP-BLUP. Let us apply the equation (1.3) to a single phenotype (for the sake of simplicity), which gives us the following equation:

$$y = \mathbf{1}_n \mu + \mathbf{Z}\mathbf{a} + e \tag{1.5}$$

Where:

- The only fixed effect is the mean of the phenotype μ
- The matrix $\mathbf{X}$ of Equation (1.3) became a vector of one (denoted as $\mathbf{1}_n$)
- The matrix $\mathbf{Z}$ contains all m SNP marker values from the available genotypes. For every marker j and individual i and a and A be different nucleotides, the value at element Z_{ij} will be equal to 0 if individual i has the genotype aa at marker j, 1 if the genotype is aA, and 2 if the genotype is AA. All individuals have to be present in the matrix, even those who need to be predicted. Note that the format of the dataset described in Section 1.2 is the same as for the $\mathbf{Z}$ matrix. This matrix is thus known and available thanks to the structure of the dataset.
- The vector $\mathbf{a}$ is the effect of the SNP, element a_i represent the contribution of effect i. The vector is assumed to be distributed from a normal distribution of zero mean and a variance of σ_a^2 shared for all markers.
- We still assume that the random error $e \sim \mathcal{N}(0, \sigma_e^2)$.

In Equation (1.5), we can see that the only fixed effect is the mean of the phenotype μ, and the random effects are the ones from every marker $\mathbf{a}$. Given the general solution of the mixed linear model (1.4) and the model formulation described by Equation (1.5), the formulation of the solution of SNP-BLUP is as follows:

$$\begin{pmatrix} \mathbf{1}_n^T \mathbf{1}_n & \mathbf{1}_n^T \mathbf{Z} \\ \mathbf{Z}^T \mathbf{1}_n & \mathbf{Z}^T \mathbf{Z} + \mathbf{I}\lambda \end{pmatrix} \begin{pmatrix} \mu \\ \mathbf{a} \end{pmatrix} = \begin{pmatrix} \mathbf{1}_n^T y \\ \mathbf{Z}^T y \end{pmatrix} \tag{1.6}$$

Where $\mathbf{I} \in \mathbb{R}^{m \times m}$ is the identity matrix and $\lambda = \frac{\sigma_e^2}{\sigma_a^2}$ is called the shrinkage parameter. Note that the solution of the SNP-BLUP, described in equation (1.6), is similar to the solution of a least squares problem. The only difference is the $\mathbf{I}\lambda$ term that is not present in a least squares solution. The scalar λ "shrinks LS estimates toward zero to an extent depending on the noise-signal ratio" [35]. It also makes sure that the model has a unique solution even when the matrix $\mathbf{Z}^T\mathbf{Z}$ is singular.

The main issue with the SNP-BLUP formulation is that the dimensions of the problem to solve are proportional to the number of SNP markers used. In genomic selection, the number of markers is usually in the order of ten thousand markers. For example, in our dataset, we have 36304 markers. The SNP-BLUP formulation is thus not very practical for genomic selection, but explaining the formulation is useful when we compare the model with the ridge regression later in Section 2.3.

1.3.3 GBLUP

The SNP-BLUP formulation described in the previous subsection is not the only formulation possible. It is possible to formulate the problem in a second way that does not depend on the number of markers. This formulation is called the GBLUP. This model is important in this thesis as it is the one that we will use as a baseline when comparing the results of the models developed in this work. The equation of the model is very similar to the one for the SNP-BLUP:

$$y = \mathbf{1}_n \mu + \mathbf{P}\mathbf{g} + e \tag{1.7}$$

Where:

- $\mathbf{P} \in \mathbb{R}^{r \times f}$ where r is the number of phenotypes individuals and f the total number of individuals in the analysis (with and without phenotypic data). The value at element $\mathbf{P}_{ij}$ is one if the phenotype i in the list of available phenotypes corresponds to animal j or zero otherwise [36]. The individuals without records will then be estimated based on the available records and the model.
- The $\mathbf{g} \in \mathbb{R}^f$ vector is the vector of genetic values is now distributed following $\mathcal{N}(0, \mathbf{G}\sigma_g^2)$ where $\mathbf{G} \in \mathbb{R}^{f \times f}$ is the genomic relationship matrix and σ_g^2 is the additive genetic variance (called V_A in the Section 1.1 when introducing the heritability).
- Like in the formulation for the SNP-BLUP, $\mathbf{1}_n$ is a vector of ones, μ is the unknown mean of the phenotype and $e \sim \mathcal{N}(0, \sigma_e^2)$ is the random error.

The genomic relationship matrix (GRM) stores in the element $\mathbf{G}_{ij}$, the proportion of the markers that are shared between individual i and j. In Equation (1.7), there is no sign of the genotypes of the animals. This is because the GRM is constructed based on the $\mathbf{Z}$ matrix introduced in the previous subsection, which contains the genotypes of all individuals in our dataset. There are several methods to compute the GRM [37]. We will only cover in detail the two methods we have used during this project: the VanRaden method and the Yang method. The expression of the element $\mathbf{G}_{jk}$ in the VanRaden-GRM is:

$$\mathbf{G}_{jk} = \frac{\sum_i (\mathbf{Z}_{ij} - 2p_i)(\mathbf{Z}_{ik} - 2p_i)}{2\sum_i p_i(1 - p_i)} \tag{1.8}$$

where i refers to loci and j/k refers to individuals. Note that in the Equation (1.8), the denominator is the same $\forall j, k$ and can be seen as a constant. This rescaling is done to be analogous to the pedigree-based additive relationship matrix that was used in BLUP methods before the GBLUP one. The $2p_i$ term in the numerator is used to set the mean for a given locus to zero in the $\mathbf{Z}$ matrix. Using the fact that we study diploid animals, the mean $\mathbb{E}[\mathbf{Z}_{i.}]$ is equal to $2p_i$.

The expression of the Yang-GRM is as follows:

$$\mathbf{G}_{jk} = \begin{cases} \frac{1}{N} \sum_i \frac{(\mathbf{Z}_{ij} - 2p_i)(\mathbf{Z}_{ik} - 2p_i)}{2p_i(1-p_i)} & \text{if } j \neq k \\ 1 + \frac{1}{N} \sum_i \frac{\mathbf{Z}_{ij}^2 - (1+2p_i)\mathbf{Z}_{ij} + 2p_i^2}{2p_i(1-p_i)} & \text{if } j = k \end{cases} \tag{1.9}$$

where i refers to loci, j/k refers to individuals, and N is the number of markers. The second case in Equation (1.9), where $j = k$ is a special case that we will not detail here. We will focus on the first case.

When we observe the two expressions, we can see that they are very similar. In Equation (1.9), the product $(\mathbf{Z}_{ij} - 2p_i)(\mathbf{Z}_{ik} - 2p_i)$ is normalized where it is not the case in Equation (1.8) for the VanRaden-GRM. Indeed, $Var(\mathbf{Z}_{ij} - 2p_i) = \mathbb{E}[(\mathbf{Z}_{ij} - 2p_i)^2] = 2p_i(1 - p_i)$ as by construction $\mathbb{E}[\mathbf{Z}_{ij} - 2p_i] = 0$. The expression of Yang-GRM makes the hypothesis that all markers as the same contribution to the variance, which implies that the rare variants have a larger impact than in VanRaden, where the contribution is defined by $2p_i(1-p_i)$ which is maximum when p_i is equal to 0.5: when the effect is common.

The VanRaden-GRM and Yang-GRM are thus very similar but make different assumptions about the importance of the rare effects. According to [38]: "There is very little difference in accuracy between these two relationship matrices". In this thesis, we will evaluate the GBLUP model with both GRM to evaluate the choice of the GRM construction method.

The last point to discuss about the GBLUP is the formulation of the solution to the model. According to Equation (1.4), the solution can be expressed as follow:

$$\begin{pmatrix} \mathbf{1}_n^T\mathbf{1}_n & \mathbf{1}_n^T\mathbf{P} \\ \mathbf{P}^T\mathbf{1}_n & \mathbf{P}^T\mathbf{P}+\mathbf{G}^{-1}\frac{\sigma_e^2}{\sigma_g^2} \end{pmatrix} \begin{pmatrix} \mu \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{1}_n^T y \\ \mathbf{P}^T y \end{pmatrix} \tag{1.10}$$

Compared to the solution derived in Equation (1.6), the expression from equation (1.10) will create as many equations as the number of individuals instead of the number of SNP markers used. These two formulations can be shown to be equivalent [38]. The only difference resides in the number of equations to be solved. When the number of markers is greater than the number of individuals, then the GBLUP formulation is to be preferred, while in the opposite case, the SNP-BLUP will require solving fewer equations. As the number of markers is usually between 20000 and 50000, and there are not many datasets that can surpass this size, the GBLUP formulation is the one most used in practice.

1.3.4 Predicting new phenotypes with GBLUP

The procedure to compute the prediction from the GBLUP model with our dataset will be split into three steps. The software used to perform all steps is LDAK version 6 [39]. The order of the steps is important as the results of the previous steps are used in the next ones. There is thus no possibility of executing two steps in parallel.

1. Compute the GRM: As the dataset provides the **Z** matrix, everything is available to compute the GRM, whatever the method.

2. Once the GRM is computed, we still can not solve Equation (1.10) as both σ_e^2 and σ_g^2 are unknown. It is thus required to estimate those variances to be able to make any prediction. We used the restricted maximum likelihood (REML) algorithm that is often used to solve mixed linear models, as it can compute an unbiased estimation of the covariance matrix parameters [40]. In this case, these parameters are σ_e^2 and σ_g^2 that we want to estimate.

3. Based on the variance estimation from the second step and the GRM from step one, we are set to finally solve the equations and get the GBLUP predictions for all cows from the dataset.

One last important detail in the second step is that the phenotypic data of the samples from the validation and the test set must be masked such that it can not be used during the REML algorithm. By doing so, the model will predict the phenotype of these samples (and not use it to estimate the variances).

1.3.5 Results and discussion

Table 1.3 summarizes the prediction on the test set based on the data of the training set only, depending on the method used to construct the GRM. We can see that the results for the correlation are close to 0.5 for every phenotype. For the mean absolute error, we can see that the results are not in the same order of magnitude as for the correlation. Using the variances found in Table 1.2b, we can see that the greater the variance, the greater the mean absolute error. We can also observe that the GBLUP yields better results in both metrics for all phenotypes when using the VanRaden method. The claim from [38]

Table 1.3: GBLUP results in function of the GRM computation method

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
GBLUP (Yang)	0.488	0.474	0.506	0.500	0.487	0.518

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
GBLUP (Yang)	1.212	2.425	1.058	1.014	5.356	1.104

is correct for our dataset, as the differences between both methods are in the order of the thousand. This gap is too small to claim that one method is better than another. However, due to the slight consistent improvement of the VanRaden method, only these results will be used as the baseline of this thesis. The results obtained with the Yang GRM will not be provided in the future tables to lighten the tables used to compare the results.

1.4 Overview of other genomic selection methods

As more and more data are being collected for the reasons mentioned before, researchers naturally became interested in one of the most popular fields of recent years: machine learning methods. It can provide complex and accurate models based on a large amount of data. The problem of genomic selection can very easily be posed as a regression problem. The input data would be the genotypes of the individual, and the output would be the predicted value of the phenotype. The main one, GBLUP, has been described in Section 1.3. Bayesian methods are also used as models in genomic selection: some examples are BayesA and BayesB [20] and BayesCπ and BayesDπ [41]. These models are either linear or statistical models. Machine learning models could be the basis of a new type of model that will perform better than the ones already used. Such models would also benefit from the increase in the datasets explained before.

In the literature, there are a few papers describing neural network architectures for cattle. For example, [42] presents a model where the phenotypes were first estimated using the GBLUP method (which will be described in the following section). Based on the results of GBLUP, the data is then given to an MLP model whose goal is to adjust the predictions based on non-linear interactions. Their method was evaluated on a simulated dataset and a real dataset containing more than a hundred thousand genotypes of French Holstein dairy cattle. However, their methods did not show any improvement in the Pearson correlation coefficient between the target and the model output (which is often used as a performance metric in this literature) compared to the values obtained using only the GBLUP model. [43] applied a feedforward Bayesian model called BANN for biologically annotated neural networks introduced in [44]. They used a dataset of six thousand Chinese Holstein dairy cattle in their paper and obtained a better Pearson correlation coefficient between the target and the model output than all the other tested methods.

As there are not many papers that have used animal datasets to evaluate the quality of the proposed model, architectures tested on crops like wheat and maize were also considered. [45] reviewed 23 papers that introduced either a multi-layer perceptron (MLP) or a convolutional neural network (CNN) architecture to different types of crops and some animal datasets. It has shown no relevant differences as the neural models performed better in only half of the paper's analyses. According to their analysis, they think that some models were poorly designed (bad model design or bad splitting of the dataset). They also found the CNN architectures to be better than the MLP ones, as most papers that outperformed the classical genome-based prediction models used CNN architectures. [46] introduced a three-layer CNN with batch normalization and dropout layers. They evaluate their results on four different datasets of different crops. The size of the dataset varies from a few hundred to two thousand genotypes. They showed that their architecture performed better on all analyzed traits and that their model performed better as more data became available to train the model.

As transformers became very popular since their introduction in 2017 [47], some researchers also tried to develop attention-based models. [48] introduced a model, named GPformer, based on auto-correlation attention and convolutional layers. The embeddings are created using a convolutional layer combined with the SNP physical position as positional encoding. A knowledge-guided module is also introduced instead of using the raw SNP inputs. This module used the results of genome-wide association studies (GWAS) to add information for every marker. They used four crop datasets whose size varies between three hundred and two thousand four hundred genotypes. The paper introduced a custom metric, the consistent index, that combines the mean absolute error and the Pearson correlation coefficient. Using this consistent index, GPformer performed better in almost all analyzed traits. The usage of the knowledge-guided modules enabled to increase even more the results based on the consistent index, such that the GPformer with this module outperformed every tested model. However, these results can not be easily compared to other papers due to the usage of their custom consistent index.

To conclude this section, this overview of the literature showed that the usage of machine learning methods in genomic selection is still in an early stage. When using the crops dataset, different models have been proposed, but there is no consensus about one. Every paper compares its architecture with different machine learning based models, which makes the comparison between the papers difficult. There is no clear state-of-the-art machine learning model that is used by a large number of researchers yet. However, algorithms developed based on crop datasets yielded better results than the conventional methods, which could indicate that an improvement is possible by using machine learning methods. The fact that many architectures were used for those models indicates that there are improvements to be made using those datasets. The downside of having many good models that are trained on different datasets is the difficulty in selecting the models that will suit the problem the most. For dairy cattle, only a few architectures were proposed. None of them significantly improves the Pearson correlation of their models with the target phenotypes, which indicates that we are still at an early stage of the research. As the goal of this thesis is to evaluate many different machine learning methods, researchers could use the conclusions of this work to continue the search for a better model. The exploration of the different methods will also enable us to define the methods that are worth using in the future from those that do not yield good results. As there are few architectures proposed in the literature, this exploration may be of great help to better understand the link between machine learning and genomic selection.

Chapter 2

Machine learning methods

The next two sections will present the different methods that were tested during this work. This chapter will present the so-called machine learning methods. The next chapter will present so-called deep learning methods. This next chapter will present the methods trained using the PyTorch Python library [6]. This section will focus on models that follow the scikit-learn library interface [49]. This separation makes this report more structured by avoiding a single big chapter with all methods within. The separation can also be justified from a practical point of view. The methodology employed in this chapter is slightly different from the one used in the following chapter. Splitting the methods into two chapters will enable us to clearly delimit which methodology was used for which models, such that no ambiguity remains after reading this document. The next section will thus explain the methodology applied to all models that will be presented in the later sections.

2.1 Methodology

The methodology used for the following model begins by determining which hyperparameters will be evaluated, together with the starting range of values. Once these are defined, all possible combinations were tested in order to look for a local maximum in the prediction accuracy (another term for the Pearson correlation coefficient). The hyperparameters that lead to the best-performing model on the validation set within a model family are then selected as our final model. For some architectures, some experiments were performed during the grid search, the details will be provided for every model concerned. The second most used metric in the field, the mean absolute error, will also be provided as a secondary metric. The best model will be the one that achieves the maximum prediction accuracy. The validation set will not be added to the training set when we evaluate on the test set, as the hyperparameters are not optimized with this data. It could thus make the predictions worse and decrease the performance of the models. In each section, the results of these models will be compared to the GBLUP baseline, and the results will be discussed. All models will be compared together in Section 4.6. As some of these models are very quick to train, we performed five runs with different seeds to evaluate the variance of the model. In that case, we will provide the mean of these five runs and the standard deviation in order to evaluate the variance of these models. We do not perform several runs for the slower algorithms, as the goal of this thesis was to focus on the exploration of the different techniques. The choice was made to privilege experiments over variance estimation of the final models.

2.2 Random forest

The basis of the Random Forest algorithm is binary decision trees. Figure 2.1 shows a binary regression tree where X_1 and X_2 are input features (in our context, SNP markers), t_i are thresholds, and r_i are the model predictions. The predictions are defined as the average of the phenotypic values of the training samples

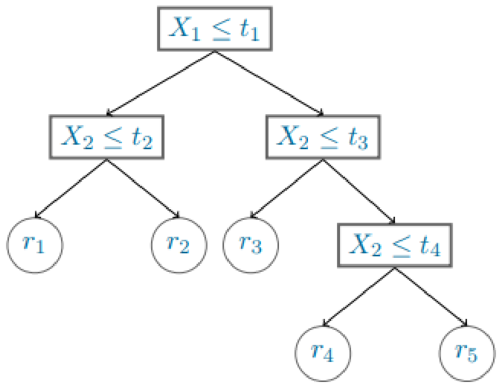


Figure 2.1: Example of a binary regression tree (taken from [4])

ending in the corresponding final node (called a leaf). Regression trees are quite powerful predictors as they can perfectly fit the training data by creating a case for every training example. However, this leads to poor performance on data not used during the training. We see that the model overfits the training data. To avoid developing the tree to a point where it overfits, several methods were developed. The only method we will consider consists of restricting the depth of the tree. In a decision tree, the depth is defined as the maximal number of conditions encountered before reaching a leaf. For example, the depth of the tree in Figure 2.1 is three when we always follow the branch at the right. When we set a maximal value that the tree cannot exceed, we limit the number of leaves of that tree to 2^d, where d is the depth. The depth also constrains the number of SNP markers that will be considered to produce a solution, as only one marker can be evaluated in a condition.

2.2.1 Experiments

Random forest is an algorithm specifically designed for trees. It will construct several trees (in our case, 1000). For each tree, we build the tree from a bootstrap sample of the same size as the training set. In other words, it performs sampling with replacement. Some samples will be selected several times, while other samples could not be selected, which reduces the effective number of samples used to build the tree while avoiding constructing several times the same tree as the training set is not the same. To speed things up and further perturb the data, we also restrict the number of SNP markers considered to find the best split. The final prediction of the model is the average prediction of the 1000 trees. The averaging of the prediction will reduce the variance of the prediction. The drawback of the averaging is an increase in the bias, as the best features may not be selected at each node due to the restrictions and the reduction of the learning set caused by the bootstrap sampling when building the trees. The two hyperparameters we have evaluated are thus the maximum authorized depth for every tree, called max depth, and the number of features considered at each split, called max features. For the max depth hyperparameter, the value from 1 to 35 where evaluated with an increment of 1 between 1 and 15 and an increment of 2 between 15 and 35. Values higher than 35 were not considered, as the highest depth in a tree observed in an unconstrained random forest is 42, and the average depth is 31.749. Evaluating for higher values of max depth will not be interesting, as the relaxation will only concern a few trees, which will not impact the final result much. For the max features, the tested values were $\frac{N}{2}, \frac{N}{3}, \frac{N}{4}, \frac{N}{10}, \frac{N}{100}, \sqrt{N}, \frac{N}{1000}$ where N = 36304 is the number of markers. The choice of the values was defined arbitrarily. The hyperparameter tuning induced the creation, the training, and the evaluation of $20 * 7 * 6 = 840$ models for all six phenotypes. Table A.1 contains the best found configuration for all phenotypes. The results of these best models on the validation set can be found in Table B.1.

2.2.2 Results and discussion

Table 2.1: Random forest results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
Random Forest	0.382 $\pm$ 0.00138	0.358 $\pm$ 0.00087	0.394 $\pm$ 0.00160	0.374 $\pm$ 0.00180	0.324 $\pm$ 0.00312	0.394 $\pm$ 0.00172

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
Random forest	1.324 $\pm$ 0.00082	2.635$\pm$ 0.00042	1.152$\pm$0.00061	1.112$\pm$ 0.00071	5.974$\pm$0.00314	1.187 $\pm$0.00123

The results are summarized in Table 2.1. The mean absolute error results are close to those obtained by the baseline, except for the size phenotype. However, when we look at the correlation metric, the results from the random forest are far from those obtained with the GBLUP. We can also see that the variance of the model in both metrics is very low, which indicates that the reduction of variance induced by the random forest works well. As a conclusion, this algorithm does not suit the problem of genomic selection as it failed to reach similar performances to the baseline.

2.3 Ridge regression

Ridge regression is one of the most popular variants of the linear regression problem. Let be $\mathbf{Z} \in \mathbb{R}^{n \times m}$ the matrix containing the m SNP markers from n individuals. The ridge regression problem modelisation of the problem is:

$$\mathbf{y} = b\mathbf{1}_n + \mathbf{Z}\mathbf{w} \tag{2.1}$$

Where $\mathbf{y} \in \mathbb{R}^n$ is the vector of prediction of the ridge model, $\mathbf{1}_n \in \mathbb{R}^n$ a vector full of one, b the bias term and $\mathbf{w} \in \mathbb{R}^m$ the vector of weights of the models for every markers. The parameters $\mathbf{w}$ and b are defined such that the prediction given by Equation (2.1) minimized the square error $\left(|\sum_{i=1}^{n} y_i - b - \sum_{j=1}^{m} \mathbf{Z}_{ij} w_j\right)^2 + \lambda||\mathbf{w}||^2$. The second term is the regularization term introduced by Ridge regression, where λ is a hyperparameter to be chosen. This term encourages the model to shrink the model weights $\mathbf{w}$ toward zero. For example, if the vector $\mathbf{w}$ is equal to (1,-1), then its norm will be equal to $\sqrt{2} \approx 1.414$. But if the vector becomes equal to (-0.5,0.5), then the norm is $\sqrt{\frac{1}{2}} \approx 0.707$, which is smaller than in the previous case. The hyperparameter λ controls the trade-off between the decrease of the square loss and the shrinkage of the norm of the $\mathbf{w}$. The solution of such a model is [4]:

$$\mathbf{w}^*(\lambda) = (\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I})^{-1}\mathbf{Z}^T\mathbf{y} \tag{2.2}$$

Equation 2.2 also illustrates another advantage of the ridge regression: it has a unique solution $\forall \lambda > 0$. Indeed, the matrix $\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I}$ can not become singular and can thus always be invertible. This is a great property to have as the matrix $\mathbf{Z}^T\mathbf{Z}$ can become singular if the training set contains the genotypes of several generations of individuals that have undergone genomic selection, as individuals become more and more inbred [50]. We can also see that Equation (2.2) is similar to the solution of the SNP-BLUP model, Equation (1.6), if we consider that the fixed effect, the mean, is zero.

2.3.1 Experiments

λ is thus the only hyperparameter to optimize in this model. We started the search from λ = 1 and increased the value of λ with a step that starts at one to end at 50 when the values of λ became bigger.

The starting value was chosen to match the default value of the scikit-learn implementation. For our dataset, the answer was to look for higher values based on the best configuration summarized in Table A.2. We stopped the search whenever we reached a local maximum in the correlation metric. Table A.1 contains the best found configuration for all phenotypes.

2.3.2 Results and discussion

Table 2.2: Ridge regression results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
Ridge	0.490	0.477	0.508	0.504	0.491	0.523

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
Ridge	1.208	2.421	1.053	1.012	5.352	1.098

Table 2.2 shows the results of the Ridge regression compared to the GBLUP. Results are almost identical; the differences in both metrics are in the order of 10^{-3}, which is not significant at all. This similarity can be explained by the observation made above: the SNP-BLUP and ridge regression have a very similar expression. As GBLUP is equivalent to the SNP-BLUP, it is logical that the model achieves similar performance. The differences between the two results should come from the estimation of λ via the REML algorithm, which is an automatic way to find a good λ. The algorithm may find a slightly different λ compared to the value found by the performed grid search. The results obtained are still valuable as they enable the validation of the results obtained with the GBLUP method. Two totally different implementations were used to find the same results; we can thus be confident that no mistakes were made while applying the GBLUP algorithm.

2.4 Support vector machine

Support vector machines are another variation of the linear regression model. The algorithm tries to find a weight vector $\mathbf{w} \in \mathbb{R}^m$, where m is the number of SNP markers, according to the following optimization problem ([51, 52]):

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\left(\xi_i + \xi_i^*\right) \\ \text{s.t.} \quad & y_i - \mathbf{w}^T\mathbf{Z}_{i.} - b \leq \epsilon + \xi_i^* \quad \forall i \in [1, n] \\ & \mathbf{w}^T\mathbf{Z}_{i.} + b - y_i \leq \epsilon + \xi_i \quad \forall i \in [1, n] \\ & \xi_i \geq 0 \quad \forall i \in [1, n] \\ & \xi_i^* \geq 0 \quad \forall i \in [1, n] \end{aligned} \tag{2.3}$$

where $\mathbf{Z}_{i.} \in \mathbb{R}^m$ refers to the m markers of the individual i, ξ_i and ξ_i^* slack variables, C and ϵ are hyperparameters. The prediction of the model is defined as $\mathbf{w}^T\mathbf{Z}_{i.} + b$. The first two constraints of (2.3) mean that the absolute error of the prediction against the true value must be smaller than or equal to ϵ; otherwise, the slack variable ξ must be non-zero. The last term of the objective function ensures that the slack variables are encouraged to be as close to zero as possible. In other words, the models make prediction errors less than ϵ. The hyperparameter C controls the tradeoff between the two terms of the objective function, while ϵ defines the tolerance of error of the model. Note that the fact that there are two constraints is due to the requirement of the optimization formulation. As the absolute value operator is not linear, we split the constraint into two to have a linear constraint. It makes the formulation a quadratic optimization problem that can be solved. The expression from (2.3) helps to understand the goal of the SVM algorithm. Unfortunately, the formulation that is used in the SVM algorithm is not this one. We prefer to use the Lagrangian dual formulation as it enables us to introduce the kernel trick. This formulation is ([51, 52]):

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\,\mathbf{Z}_{i.}^T\mathbf{Z}_{j.}+\varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)-\sum_{i=1}^{n}y_i\,(\alpha_i-\alpha_i^*) \\ \text{s.t.} \quad & \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0, \\ & 0\le\alpha_i\le C \quad \forall i\in[1,n] \\ & 0\le\alpha_i^*\le C \quad \forall i\in[1,n] \end{aligned} \tag{2.4}$$

The kernel trick consists of leaving the dot product $\mathbf{Z}_{i.}^T\mathbf{Z}_{j.}$ from Equation (2.4)implicit by replacing it by a Kernel $K(Z_{i.}, Z_{j.})$ that represent a measure of similarity between the two argument. This Kernel K must satisfy the following constraint to be valid. The $N \times N$ matrix B where the element $B_{ij} = K(o_i, o_j)$ must be symmetric and positive (semi)definite $\forall N \in \mathbb{N}$ and $\forall o_1, \ \ldots, o_N$ [4]. This constraint does not prevent us from using nonlinear functions. It would enable us to define a nonlinear predictor despite the linear formulation of the algorithm. During this thesis, we have considered the radial basis function (RBF) kernel. Its equation is given by [53]:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma(||\mathbf{x}_i-\mathbf{x}_j||^2)} \tag{2.5}$$

where γ is a hyperparameter to choose. It can be seen as "the inverse of the radius of influence of a sample" [54].

An important detail of this algorithm concerns the complexity of the model. The time complexity of the model is cubic in the size of the training set due to the quadratic solvers that are used to solve the model, and has a quadratic memory complexity with respect to the training set size due to the storage of all possible values for the kernel [55]. This algorithm can not be used as it is for a big dataset. Indeed, for the dataset used in this thesis (between 12k and 13k training samples), the training took between 1 and six hours, depending on the hyperparameter choice. The tuning of hyperparameters will thus take a lot of time. This is the reason why fewer combinations were tested for this algorithm compared to the others.

2.4.1 Experiments

The hyperparameters tuned in this thesis are the C from Equations (2.3) and (2.4) and the γ from the RBF kernel found in Equation (2.5). In an ideal scenario, the ϵ would also be optimized, but we could not do many experiments due to the reasons explained in the previous paragraph. The C hyperparameter does influence the slack variable and thus the trade-off between the two terms of the objective function defined in (2.3), but the effect is not the same as the ϵ hyperparameter. Optimizing only the value of C is an acceptable tradeoff, but not the optimal way of tuning a support vector machine. The starting point for the initial values was the default parameters of the scikit-learn implementation, $\gamma = 4.36 * 10^{-5}$ and C = 1, as they provided a good initial model. For the first phenotype, the shoulder one, a larger grid search was tested in order to explore the impact of the different hyperparameters. The search was narrowed until a local maximum in the correlation result was found. To reduce the number of combinations tried for the other phenotypes, we started the tuning by testing the values optimal for the shoulder phenotype: C

$\in [0.9, 1, 3.25, 5.5, 7.75]$ and $\gamma \in [7.75 * 10^{-5}, 10^{-5}, 1.225 * 10^{-4}, 1.45 * 10^{-4}, 1.675 * 10^{-4}]$. Based on this starting area, the next hyperparameter combinations were chosen to explore the promising areas of the space. Like for the first phenotype, the search was stopped when a local maximum was discovered. The summary of the best configuration found for the SVM can be found in Table A.3.

2.4.2 Results and discussion

Table 2.3: SVM results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
SVM	0.486	0.475	0.512	0.500	0.490	0.518

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
SVM	1.211	2.430	1.045	1.013	5.391	1.105

Table 2.3 shows the results of the models compared to the GBLUP. We can see that the results are very similar: the slight variations are in the order of the thousandth, which is too small to conclude anything about the model. However, due to the time complexity of the algorithm, SVMs are not very suitable for datasets with 10k individuals in comparison with the GBLUP, as it is much slower to train and produce the same results. For a small dataset, the unique optimization formulation of the SVM may compensate when there is only a small amount of data available. This section showed that SVM can perform as well as the GBLUP. The question of whether this observation can also be applied to smaller datasets remains open and could be an interesting follow-up to this thesis.

2.5 XGBoost

The XGBoost algorithm is part of the "boosting" algorithm family. Instead of creating a single complex model, these methods are based on many "weak" models that have a high bias to construct a solution. For the XGBoost model, regression trees are used as the "weak" models.

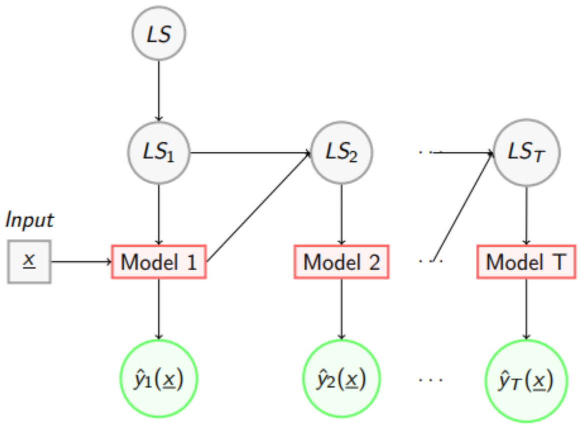

LS
LS_1
LS_2
LS_T
Input
$\underline{x}$
Model 1
Model 2
Model T
$\hat{y}_1(\underline{x})$
$\hat{y}_2(\underline{x})$
$\hat{y}_T(\underline{x})$

Figure 2.2: General representation of booting algorithms (taken from [4])

Figure 2.2 shows the general structure of the boosting algorithm. These models are trained thanks to a modified version of the data that depends on the prediction of the previous model and the previous learning set. As we are in a regression problem, the model's predictions $\hat{y}_i(x)$ are combined by a weighted sum in order to find the model prediction.

There are several boosting algorithms possible for regression problems. The XGBoost algorithm uses a modified version of the gradient boosting algorithm [56]. The definition of the loss function used by XGBoost is [57]:

$$L(LS) = \sum_{i \in LS} l(y_i, \hat{y}(\mathbf{x}_i) + \sum_{t \in \mathtt{T}} \gamma N_{leaf,t} + \frac{\lambda}{2} \sum_{l=1}^{N_{leaf,t}} w_l^2 \tag{2.6}$$

where $\hat{y}(\mathbf{x}_i)$ is the prediction of the XGboost model, l is a differentiable loss, $\mathtt{T}$ is the set of trees from the model, $N_{leaf,t}$ is the number of leaves of tree t, γ and λ are hyperparameters, and w_l is the prediction associated with leave l. Note that in this thesis, only the default square loss will be used as the loss function. The last two terms from Equation (2.6) are complexity penalty terms: $\gamma N_{leaf,t}$ penalizes the number of leaves, which indirectly limits the depth of the tree (as a regression tree may only have 2^d leaves for a tree of depth d. The term $\frac{\lambda}{2} \sum_{l=1}^{N_{leaf,t}} w_l^2$ is an L2 penalization of the prediction of the tree, like the one found in the Ridge regression. This regularized loss is designed to prevent the built tree from overfitting. This helps to maintain the intuition of the boosting algorithm that uses a combination of "weak" learners, as the trees will not be able to predict correctly due to all constraints in place.

The algorithm starts from an approximation $\hat{y}(x) = \arg\min_\gamma \sum_i L(y_i, \gamma)$ where $L(y_i, \gamma)$ is a loss function here. This first approximation is the best constant prediction according to the loss function. The second step consists of a loop of T iterations. For every iteration, the steps are:

1. For all couples genotypes/phenotypes $(\mathbf{x}_i, y_i)$ in the dataset, compute the gradient of the loss with respect to the prediction of the estimator $\hat{y}(x)$, $r(\mathbf{x}_i, y_i)$, which is defined as follow:

$$r(\mathbf{x}_i, y_i) = -\left| \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right|_{F=\hat{y}}$$

   In the XGBoost algorithm, this term is computed using a second-order Taylor expansion of the loss function [57].

2. Train a regression tree $\hat{y}_t = \arg\min_{\bar{y}} \sum_i (\bar{y}(\mathbf{x}_i) - r(\mathbf{x}_i, y_i))^2$.

3. Update the estimator $\hat{y}(x)$ via the formula $\hat{y}(x) = \hat{y}(x) + \mu \hat{y}_t(x)$.

To have more intuition on the meaning of the $r(\mathbf{x}_i, y_i)$, let us take an example using the square loss $L(y_i, F(\mathbf{x}_i)) = \frac{1}{2}(y_i - F(\mathbf{x}_i))^2$ as a loss function. The first step of an iteration is to compute the expression of $r(\mathbf{x}_i, y_i)$, which is $y_i - \hat{y}(\mathbf{x}_i)$. It represents the error made by the estimator $\hat{y}$ for the input $\mathbf{x}_i$. The second step results in the training of a regression tree that aims to predict $r(\mathbf{x}_i, y_i)$. Step three consists of updating the estimator. Let us assume the regression tree perfectly predicts $r(\mathbf{x}_i, y_i)$ and a learning rate μ equal to one, the estimator after the update becomes equal to $\hat{y}(\mathbf{x}_i) + r(\mathbf{x}_i, y_i) = \hat{y}(\mathbf{x}_i) + y_i - \hat{y}(\mathbf{x}_i) = y_i$: The estimator perfectly predict the output. This toy example illustrates the role of $r(\mathbf{x}_i, y_i)$, which estimates the error made by the current estimator. The algorithm tries to add a new tree to correct the error until we reach the maximum number of iterations.

2.5.1 Experiments

The considered hyperparameters of the XGBoost algorithm are the number of estimators T, which is equal to the number of iterations of the steps described above, as a single tree is constructed at each iteration. In this project, this value was set to 1000 and was not tuned. The learning rate μ is the first tuned hyperparameter. A lower learning rate requires more trees to converge toward the model's best performance. Each built tree prediction is multiplied by the learning rate. Reducing the learning rate also reduces the impact of every iteration, which explains why we need more trees when we reduce the learning rate. The tested values are [10^{-4}, $5*10^{-4}$, 10^{-3}, 10^{-2}, 0.1, 0.2, 0,3, 0.4, 0.5]. Evaluate values lower and higher than the default value of the parameter 0.3. Very low values were considered, as these are the common values of learning rate seen for neural networks. The second hyperparameter tuned is the max depth of the regression trees. The values tested are [2, 3, 4, 5, 6, 7, 8, 9, 10]. The range was centered on the default parameter value: 6. Finally, the last tuned hyperparameter is the subsampling one, which defines the proportion of the dataset to be used when building the trees. Like in the Random Forest, the advantage of reducing the learning set is to reduce the variance of the final model by avoiding overfitting on all samples. However, the variance of the model increases as the training depends on the individuals selected in the learning set. The tested values are [0.25, 0.5, 0.75, 1], where 1 means no subsampling. The models were trained using the open source implementation from [57]. The best configurations are described in Table A.4. The results on the validation set of this best configuration are described in the Table B.1.

2.5.2 Results and discussion

Table 2.4: XGBoost results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
XGBoost	0.435± 0.00118	0.418 ± 0.00295	0.447 ± 0.00551	0.441 ± 0.00545	0.378 ± 0.00498	0.462 ± 0.00252

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
XGBoost	1.260 ± 0.00170	2.530± 0.00319	1.096 ± 0.00449	1.060 ± 0.00319	5.746 ± 0.01535	1.138 ± 0.00372

Table 2.4 shows the performance of the XGBoost models compared to the one from the GBLUP baseline. We can see that for the mean absolute error, the XGBoost is close to the prediction of the GBLUP. However, the gap for the correlation metric is much bigger. The fact that the mean absolute error is close to the baseline means the model learns correctly how to predict individual phenotype values. However, the performance of the XGBoost model on the correlation metric is not as good as the baseline. This could be explained by the algorithm. During each iteration, the algorithm focuses on correcting the error made by the current estimator by adding a new estimator. As the focus of the latter is to correct the numerical error, the algorithm does not look to optimize the correlation. These kinds of models are thus by design, great estimators, but they fail to predict accurately the most important metric in genomic selection: the correlation.

2.6 Multi-trait regression

This final section will be a little bit different than the previous one, where a different type of algorithm is used. In this section, we will reuse the Random Forest and the XGBoost models. However, the task will be slightly different. In the previous section, all models aim to predict a single phenotype. We must then build a model specialized for every phenotype, which increases the workload. The tuning must be done for all phenotypes. There is no practical reason that justifies this choice; the input of every model is the same:

the animals' genotypes. We will thus train two extra models that will predict all the target phenotypes in a single run. Instead of outputting a scalar, we will output a vector containing the prediction for all targeted phenotypes. In this thesis, the targeted phenotypes are the first four phenotypes: the shoulder, top, and buttock (rear and side). The reason behind this choice was introduced in Section 1.2. Some phenotypes are missing for a few individuals, but for these first four phenotypes, the individuals of every set are the same. Choosing to target these four phenotypes allows us to compare the results from the models built with the multi-trait regression with the ones obtained in the previous sections, as they have the same learning, validation, and test sets. To avoid any confusion between the models, we will name, respectively, the Random Forest and the XGBoost model trained on the multi-trait regression problem, as Multi-RF and Multi-XGB.

2.6.1 Experiments

The hyperparameters are the same as those described in Section 2.2 and Section 2.5, respectively, for Multi-RF and Multi-XGB. The best model was selected using the sum of correlations of the predictions from the four phenotypes. This model selection is fair as it does not favor one phenotype against all others, which was the main criterion for choosing the model selection technique.

One problem that may arise from the multi-trait regression is that the targeted phenotypes do not have the same variance. Values from Table 1.2 show that the second phenotype has nearly a variance three to seven times higher than the other ones. Both algorithms use the square loss in their implementation, which penalizes predictions that are far away from the target value. The risk is that the models try to focus on the optimization of the phenotypes with a large variance, as these values will be more penalized by the square loss. To evaluate if this possibility has an impact, we tried to normalize the data by dividing all phenotypic values by their standard deviation of the phenotype. Table B.2 shows the best summed correlations for the two algorithms on the validation set. We can see that it improves the prediction for the Multi-XGB but not for the Multi-RF. The final configuration of Multi-XGB will use normalization. The impact of normalization in this setup does not lead to big improvements or degradations of the results. The impact is limited and depends on the algorithm used.

2.6.2 Results and discussion

Table 2.5: Multi-trait models results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)
GBLUP (VanRaden)	0.492	0.477	0.507	0.504
Multi-RF	0.378± 0.00132	0.352 ± 0.00075	0.364±0.0025	0.372±0.00222
Multi-XGB	0.436± 0.00274	0.416 ± 0.00256	0.455 ±0.00204	0.440 ± 0.00439

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)
GBLUP (VanRaden)	1.209	2.421	1.056	1.013
Multi-RF	1.331 ± 0.00056	2.639 ± 0.00124	1.166 ± 0.00066	1.125 ± 0.00065
Multi-XGB	1.380 ± 0.00244	2.611 ± 0.00471	1.104 ± 0.00222	1.089 ± 0.00147

Table 2.5 summarizes the results from the best models. The configuration of these models can be found in Table A.5. It can be seen that no phenotypes are favored compared to the others. There are no

phenotypes that perform significantly worse or better than the others for either of the metrics. We can also see that both models' performances are worse than those of the baseline.

Chapter 3

Deep learning methods

3.1 Methodology

This chapter aims to present the different deep learning architectures proposed and evaluated during this thesis. All models from this section are models trained using the Pytorch Python library [6]. The fact that we do not follow this interface anymore implies that the training of the model can not just be done with a single function call that will take care of everything. This section aims to explain the implementation to ensure the results of the architectures presented in the following section are reproducible.

The model training was performed using the Adam optimizer [58]. The loss function is the mean absolute error. It was decided not to use the classical mean square error, which is used for a majority of regression problems. The reason behind that choice is that the mean square error is not a metric that is used in the field of genomic selection. The metric that is used is the Pearson correlation coefficient. Unfortunately, this metric can not be used as a loss function as it is not decomposable and does not provide a unique solution. The mean absolute error is the second metric that is most used when comparing different models. That is why it was used in this work. The duration of the training was set to a maximum of two hundred epochs, where an epoch is a single iteration on the whole training set. The evaluation of the model on the validation is performed after every epoch. As the number of epochs required to reach the optimum depends on the architecture and is often way less than two hundred, an early stopping mechanism was implemented. If the correlation of the model prediction on the validation set is worse than 95% of the best correlation during fifteen consecutive epochs, then the training is stopped. The loss on the training set, the loss on the validation set, and the correlation on the validation set were computed at every epoch and tracked thanks to the weight and bias software [59]. To manage the different configurations easily through YAML files, the Python package hydra [60] was used to ease the development and the management of the configuration of all experiments. All experiments were performed on a maximum of 2 Nvidia 2080 Ti with 12 GB of VRAM.

In the next section, many experiments will be performed, but only a single configuration will be selected and evaluated on the test set. The process of finding and selecting the best configuration is different for each architecture. To avoid any confusion, the details are explained for every architecture in its section. As the main objective was to explore the possibility with the given dataset, hyperparameter tuning will not be performed for every architecture. As the resources are limited, only the best models were tuned in order to be able to evaluate several different architectures while evaluating if these best models are better than the current models used in genomic selection. Once the best configuration was selected, the models were retrained using only the data from the training set before being evaluated on the test set. This extra step was mandatory as the models of the experiments were not saved. This choice was taken to avoid overloading the storage from the cluster where the models were trained. As a lot of experiments were performed during the thesis, the management would not have been easy, as it would require constantly tracking the best results to manage the instance of the model that must be stored for every phenotype and architecture. To avoid making mistakes due to errors in the code or due to human errors, it was decided to proceed in that way. The usage of the Hydra package, GitHub, and the Weight and Bias software enabled us to track easily the configuration used, such that it becomes very easy to reproduce any performed experiment.

The next sections will present each architecture in turn and the experiments behind each architecture that were performed. Note that as this thesis aims to be read not only by engineers, the basic concepts will be explained. These reminders do not claim to be exhaustive, but they cover the main concepts required to understand the paper and why the architecture is used in this work.

3.2 Multi layer perceptron

The first kind of neural architecture that will be tested is the multi-layer perceptron. Simple but powerful, these kinds of models are the most basic form of neural network available.

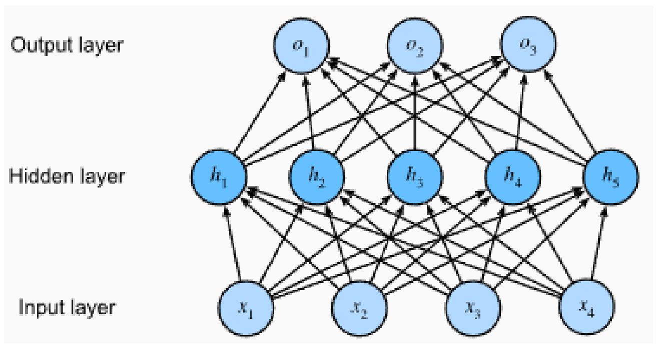


Figure 3.1: MLP with four inputs, five neurons, and three outputs (taken from [5])

Figure 3.1 represents an MLP with a single hidden layer. Every arrow represents a contribution to the neuron with an associated learnable weight. The equations describing the figure are:

$$\begin{aligned} h_i &= a\left(\sum_{j=1}^{4} w_{ij}^{(1)} x_j + b_i^{(1)}\right) \\ o_i &= \sum_{j=1}^{5} w_{ij}^{(2)} h_j + b_i^{(2)} \end{aligned} \tag{3.1}$$

Where $w_{ij}^{(L)}$ is the weight linking neuron j in layer L-1 to neuron i in layer L, $b_j^{(L)}$ is the bias associated to the neuron i of layer L, and $a(.)$ is a non linear function called the activation function. This activation function is really important and must be nonlinear. Indeed, without it, the model would be a mixture of linear models. Adding this activation function introduces a small amount of nonlinearity that yields very powerful results. Indeed, it is proven that when the model has enough hidden neurons, it can approximate any continuous function [61]. For this reason, it makes sense to begin the exploration of neural-based methods by using such an architecture as a starting point for the exploration of deep learning models.

As in the literature on genomic selection, we almost only see models with few layers, we will evaluate two different architectures for the perceptrons: Shallow-MLP and Deep-MLP. Shallow-MLP is a model with one hidden layer and a ReLU activation function. Figure 3.2 shows the graph of the ReLU function. It can be seen that the function is almost linear everywhere but at zero. If the input is negative, the output of the ReLU will be zero, otherwise, the ReLU function does not modify the input.

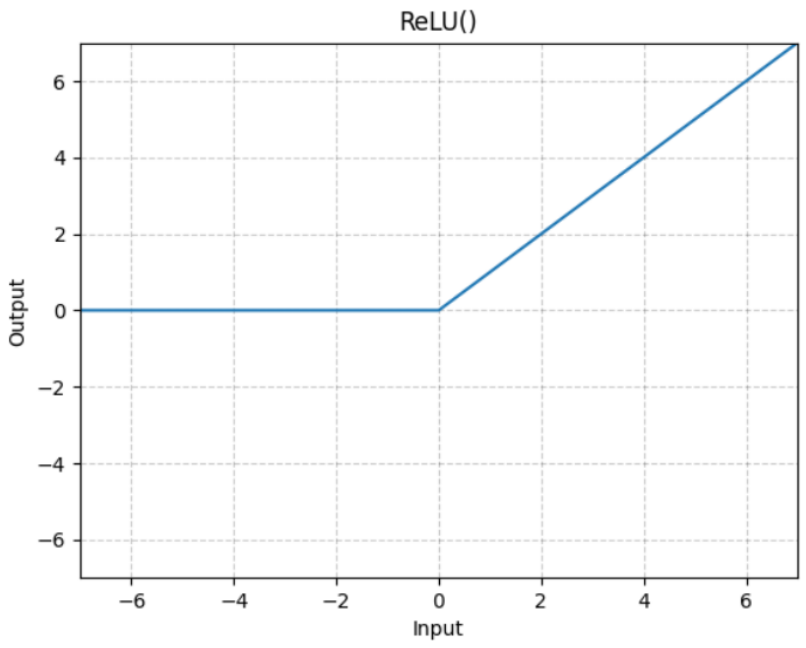


Figure 3.2: Graph of the ReLU function (taken from the documentation of [6])

The number of neurons of the hidden layer, also called hidden size, will be a hyperparameter of the model. Figure 3.1 does represent quite well the Shallow-MLP architecture, with the differences that there are 36304 inputs and only a single output. This is the simplest architecture possible for a model to evaluate whether these models perform better than bigger ones, as in the literature. Deep-MLP will be composed of nine hidden layers with ReLU activation between the layers. The structure is as follows: the first four layers have a constant hidden size, and the last four layers are half the size of the first four layers. The middle fifth layer is set to three-fourths of the initial size to smooth the decrease of the hidden size. For the sake of clarity, Figure 3.3 illustrates this architecture, where every blue rectangle represents a linear layer with its hidden size. The idea behind this architecture is to have more layers to compare with a smaller one. Decreasing the layer size progressively when we have several layers is usually a good idea. It forces, by design, the model to compress the information as there are fewer neurons to store it. Having only a single decrease enables being as general as possible by putting the minimum constraints on the choice of the hidden size: any value greater than 2 is suitable for the hidden size.

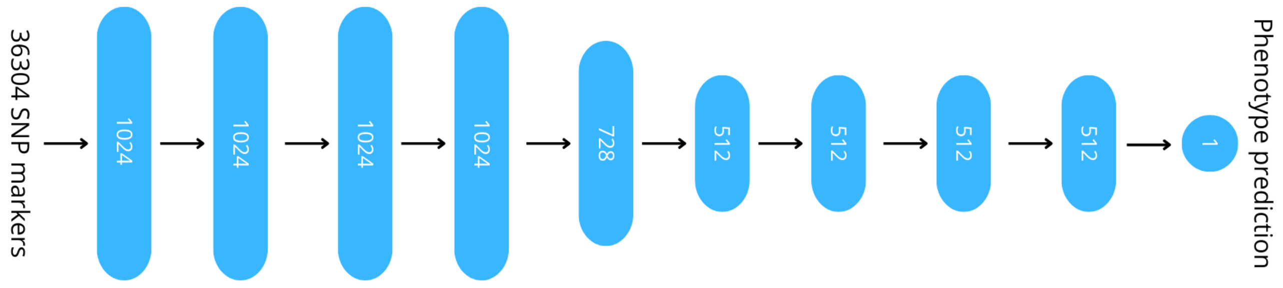


Figure 3.3: Abstract schematic of the Deep-MLP architecture with an initial hidden size of 1024

3.2.1 Experiments

As the model is very simple, there are few degrees of freedom to try to improve the results. There are only a few hyperparameters that we can tune on perceptrons. Two experiments were performed on these architectures: a test on the impact of normalization and multiple runs to determine suitable hyperparameters.

The intuition behind the normalization is that the model may struggle to handle the variance in the chosen phenotypic values from the different examples in the training set. These variances depend a lot on the chosen phenotype and may decrease the performance of the model. To evaluate that, we run two instances of Shallow and Deep-MLP on all phenotypes, the first one is trained to predict the raw phenotypic values, and the second is trained to predict the normalized target. The normalization applied consists of dividing the target by the standard deviation of the phenotypes without centering the targets.

The results for this experiment can be found in Table B.3 in Appendix B. The obtained correlations are very similar for all the phenotypes. There is also no clear better option: more or less half models achieve better results with normalization than without. As the improvement seen is less than one hundredth for the correlation, the conclusion is that normalization of the targets does not increase or decrease the quality of the results. To simplify the training, the normalization will not be applied for the second experiment for Shallow-MLP and Deep-MLP.

The second experiment will be related to the hyperparameter tuning of the model. As there are not many hyperparameters to tune, we do not have lots of degrees of freedom for the choice of the hyperparameters to tune. In this thesis, the term sweep will refer to the hyperparameter tuning using the Weights and Biases software. Based on the results of the first sweep, a second one was designed to try bigger values for the hidden size with a more restricted learning rate range. Note that this second sweep was only performed on the Shallow-MLP architecture. The selection of the hyperparameters to be tested was chosen at random from the range of possible values. The number of runs for the sweep was chosen to be big enough to select at least one time every possible value for every possible hyperparameter. The first sweep has done 40 runs per phenotype, while the second sweep will perform only 25 runs due to the smaller parameter range.

Table 3.1: Hyperparameter ranges tested for the MLP models

(a) Values for the first sweep.

Parameters	Min	Max	Step
Learning rate	10^{-3}	10^{-6}	$+\frac{(10^{-3}-10^{-6})}{10}$
Hidden size	32	8192	$*2$
Dropout	0.05	0.5	+ 0.05
Batch size	8	1024	$*2$

(b) Values for the second sweep.

Parameters	Min	Max	Step
Learning rate	10^{-2}	$5*10^{-4}$	/
Hidden size	2048	12288	$*2$
Dropout	0.2	0.8	+ 0.1
Batch size	128	1024	$*2$

The four parameters we will optimize for this architecture are the learning rate of the optimizer, the hidden size, the dropout probability (the probability that an input is randomly set to zero before every layer during the model training), and the size of the batch during the training of the model. The details of the values of the different parameters can be found in Table 3.1. The min and max columns are the bounds for the parameters, the step indicates the operation performed starting from the minimum value to the maximum one on the previous value to obtain all considered values. In Table 3.1b, the step for the learning rate is not provided. This is because the considered values are 10^{-2}, $5*10^{-3}$, 10^{-3}, and $5*10^{-4}$. These values do not have a common step that enables us to describe all the chosen values correctly. The best performing configurations are chosen by taking the best performing set of hyperparameters on the validation set. The maximal correlation obtained during every sweep can be found in Table B.4. The whole configuration selected can be found in the table A.6 and A.7 in the Appendix A.

3.2.2 Results and discussion

Table 3.2: Results of the MLP architectures on the test set

(a) Correlation

Method	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
Shallow-MLP	0.477	0.464	0.496	0.498	0.425	0.516
Deep-MLP	0.464	0.460	0.500	0.493	0.467	0.507

(b) mean absolute error

Method	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
Shallow-MLP	1.265	2.445	1.060	1.105	5.968	1.114
Deep-MLP	1.236	2.801	1.088	1.061	5.524	1.125

Table 3.2 presents the results in terms of correlation and mean absolute error of the Shallow-MLP model and the Deep-MLP model. The baseline is also added in order to put the results in the proper context. The quality of the results for the correlation depends on the phenotypes: for both buttock phenotypes, we are very close to the results obtained with GBLUP, while the results for the size phenotype are worse than those from the baseline. It is also worth noting that no MLP model was able to surpass the baseline.

The second interesting fact is that the mean absolute error was never improved by using MLPs. It could be expected that the correlation is slightly worse, as we did not optimize for the correlation during the model's training. The fact that the MLP architectures use some nonlinearities for the prediction does not seem to help in order to achieve better predictions. Finally, the results show that a low mean absolute error does not imply reaching a high correlation. The top phenotype illustrates that despite having a mean absolute error more or less 20% higher with the Deep-MLP, it can still reach very similar correlations.

These results could be explained by the fact that the perceptron architecture is not the best choice, as it takes into account all SNPs of the input for every neuron by definition. In reality, likely, most of the SNPs do not have any impact on the target phenotype and perturb the prediction by diluting all the useful information with random ones.

3.3 Locally connected neural network

The locally connected neural network is a custom network that was introduced as part of the paper [7]. Figure 3.4 illustrates the deepGBLUP model: the model presented in the paper [7]. $\mathbf{G} \in \mathbb{R}^{n\times n}$ denotes the genetic relationship matrices that will be used for the GBLUP, one for the additive effects, one for the dominance effects, and one for the epistasis effects. The values $\hat{\mathbf{b}}_a$,$\hat{\mathbf{b}}_d$, and $\hat{\mathbf{b}}_e$ represent the prediction of the GBLUP depending on the genetic relationship matrices used. $\hat{\mathbf{b}}_{deep}$ is the prediction of the neural network, and $\bar{y}_{train}$ stands for the mean of the phenotype value on the train set. The LCL block stands for locally connected layer and will be explained later in this section, as this is the idea we will reuse. The architecture can be split into two parts: the first is the neural network, and the second is the different GBLUP. The goal of the neural network is to correct the value of the sum of the GBLUP and the mean $\bar{y}_{train}$. As the neural network is the only non-linear predictor of the architecture and the GBLUP are linear

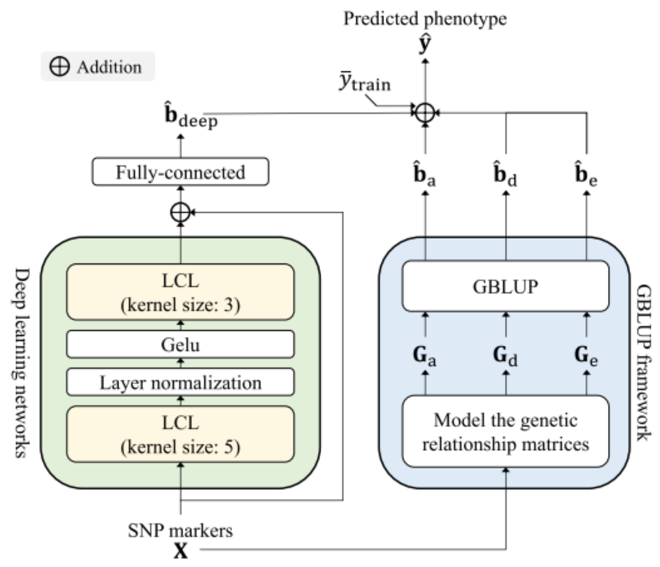


Figure 3.4: DeepGBLUP architecture (taken from [7])

models, the idea of the neural network is to learn how to correct the sum of the GBLUP to achieve better prediction, potentially by leveraging nonlinear interactions of the markers.

This model will not be tested on our dataset as we found several major flaws in the design of the deepGBLUP architecture. The first major incoherence with the model is linked to the usage of three types of GBLUP at the same time. The additive, dominance, and epistasis effects are not independent of each other. The prediction of these effects can not be simply summed together like in the deepGBLUP model. The GBLUP predictions are not parametric, which means that when performing backpropagation to update the model weights, the GBLUP predictions are considered as constants. The model is not aware of this dependence of the GBLUP predictions and can not do anything to mitigate its impact on it. The second flaw in the deepGBLUP architecture is due to the addition of the mean of the phenotype $\bar{y}_{train}$. Adding this term makes very little sense. The model is already using the predictions of three different unbiased models. The mean of the phenotype does not provide any relevant information that was not provided by the GBLUPs. Furthermore, when we look at the neural network structure, it ends with a fully connected layer. Using as an example Equation (3.1), we can see the bias term that is independent of the input. It makes no sense to add the mean $\bar{y}_{train}$ directly in the structure when there is already this learnable bias that will be adapted during the model training anyway. For all these reasons, the deepGBLUP will not be evaluated as it lacks coherence in its design. However, the authors have introduced an interesting idea with the locally connected layer in the neural part. This is what we will focus on in the rest of this section.

The locally connected layer (abbreviated to LCL) is a kind of middle point between the linear layer presented in the previous section and a convolutional layer. Instead of sharing the weights for all input like in the convolutional layer, these are now fixed for every position. Figure 3.5 illustrates a small network with two LCL layers. If we had used a convolutional layer instead of the LCL, we would have had $w_{11} = w_{14} = w_{17}$ and $w_{21} = w_{23}$. In a linear layer, we would have a connection between x_1 and all hidden neurons, and all hidden neurons would have a connection with all output neurons. This new layer enables making hidden neurons only depend on a few close neurons, like the convolutional layer, but without the sharing of the weights. Figure 3.5 also shows the impact of two hyperparameters of the layer: the kernel size controls the number of neurons we take into account in the neighborhood, like in a convolutional layer. We are

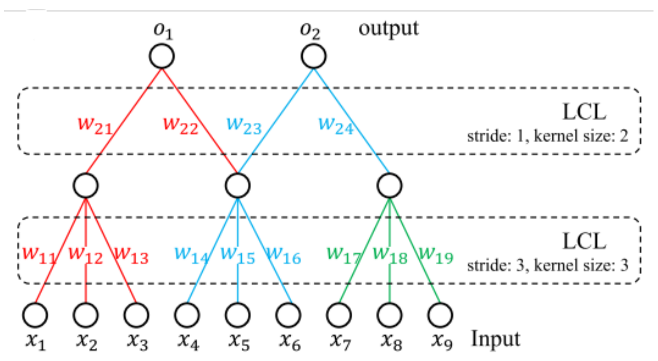


Figure 3.5: Illustration of the LCL layer (taken from [7])

thus able to control the number of neurons that are used in the computation. Something that we could not do with a linear layer, where, by design, all neurons are connected to the ones from the previous layers. The stride hyperparameter controls the step size of the kernel. We can thus decide if we accept to use a neuron several times in the computation of the neurons from the next layer (like in the last layer of Figure 3.5) or not (like in the first layer of Figure 3.5). To summarize, the LCL layer has unique properties compared to the classical linear and convolutional layers. As its properties look interesting, we will explore an architecture with this layer in the remainder of this section, and we will reuse the layer in some experiments in the convolutional models section.

3.3.1 Experiments

Starting from the architecture described in Figure 3.4 with only the neural network, several experiments were performed to evaluate the best architecture. The parameters modified were the kernel size, the stride, and the number of LCL layers, the structure of the MLP (number of neurons for each layer, number of layers). The ReLU activation function was also tried instead of using the GeLU. Dropout layers have been introduced before every weight layer. Dropout layers are layers that set the inputs with a given probability to zero during training. This probability is a hyperparameter to tune according to the architecture and dataset. The addition of such layers prevents overfitting by preventing the model from always looking at the same features and thus forgetting about the other ones [62]. Figure 3.6 shows the best architecture

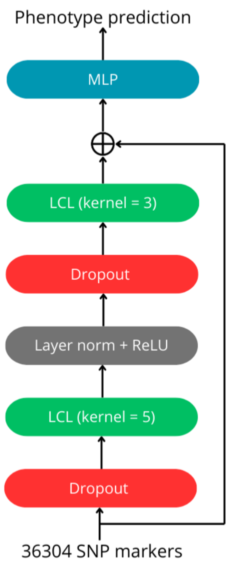


Figure 3.6: Architecture of LCLNN

according to the performed experiments. To avoid any confusion, we will call this model LCLNN (Locally Connected Layers Neural Network). The best configuration is composed of two LCL layers with a kernel size, respectively, of 5 and 3 and a stride of 1. The dropout probability is the same for all dropout layers. Compared to the architecture shown in Figure 3.4, the only differences are the activation function that changes from the GeLU to the ReLU and the addition of the dropout layers before the two LCL layers. The structure of the final MLP can be tuned depending on the phenotype: the number of hidden layers and the number of neurons per layer are not fixed. The experiment shows that one or two layers are the best-performing number of layers. The activation function used in the MLP is also the ReLU.

Table 3.3: Hyperparameters evaluated for the LCLNN architecture

Parameters	Min	Max	Step
Learning rate	10^{-5}	10^{-3}	-
Hidden size 0	64	2048	$* 2$
Hidden size 1	0	64	$* 2$
Dropout	0	0.5	+ 0.1

To find the best configuration for every model, a sweep will be performed with the ranges described in Table 3.3. The step for the learning rate is missing in the table because it is not constant. The tested values for the learning rate are 10^{-3}, $5{*}10^{-3}$, 10^{-4}, $5{*}10^{-4}$ and 10^{-5}. Hidden size 0 refers to the number of neurons of the first hidden layer of the final MLP. Hidden size 1 refers to the number of neurons of the second hidden layer. If the hidden size 1 is equal to zero, it means that the MLP is only a one-hidden-layer MLP instead of a two-layer MLP. The smallest size for the second hidden layer is 2. The maximum value for the hidden size 1 was chosen to be the minimum size of the hidden size 1 such that the second hidden layer is never bigger than the first layer. Note that the batch size used was 256 for every phenotype. The best configuration for every phenotype was chosen among the 30 runs performed with a random choice of the combination of the hyperparameters to evaluate. These configurations can be found in Table A.8. Based on these results, a second sweep could have been done, as many best configurations use either a lower or a higher bound for the parameters, which could indicate the range was not optimal for this architecture. Unfortunately, this could not be done due to time constraints that would prevent doing a whole new search for every parameter.

3.3.2 Results and discussion

Table 3.4: LCLNN models results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
LCLNN	0.464	0.470	0.502	0.498	0.473	0.505

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
LCLNN	1.704	3.085	1.368	1.085	5.394	1.404

Table 3.4 summarizes the results for this model. For the correlation metric, the results are similar or slightly worse compared to GBLUP. For the mean absolute error, the performance of the LCLNN depends

a lot on the phenotypes. The gap varies from a few hundredths for the Buttock (rear) phenotype to six tenths for the top phenotype. Furthermore, the two metrics seem to have very little correlation. If we take the Top and the Buttock (side) phenotypes, the gap between LCLNN and the baseline is almost equal for the correlation, but for the mean absolute error, the gap is 0.6 for the Top phenotype against 0.3 for the Buttock (side). This indicates that the choice of the loss function is not optimal. As we optimize on the mean absolute error to a local minimum, we would like to also find a local maximum for the correlation. Maybe the usage of a new type of loss that would have a higher correlation with the correlation metric could help to find better results, as the local minimum of this new loss function would be closer to a maximum in the correlation metric.

3.4 ResGS

This section will present the first convolutional neural network (CNN for short). These architectures are designed to process images by enforcing some specific properties to mimic the way humans see. The three properties that CNNs enforce are the locality, the invariance to translation, and the hierarchical structure: the outputs from simple cells are processed by higher-order cells, which are more complex and capable of extracting information [9]. These properties, called inductive biases, can not be enforced by a modification of the loss function. These properties should be included in the neural architecture itself. Convolutional neural networks are the first successful example of neural networks built using inductive biases (the prior knowledge of the problem) to construct a powerful but specific model.

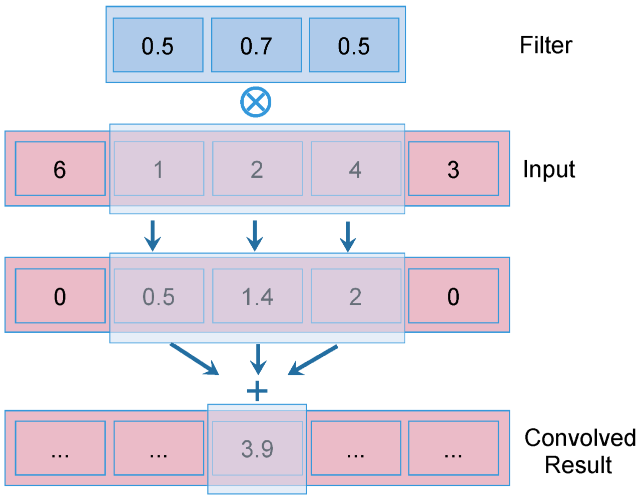


Figure 3.7: Example of a 1-D convolution (taken from [8])

Let us begin the introduction of the main concepts of a CNN by defining the convolution operation. To avoid any confusion, we will only present here the one-dimensional convolution, as this is the one we will use in this thesis. The concept can be expanded to 2-D inputs like images very easily. Figure 3.7 illustrates the convolution operation. We need a sequence as input and a kernel (called a filter in the figure). The convolution consists of multiplying the input element-wise with the kernel and then summing the elements to find the convolved results. This operation can be done again by shifting the kernel by a defined step size (called the stride). The mathematical definition of the convolution is [9]:

$$\mathbf{x} \circledast \mathbf{k}[i] = \sum_{m=0}^{w-1} \mathbf{x}_{m+i}\mathbf{k}_m \tag{3.2}$$

Where $\mathbf{x} \in \mathbb{R}^n$ is the input sequence, and $\mathbf{k} \in \mathbb{R}^w$. The resulting vector $\mathbf{x} \circledast \mathbf{k}$ will contain exactly $n-w+1$ elements. Note that Equation (3.2) defines the cross-correlation operation, but this is the operation called convolution in the machine learning field. This operation satisfies the locality inductive bias. The result only takes into account w values from the input, where w is the size of the kernel. The convolution is also equivariant to translation: if the input is shifted, then the output is also shifted by the same amount. This is because the weights of the kernel do not depend on the position of the input, as they are shared for the whole input. We can expand Equation (3.2) such that **k** and **x** do not necessarily contain a single kernel or a single input. In this thesis, instead of dealing with vectors, we will deal with matrices. The number of kernels/inputs is called the channels. The application of several kernels may help later to construct more complex, higher-order features. Having more than one input enables having more complex inputs. The most well-known example of channels is the colored images. A grayscale image has only one channel, the nuance of gray of each pixel. A colored image is composed of three channels of the size of the image that define the intensity of the red, green, and blue colors, enabling it to have a much more complex color scheme than a single-channel grayscale image.

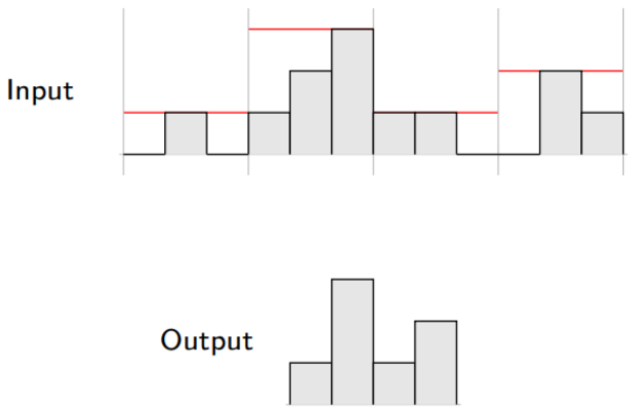


Figure 3.8: Example of 1-D max pooling (taken from [9])

To satisfy the remaining properties, we need to introduce the pooling operation. The goal of this operation is to reduce the size of the input to be able to introduce a hierarchical structure and to ensure invariance to translations. There are two types of pooling: the max pool and the average pool. Figure 3.8 illustrates an example of the max pool operation. First, we define a size for the pooling and delimitate the input of non-overlapping regions of the given size (in Figure 3.8, the size is equal to three). Then, for every region, we performed the aggregation operation. In this example, we take the maximum value of all values in the region. If we had used the average pooling, we would have taken the mean of the values of the region. Such an aggregating operation reduces the size of the input by a factor equal to the size of the pooling. This compresses the information and enables building a hierarchical structure by alternating convolutional layers with pooling layers. This operation also provides local invariance within the regions: whatever the ordering of the value, the results of the max operation or average operation will still be the same. This does not lead to a total invariance to translation, but it is considered enough for our problems.

As shown in Section 1.4, these types of architectures were tested as it is for genomic selection. However, when we use architectures built with inductive biases, it is important to verify that these biases are coherent with the problem we want to solve. As we have little information on how the genes impact the phenotypes, the hierarchical structure hypothesis is neither confirmed nor denied. Using hierarchical architectures may help to find answers according to the results. For the locality hypothesis, there are no indications that the markers important for a phenotype are close to each other, but testing this hypothesis may be interesting to evaluate if this is true or false for the studied phenotypes. For the invariance to translation, this bias does not make much sense for the genomic selection problem. Let us remember that the sequence used as input is the SNP markers of an individual. If the marker's values are shifted between two individuals, it means that these individuals are completely different and should not be considered equal by the model. This questions the usage of the pooling layers, which introduce this translation invariance in the architecture. However, we still need a reduction method to be able to use a hierarchical structure.

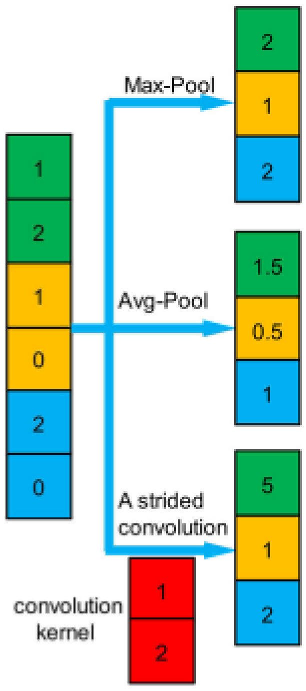
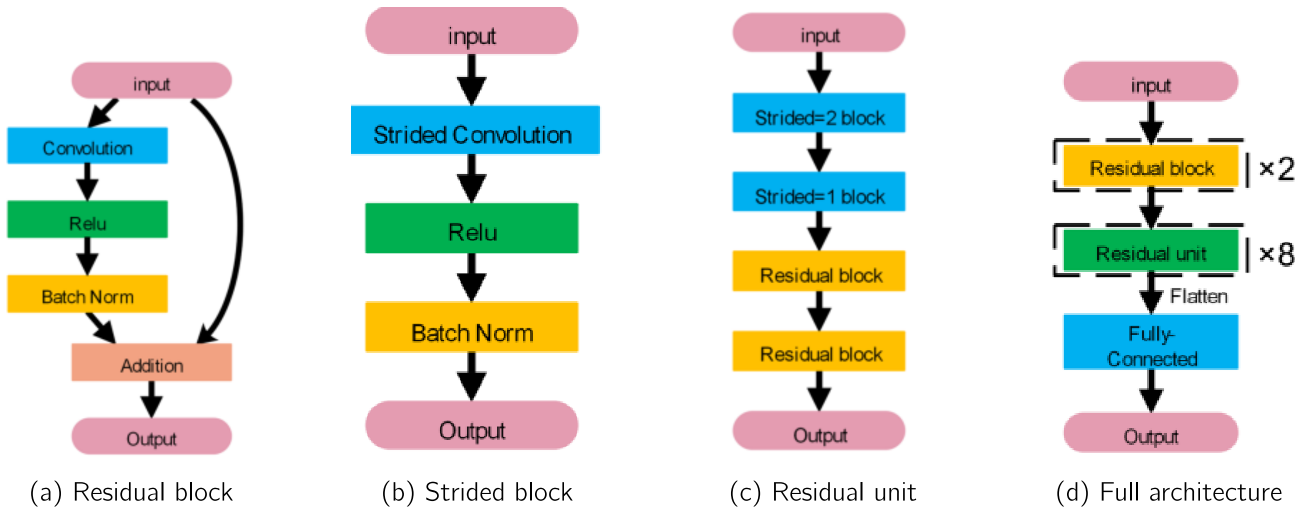


Figure 3.9: Comparison of a strided convolution and pooling operations (taken from [10])

An idea is to use a strided convolution instead of a pooling layer [10]. The size of the pooling is equal to the size of the kernel and the step size (the stride parameter in the Pytorch implementation). Figure 3.9 compares the three methods. It can be seen that for the pooling methods, the position of the markers does not matter, but for the strided convolution, it is not the case because the markers are multiplied by the kernel values. If those are different, then the marker position matters, which destroys the invariance to the translation while still reducing the size of the input.



(a) Residual block (b) Strided block (c) Residual unit (d) Full architecture

Figure 3.10: Illustration of ResGS architecture (taken from [10])

The ResGS architecture was proposed in the paper [10] to test if the pooling layer has a bad impact on the model predictions. Another advantage is that the proposed architecture is composed of many layers and uses residual connections. Testing this architecture will also enable us to test whether the residuals link combined with many layers can perform well. Figure 3.10 illustrates the architecture with an abstract block view. As the proposed architecture was more than 35 layers deep, it would have been impractical to reproduce the full architecture on a single figure. Figures 3.10a and 3.10b are the basic building blocks of the architecture. The strided block is the same as the residual block but without the residual connection. This block will be used to reduce the dimension of the input. Figure 3.10c is an intermediate building block composed of the previous building block. As the stride parameter is equal to two in the first block, the residual unit begins by reducing the size of the input by a factor of 2. The second strided block has a kernel size of one and aims to adjust the number of channels to the targeted value. Finally, the full architecture proposed in the paper is defined in Figure 3.10d. Between the last fully connected layer and the MLP, there is a convolutional layer that is used to update the number of channels of the intermediate results to control the size of the input layer of the MLP, whatever the number of residual blocks. The targeted input size was 6400 in the proposed architecture. Because this parameter was considered crucial by the authors, several values were tested during the hyperparameter tuning to find the most appropriate value for our dataset.

3.4.1 Experiments

The first experiment consists in defining the best number of residual units. Models with 4, 8, and 11 residual units were tested, and it was the model with 11 residual units that performed the best. Note that it was not possible to test adding more residual units without changing the structure. Indeed, as we divide the size of the inputs by two at each iteration, it is not possible to add more layers, as it would shrink the input to a value smaller than the kernel size. The second experiment aimed to expand the MLP at the end of the model. The impact of the addition of a dropout and of the addition of a hidden layer where considered. The last experiment was about the choice of the strided convolution to reduce the input size in the architecture. One big flaw of paper [10] is that the authors claim their solution is better than using max pooling layers, but no experiment was performed to validate the claim. This test will thus be performed in this thesis. We will consider three models: the first will be as defined in the paper (and then use the strided convolution), the second will be the max pooling, and the third will use a strided LCL layer. Using the same kernel for all positions may not be the best solution, and the LCL provides an alternative that has weights that depend on the position. The only problem is that we have to reduce the number of output channels when using the LCL layers as the model has more than 2,6 billion parameters instead of a few tens of millions, which makes it impossible to run on the available hardware. The number of channel layers was reduced to the biggest values that would fit in the hardware normally assigned to the ResGS model, and it still has more than 4 times more parameters than the first model that uses the strided convolution. Note that the three models did not follow the training procedure from the following subsection; the models are trained to predict the whole phenotype, like all models presented in the previous section of the report.

Table B.5 summarizes all results from the three models. As the implementation of the LCL layers was based on the code of [7] and expanded with the help of [5], the implementation of this third model is very slow, as it is not optimized. As a consequence, the model was only trained on the first phenotype. Results show that the model that used the strided convolution performed way better than the max pooling. This indicates that the architecture is penalized when using the max pooling. This is very important as it questions all CNN architectures from the literature that use max pooling. Their model may be constrained by the usage of max pooling, and a well-performing CNN may be improved by using strided convolutions instead. For the LCL, the performance is similar to the one obtained with the usage of max pooling. It is not possible to draw any conclusion based only on these results. Maybe it is caused by a lack of data (as the model has four times more parameters, the size of the dataset may be insufficient to train the model efficiently. The reduction of the number of channels caused by the explosion of the number of parameters could also explain the performance: the reduction is too big, with a loss of information due to the compression in the few channels. All these are only hypotheses that may be worth exploring to better understand how the architecture interacts with the dataset.

3.4.2 Training procedure

[10] also introduced a different training procedure. Instead of using the phenotypic value as the target to learn, this model follows a two-stage training. The first stage consists in selecting the best model from the following algorithms: ridge regression, SVM, random forest, and Gradient boosting (an alternative implementation of the algorithm of XGBoost). These models are trained on the phenotypic value, like the ones presented in the previous chapter. The best of the four models is selected to be used in the second stage. It will consist of training the ResGS architecture to predict the error made in the first stage. The final prediction will be composed of the prediction from the first model + the prediction of the ResGS model. Instead of training the neural network to predict the phenotype, it is trained to predict the residual phenotypic value from the chosen model in the first phase. In this thesis, we will replicate the defined training pipeline but with a small adjustment. In the first phase, instead of choosing the best model, we will directly choose the ridge regression model. The reason behind that choice is that the ridge regression model is the only linear model of the four proposed in the paper. We can justify using the ResGS in the second phase to account for the non-linear effects that the ridge regression may not have been able to predict. For the nonlinear model, the usage of the ResGS for the second phase assumes the ResGS is a better predictor than the one used in the first phase, which should have been proved before proposing this training pipeline. As we have no prior indication that an algorithm should perform better than another one, using such algorithms is not justified. The hyperparameters for the ridge model used in the training of the ResGS are the ones defined in Table A.2 from the experiments explained in Section 2.3.

The impact of this new training procedure was tested. Table B.6 summarizes the results. The line "whole phenotype" means that the target is the phenotype value, while the line "with residuals" means that the ResGS model is trained on the residual from the ridge model. The results are very clear: the training pipeline gives way better results than without. We will thus continue using this training pipeline to train the ResGS model.

3.4.3 Hyperparameters

Table 3.5: Hyperparameters range for the ResGS model

Parameters	Min	Max	Step
Learning rate	10^{-5}	10^{-2}	/
Hidden size	32	2048	$*2$
Dropout	0.1	0.4	+ 0.05
Target size	2400	12400	+ 2000

Concerning the model hyperparameters: Most of the hyperparameters that were not discussed in this paper were taken from [10]. The hyperparameters chosen to be tuned and their values are defined in Table 3.5. The step for the learning rate is missing in the table because it is not constant. The tested values for the learning rate are 10^{-2}, $5 * 10^{-3}$, 10^{-3}, $5 * 10^{-4}$ and 10^{-5}. To be coherent, the value 10^{-4} should have been included, but was missing due to an error in the configuration file. The hidden side refers to the number of neurons in the hidden layer of the MLP at the end of the model. The biggest value for the hidden size is set to be bigger than the smallest possible value for the target size hyperparameter. Dropout layers that are affected by the dropout probability are only present in the MLP. The target size refers to the targeted number of input neurons after all the convolutional layers. The range was defined to try a widespread range of values centered on the default value provided in the paper [10]: 6400. The hyperparameters were selected at random for every run of the sweep. The number of runs performed for the tuning was set to 20, but in practice, only 16 were made for the size phenotype and 14 for the musculature phenotype due to a timeout occurring in the scripts. The value 20 was chosen based on the maximum range for the hyperparameter, such that the random search selects at least every possible value. The best configuration is defined in Table A.9.

3.4.4 Results and discussion

Table 3.6: ResGS models results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
ResGS	0.482	0.477	0.505	0.497	0.492	0.525

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
ResGS	1.209	2.409	1.057	1.015	5.349	1.105

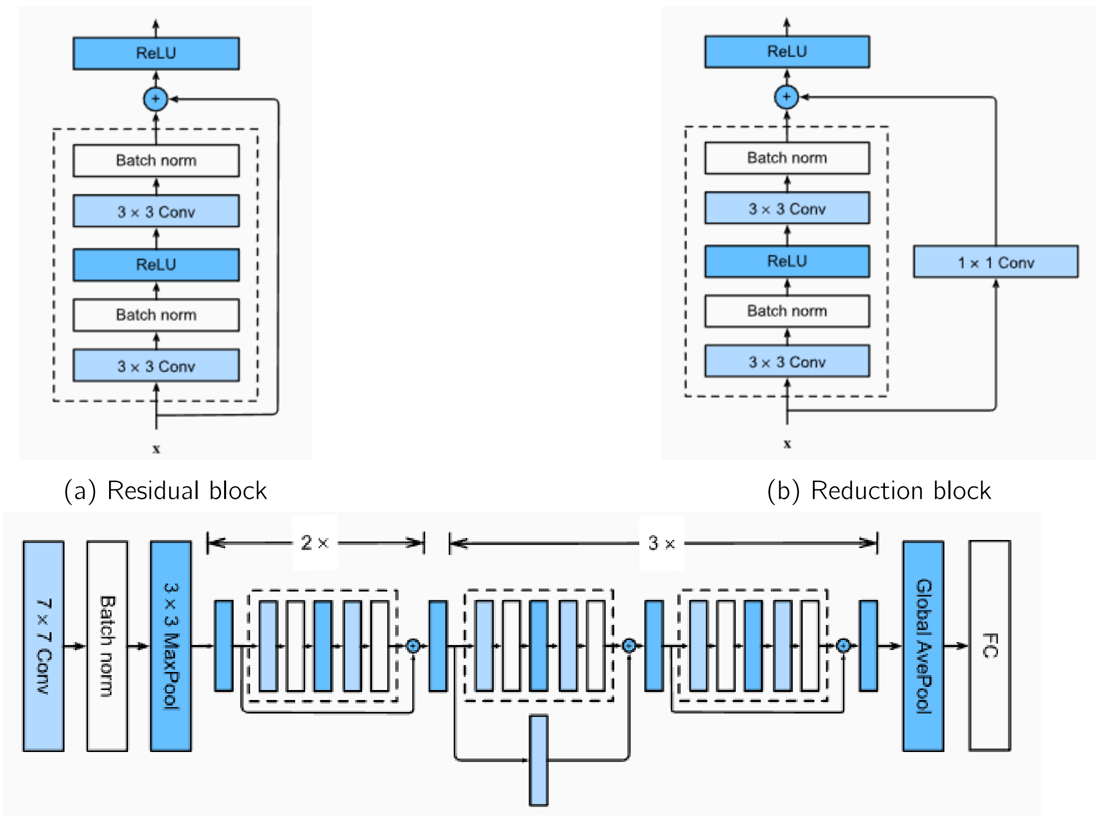
Table 3.6 summarizes the results from the ResGS models. For both metrics, the performance of the ResGS is similar to that of GBLUP. Some improvement can even be observed. For the size and musculature phenotypes, the ResGS is better and achieves the same results up to three digits for the top phenotype. For the mean absolute error of the ResGS predictions, only the one for the top phenotype is lower than that one from the baseline. ResGS is not way worse or better than the baseline, but both models have very similar results. The results are too close to claim the ResGS is better or worse than the GBLUP.

3.5 Custom ResNet

The ResNet architecture is one of the most common CNN architectures [63]. This architecture showed very good results on the image classification task and became one of the default architectures for a CNN. The architecture used in this thesis will be inspired by the ResNet one, but some modifications will be performed. The next section will introduce the ResNet architecture as proposed in the original paper. The modifications made to the original architecture and the experiments performed on the models will be detailed in Section 3.5.2. The final section will describe the results of the best-performing models using the proposed architecture.

3.5.1 ResNet: introduction

Figure 3.11 illustrates the ResNet architecture. Like in the previous section, we have a residual block shown in Figure 3.11a. The difference with the ResGS residual block presented in Figure 3.10a is that instead of having one convolutional layer before the summation with the residual link, we have two convolutions. The equivalent strided block of the ResGS (shown in Figure 3.10b) is shown in Figure 3.11b. In the ResNet, the first convolution layers double the number of input channels and divide the size of the input by two via the usage of a stride of two. The number of channels is then kept constant. Unlike in the ResGS, this block is also a residual one, there is a need for a convolution with a kernel of size one to adjust the number of channels such that the dimensions are the same when we add the residual link with the output



(a) Residual block

(b) Reduction block

(c) Full architecture of ResNet-18

Figure 3.11: Illustration of ResNet architecture (taken from [5])

of the block. Finally, Figure 3.11c shows the full architecture of a ResNet. The architecture of the figure is designated as ResNet-18, as there are a total of 18 convolutional layers in this architecture. Note that there are no reduction blocks before the first residual units, as the max pooling already has a stride of 2 that reduces the size of the input. There is thus no need for a reduction there. The ResNet architecture is very modular, as it is very easy to modify. The two main ways to modify the model are by updating the number of residual blocks after a reduction block. In this example, we have two residual blocks after the first reduction and only one after the next reduction block. The second option is to update the number of reduction blocks. In ResNet-18, we have four reductions, but we can imagine using fewer reductions. Note that in the paper [63], they proposed several architectures, and none of them has more than four reduction blocks as their inputs were small, and adding more reduction layers could compress the information too much. This subsection has presented the architecture proposed in the paper [63]; the modifications and experiments done in this thesis will be explained in the next subsection.

3.5.2 Modifications and experiments

Table 3.7: Summary table of our ResNet architecture variations.

Architecture	1	2	3	4	5
Residual block	2	1	2	2	3
Reduction block 1	-	1	2	2	4
Reduction block 2	-	1	-	2	6
Reduction block 3	-	1	-	2	3

The goal of this architecture is to test a second CNN architecture. Based on the results from Table B.4, where the smaller model performed better than the bigger one, and the fact that ResGS is a very complex model, this second architecture aims to investigate the impact of a smaller CNN. This architecture will not be a ResNet like presented in [10] but a custom one. However, we will still call these models ResNet at the end of this document.

The main difference is that we will only consider four blocks in the architecture to reduce the size of the model. The first block will be a residual block (like described in Figure 3.11a) as we keep the same structure before the first block. This first block will be followed by at most three reduction blocks, depending on the architecture. To play with the size of the model, we will change the number of convolutional layers inside the block. Table 3.7 shows the five variations used during this thesis. The value stores the number of convolutional layers in each block. Note that for the reduction blocks, the convolutional layer that reduces the size of the input is not included in the table. The base architecture is the fourth and the fifth. The chosen values are based on the number of blocks from ResNet 18 and 34. Architectures 1, 2, and 3 are smaller variation that plays on the number of convolutional layers per block and the number of reduction blocks of the model. The number of channels and the kernel size are the same as the ones described in the paper [63].

The first main experience done with this architecture was to test the impact of the max pooling on the model's performance. To do so, we will test three different aggregation versions. The first one will use the max and average pool, as described in Figure 3.11c. The second will use a strided convolution instead of the max pooling with a kernel of size three, like the max pool and the average pool will be replaced by a convolutional layer of a kernel of size one and a single output channel. The third and last version will use the same strided convolution as in the second version to replace the max pooling, but the average pool will be replaced by an LCL layer with a kernel of size one and a single output channel. Only the average pooling layer will be replaced by the LCL to avoid an explosion of the number of parameters, like the one observed with the ResGS models. These three aggregation versions will be called, respectively, pooling, strided, and strided + LCL in the Table B.7 that summarizes the results obtained using the fourth and the fifth architecture described in Table 3.7. We can observe that the strided aggregation performs better than the other two aggregations in both tested architectures. The strided + LCL aggregation yields slightly better results in some phenotypes than the pooling aggregation, and was not tested using the fifth architecture. These results are similar to the ones obtained with the ResGS model: they indicate that the usage of pooling layers decreases the quality of CNN when applied to genomic selection.

The second experiment aimed to study the impact of the encoding of the genotype. Two encoding versions were developed. Both begin to encode the genotype with one-hot encoding and then flatten the result to obtain a big vector of $36304 * 3 = 108.912$ values. The first encoding, called convolutional, will apply a convolutional layer with a kernel of size 3 and a stride of 3 to obtain a final vector of size 36304. As the kernel weights are shared and the input is a one-hot vector, the result of the encoding consists of modifying the 0,1,2 values by the learned kernel weights. It will enable us to evaluate if the fact that the input was fixed to 0, 1, or 2 influences the results. The second encoding, called LCL, will use an LCL layer instead of a convolutional layer. This will avoid the sharing of the weight: the resulting vector will not be composed of only 3 values, like in the convolutional encoding, but each marker will have its own set of possible values. It will enable giving more or less weight to some markers where this is not possible with the convolutional encoding. The fourth architecture with the strided reduction method will be used to compare the impact of the encoding. Table B.8 summarizes the results. It shows that the usage of the encoding does not improve the results of the models, indicating that the initial encoding does not negatively impact the performance of the model.
The third big experiment performed was the usage of learning rate schedulers. Learning rate schedulers are used to update the learning rate based on a given policy. For the ResNet architecture, the best model

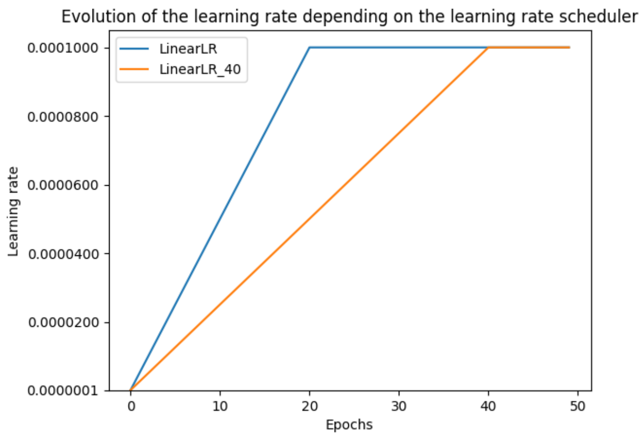


Figure 3.12: Illustration of the two learning rate scheduling policy tested

was obtained after very few epochs. The idea of the experiment was to use a linear learning rate scheduler to gradually increase the learning rate in the first epochs to slow the training to avoid making too big updates. The idea of this experiment came from [47]. Figure 3.12 shows the different learning rate schedulers we have used in this thesis. The first one, called LinearLR, increases the learning rate from 10^{-7} to 10^{-4} in 20 epochs and then keeps the learning rate constant. The final value is 10^{-4} as it is the chosen learning rate for the architecture. The second one, called LinearLR_40, has the same start and end value but is slower as it will take 40 epochs to reach the maximum value. The usage of learning rate helped to achieve better performance, as the best models for three out of six phenotypes will be trained with one of the presented learning rate schedulers.

Finally, some smaller experiments were performed. Like in the previous architectures, some experiments were made on the ending MLP with the addition of a hidden layer of various sizes. To prevent the model from overfitting too much, the introduction of a nonzero weight is introduced to the Adam optimizer. To do so, we used the AdamW optimizer that modifies the Adam optimizer to properly include the weight decay [64].

The best results obtained during the experiments are summarized in Table B.9. As the results are not as good as the previous architectures, no hyperparameter tuning was performed on this architecture. For all architectures, the weight decay was set to 10^{-4}, the strided aggregation was used, and the best architecture was architecture one from Table 3.7. The best configuration only differs in the number of neurons of the hidden layer for the MLP at the end of the model and the learning rate scheduler used during the training. The values used for every phenotype can be found in Table A.10.

3.5.3 Results and discussion

Table 3.8 summarizes the results from the best configurations. For the mean absolute error, it can be seen that the architecture results are very close to those of the baseline, except for the size phenotype, where there is a huge gap. When we look at the correlation metric, we can see that the results are not as close as those from the mean absolute error. For the size phenotype, despite the gap in the mean absolute error, we do not observe that the correlation is worse than for other phenotypes. In conclusion, the ResNet architectures failed to achieve better or similar results than the baseline.

Table 3.8: ResNet models results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
ResNet	0.435	0.427	0.475	0.450	0.437	0.471

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
ResNet	1.289	2.483	1.077	1.061	6.055	1.155

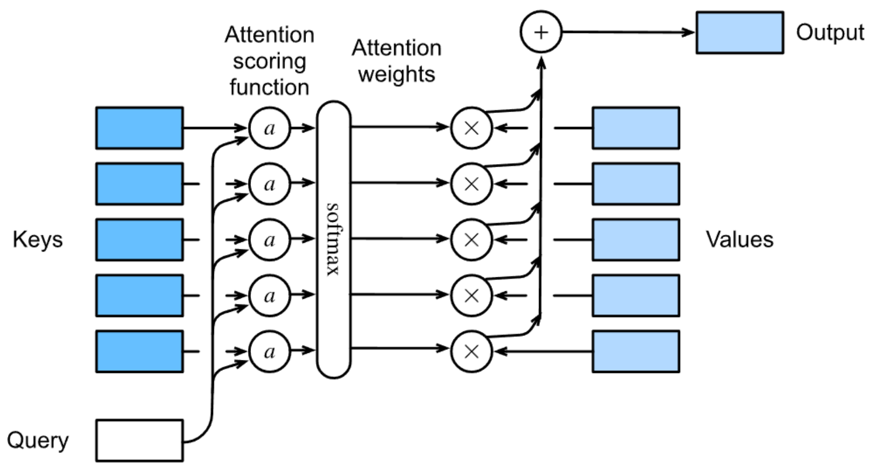


Figure 3.13: Illustration of the attention operation (taken from [5])

3.6 GPTransformer

Transformers are the third big family of neural architecture that became very popular since its introduction in [47]. This section will briefly introduce transformers in the next subsection. Then the modification made to the models will be presented along with the experiments done during this thesis.

3.6.1 Transformers: introduction

The basic concept used in transformers is self-attention. Figure 3.13 illustrates the attention operation. The query, keys, and values are sequences used as inputs to the operation. In the specific case of self-attention, these three sequences are the same sequence. The keys and the query are the input of an attention scoring function. In this thesis, the only function we will consider is the scaled dot product attention. Let be $\mathbf{K} \in \mathbb{R}^{n_k \times d}$ the keys matrix,$\mathbf{Q} \in \mathbb{R}^{n_q \times d}$ the query matrix, and $\mathbf{V} \in \mathbb{R}^{n_v \times d}$ the value matrix. n_k, n_q, and n_v respectively express the length of the keys, query, and values sequences. d is the size of a single element of the sequence. The equation (3.3) expresses the formulation of the scaled dot-product attention:

$$a(\mathbf{K}, \mathbf{Q}) = \frac{\mathbf{KQ}^T}{\sqrt{d}} \tag{3.3}$$

The dot product is a measure of similarity between the keys and the query. The closer the vectors, the bigger the dot product will be. The results of this function are then normalized by the softmax operation to provide the attention weights. These are then multiplied by the value to provide the output of the attention operation. The mathematical equation of scaled dot-product attention is :

$$\mathbf{Y} = Softmax\left(\frac{\mathbf{KQ}^T}{\sqrt{d}}\right)\mathbf{V} \tag{3.4}$$

The scaling by a factor $\sqrt{d}$ avoids that the dot-product takes too big values when d is large. If the values are too large, the gradient of the softmax of the dot product will be too small, which will impact the training negatively. [47] proposed an example to illustrate this problem that justifies the scaling: if $\mathbf{K}$ and $\mathbf{Q}$ are independent variables with zero means and unit variance. The dot product will have a zero mean but a variance equal to d. Equation (3.4) is very similar to that of a linear layer: $\mathbf{y} = \mathbf{Wx}$. Instead of having a fixed set of weights, the attention operation computes them based on the keys and the query, which enables it to adapt to the input sequence. However, the transformer architecture does not use the scaled dot-product attention like introduced here. It uses what is called multi-head attention. Let $\mathbf{h}_i$ $\forall i \in [1, N]$ the i^{th} head, its definition is:

$$\mathbf{H}_i = f(\mathbf{W}^i_q\mathbf{Q}, \mathbf{W}^i_k\mathbf{K}, \mathbf{W}^i_v\mathbf{V})$$

Where f is the scaled dot product attention and $\mathbf{W}^i_q$, $\mathbf{W}^i_k$, and $\mathbf{W}^i_v$ are the weight matrices of the head i associated respectively to the query, the keys and the values. All the head results $\mathbf{h}_i$ are then concatenated and multiplied by a final weight matrix to produce the final output. To summarize, the multi-head attention has several head that each performs the attention operation. It enables focusing on several different properties for a given input, which helps to create complex models.

It is also very important to mention the major disadvantage of the scaled dot-product attention: its time complexity. In Equation (3.4), there are two matrix multiplications: the first computes $\mathbf{KQ}^T$, which creates a $n \times n$ matrix. The second matrix multiplication multiply this previous matrix with $\mathbf{V}$. The complexity of the attention is $\mathcal{O}(n^2d)$. This quadratic complexity in function of n is a problem. In this thesis, the sequence length is very big: 36304 markers. The attention operation will thus be slower than the other architectures, which will prevent us from doing as many experiments as we wanted.

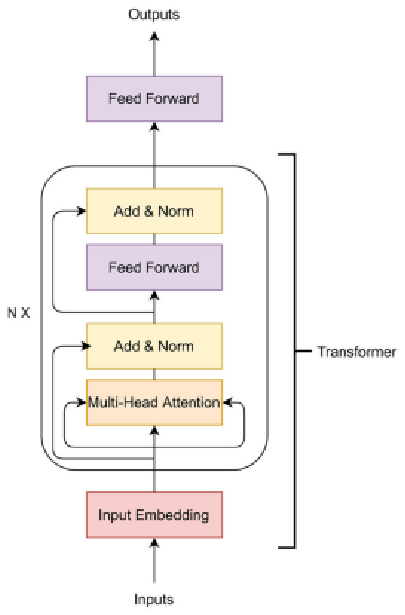


Figure 3.14: Decoder only transformer architecture (taken from [11])

The architecture we will use in this thesis is the decoder-only transformer. Figure 3.14 shows the details of the architecture. Like most of the previous architectures, transformers are composed of several blocks. Each block is composed of two parts: the multi-attention part and the MLP part. These two parts follow a residual structure: after each layer, the residual link and the layer output are recombined before the next part. The input embedding layer is used to convert the discrete elements of the sequences (here 0, 1, or 2) to vectors of the size d, called embedding. If the embedding does not take into account the position of the element, we add a positional encoding to the embedding. As the whole sequence is processed at the same time, the model does not have any information about the position of each element. The positional encoding is used to solve this problem. In this thesis, we will use the sinusoidal positional encoding proposed in [47]. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ represent the n embeddings of size d of the elements of the sequence: the sinusoidal positional encoding $\mathbf{P} \in \mathbb{R}^{n \times d}$ is defined by:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right)$$
$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right)$$

Where $p_{i,2j}$ is the element of the i^{th} and $2j$ column of the $\mathbf{P}$ matrix. The output of the input embedding block is then defined as $\mathbf{X} + \mathbf{P}$.

3.6.2 Input embeddings and architecture definition

As a starting point, we will use the hyperparameters from [11]. It uses two transformer blocks. For each block, the multi-head attention is set to have two heads, and the feed-forward block is a one-hidden-layer MLP with 256 neurons in the hidden layer. In this paper, they use a feature selection method called mutual information on the whole dataset to reduce the number of markers considered for the training and thus avoid the problem described in the previous section. A high mutual information means a high dependency of the phenotype value on the markers. If the mutual information is zero, then the marker and the phenotype value are independent. The threshold value was set to 0.02 in the paper. Using the implementation from scikit learn [49], this threshold reduced the number of markers to more or less 2000 markers. As the reduction seems satisfactory, this threshold was kept in this thesis. Reusing the mutual information will also enable us to evaluate the impact of feature selection on the predictions. Other hyperparameters are chosen to be similar to those from the model of [11].

The input embedding is a key layer of the transformer as it defines how the input is transformed before entering the model. Learning how to define the best embedding layer is then very important, as a bad choice will decrease the performance of the model. In this thesis, we will consider five different types of input embeddings. Two of them were proposed in the paper [11]. The first one uses a linear layer. These layers have n input neurons, one for every selected marker, and n*d output neurons. This vector is then reshaped to form a matrix with n rows and d columns. Note that the paper centers the encoding for the genotype: instead of using 0, 1, and 2, the values are set to -1, 0, and 1. We will call this embedding type the categorical one. The second input embedding (named the frequency embedding in this work) still uses the linear layer, but instead of encoding the values to -1, 0, and 1. They will use a different encoding based on the minor allele frequency already introduced in Section 1.3. The values for the marker i will be $(1 - p_i)^2$, $2p_i(1 - p_i)$, and p_i^2. The encoding will use the probability of the SNP at locus i. According to the paper [11], this encoding improves the results as it avoids using zero.

Figure 3.15 illustrates the categorical encoding explained before. The authors claimed that in the output neuron, the weights of the input neurons encoded as 0 are not taken into account in the computation. They have introduced the frequency embedding to avoid having zeros in the encoding. Indeed, the probability p_i will never be zero. If this is the case, it means that either p_i or $1 - p_i$ is equal to zero, and the locus would not be a SNP, as there would only be a single allele. The results of their experiments

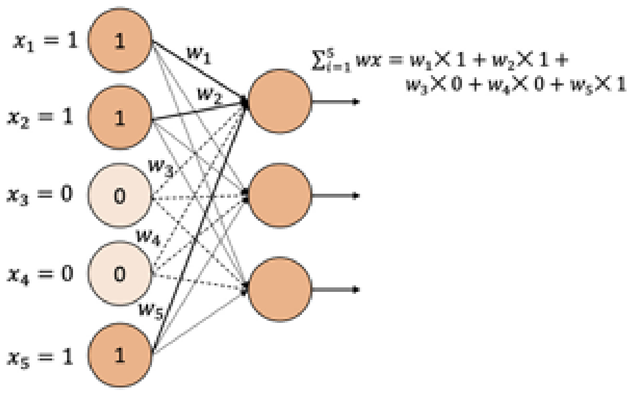

$x_1 = 1$
$x_2 = 1$
$x_3 = 0$
$x_4 = 0$
$x_5 = 1$
w_1
w_2
w_3
w_4
w_5
$\sum_{i=1}^{5} wx = w_1 \times 1 + w_2 \times 1 + w_3 \times 0 + w_4 \times 0 + w_5 \times 1$

Figure 3.15: Illustration of the categorical encoding (taken from [11])

show that the alternative encoding leads to better results on their dataset. In this thesis, we will redo the experiment to see if their claim is still valid using another dataset. Note that as a linear layer is used, there is no need to include a positional encoding. Indeed, each neuron has its own set of weights, and the generated embeddings will take into account the position of the markers. A major drawback of using a linear layer is that the number of parameters is quadratic with respect to the sequence length. There are exactly $n^2 d$ parameters. These methods work because they are combined with the mutual information feature selection. It would not be possible to use this method with the 36304 markers available, as 36304^2 is already equal to more than 1.3 billion, which is impossible to train with the available hardware. It also restricts the possible size of the embeddings, as increasing the embedding dimension of one will create more than 4 million extra parameters to train.

We will also consider two more embeddings. The third one is the one-hot encoding: the 0, 1, and 2 values will be encoded respectively as [1,0,0], [0,1,0], and [0,0,1]. The one-hot embedding does not use any parameters to produce the embedding, with the drawback that the embedding dimension is fixed to three. Unlike the previous embedding methods, the embedding dimension is fixed and cannot be modified. The implementation of the multi-head attention of Pytorch [6] requires modifying the number of heads, as it is required that the embedding dimension d is divisible by the number of heads. Therefore, it was decided to increase the number of heads to three when using the one-hot encoding.

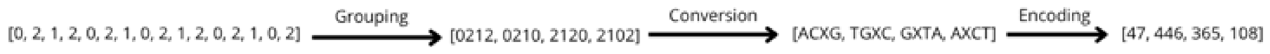


Figure 3.16: Illustration of the different steps of the tokenization

The fourth and last embedding method is the tokenization one proposed in [65]. Tokenization is a common technique used when dealing with texts as input. It consists of splitting the text into smaller parts, like letters or words, to ease the analysis. But tokenization can not be applied directly to our input data. Figure 3.16 illustrates the three steps we performed to obtain the input of the fourth method. The first step consists of grouping the markers by a defined size that we will call the token size ts. We will group markers in groups of eight, as the number of markers is divisible by eight. Note that in the example of Figure 3.16, the token size is set to four to have more tokens in the figure. The second step, the conversion, is an optional step that modifies the value of the markers. In the paper, they introduce the following conversion: if the marker is 0 or 2, we change the value by the upper letter of the nucleotide that appears in the genotype. This data is available in the .bim file provided alongside the main .bed file. If the marker is one, then we change it to X. In [65], only a small fraction of the SNPs of their dataset were heterozygous. They have decided to combine all possible combinations in a single letter to avoid creating tokens that are very rare in the dataset they have used. This hypothesis is not true for the dataset used in this thesis, as more or less 30% of the markers are heterozygous. When encountering those tokens, the prediction would be degraded as the associated embedding would be closer to the random initialization than to its optimal value, as there were few updates by the backpropagation due to the shallow presence of the token. We define the vocabulary size vs as the number of possible values for every marker after the conversion. In this case, we have A, C, T, G, and X, which makes a vocabulary size of five. The third and final step of the tokenization is the encoding. We define a value function v. This function assigns a different value to every marker, and the output of the function is between 0 and vs - 1. In the example of Figure 3.16, we have assigned v(A) = 0, v(C) = 1, v(G) = 2, v(T) = 3, and v(X) = 4. The formula of the encoding is:

$$\text{encoding}(\mathbf{t}) = \sum_{i=0}^{ts-1} v(\mathbf{t}_i) * vs^i \tag{3.5}$$

Where $\mathbf{t}_i$ is the value at position i on the group $\mathbf{t}$. Let us take as an example the first token of the Figure 3.16: ACXG. The application of Equation (3.5) would give:

$$0 * 5^3 + 1 * 5^2 + 4 * 5 + 2 = 47$$

This procedure ensures that every token has a different encoding. The number of tokens t is computed by the formula vs^{ts}. This tokenization of the sequences is precomputed to save time when training the models. The input embedding of the GPTransformer consists of a lookup in the matrix of Embedding $\in \mathbb{R}^{t \times d}$. Each row i contains the embedding for the token i. Note that the positional embedding is necessary in this case, as the embedding does not take into account the position of the token in the sequence. This method presents some advantages. The first is that we can scale the embedding dimension to much bigger values than when using a linear layer. Indeed, when we increase the embedding dimension of one, we only create $5^8 = 390625$ more parameters, which is ten times less than for the linear layer. The second advantage is that tokenization reduces by design the size of the sequences by a factor equal to the token size, as we group all these markers in a single value. There is thus no need to use the mutual information feature selection in this case.

The fifth and final type of embedding uses the embedding table but does not process the input in tokens. The embedding table will simply have three embeddings, one for each possible marker value. As the content of the embedding is updated thanks to backpropagation, this last embedding type is named the learned embedding method. It shares the same advantages as the tokenization method, but the sequence size is not reduced in this case, and the feature selection using the mutual information is used instead.

3.6.3 Experiments

For every embedding technique, we have tried several modifications. For the methods where it was possible, we tried to increase the size of the embedding to verify that the default values were not the bottleneck of the performance. A hidden layer of size 1024 was added at the end of the model before the output neuron to enable making more complex computations based on the output of the transformer block. Dropout was introduced to avoid the overfitting of the model. The value for the dropout was set to 0.25. The impact of the usage of AdamW and a weight decay of $5 * 10^{-4}$ was also tried in the same setup as the one developed for the ResNet model. We also have to train the models using the mean square error instead of the mean absolute error. Table B.10 summarizes the results obtained by the best GPTransformer models. Like in the previous section on the ResNet, the model's performance was not as good as the best neural network trained during this thesis, it was decided not to perform hyperparameter tuning on this architecture. The final configuration for a phenotype was chosen by taking the model that achieved the maximum correlation. Table A.11 shows the hyperparameters that vary across the phenotypes.

This table gives us information about the best embedding choice. The one-hot encoding and the learned embeddings are the only embedding types that are never used in the best configuration, and the tokenization is only used for the size phenotype. These three methods are the ones that use the classical embedding methods, while for the categorical and frequency embedding types, they used a linear layer, which is rarely used when using transformers. It makes sense that the methods developed for natural language processing do not work well with our dataset, as SNPs are not words, and a genotype is not a sentence. New embedding methods tailored for the input may increase the results to surpass the GBLUP model. This is very important as a bad embedding may have a non-negligible impact on the quality of the results. The performance of transformer-based architecture can only be correctly evaluated when the question of the embedding is properly tackled.

Table A.11 also analyzes the claim made in [11] that the frequency embedding yields better results in almost every phenotype compared to the categorical one. The frequency embedding is only used for the shoulder phenotype, while the categorical embedding is used for four phenotypes, which is not the behavior in the paper. The models can use the information from the heterozygous markers through the bias term of the expression. Figure 3.15 is misleading as it does not show the bias term, which can be defined in this case as the value of the neuron when all markers are heterozygous. The model will use the weights to learn how to correct the effect of having a homozygous marker for the locus, which can explain why the categorical embeddings still perform well even if the heterozygous markers are encoded as zeros. Note that this result is also coherent with the results obtained by using a convolution layer to encode the value in Section 3.5. It seems that there is no need to recode the markers and that the original encoding is used more effectively by the models.

3.6.4 Results and discussion

Table 3.9: GPTransformer models results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
GPTransformer	0.415	0.370	0.453	0.452	0.312	0.450

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
GPTransformer	1.262	2.589	1.088	1.060	5.894	1.187

Table 3.9 summarizes the results for the best-performing configuration of the GPTransformer architecture. For the correlation metrics, we can see that the performance of the GPTransformer is very poor in comparison to the baseline. For the mean absolute error metric, we can see that the performances are closer to the one from the baseline, except for the size phenotype, where the gap is bigger than for the other phenotypes. However, we can also see that the gap is bigger for the correlation metric. When we look at the gap between the baseline and the GPTransformer, it seems that it is proportional to the variance of the phenotype (found in Table 1.2). The phenotypes with the biggest variances are the Size, the Top, and the Shoulder phenotypes. These are also the phenotypes with the biggest gap for the correlation metric. Based on this observation, an experiment could be done to try the impact of the normalization of the phenotypes on the performance of the GPTransformer model.

3.7 GPTransformer2

The second architecture was initially proposed in [65]. It was decided to call it GPTransformer2 as it can be seen as an update of the previous GPTransformer architecture. Unlike the other sections, this one will focus on studying different variations of the tokenization. Doing so will help to better understand if applying this method to SNP markers is suitable. The next subsection will present the modification in the GPTransformer architecture, and then present the different modifications proposed.

3.7.1 From GPTransformer to GPTransformer2

The main modification introduced with the GPTransformer2 was the introduction of a linear layer between the last transformer block and the MLP that produces the prediction. The goal of this layer is to reduce the embedding dimension to reduce the number of inputs of the MLP. In the paper, we have taken the values from [65] for the embedding size: 32, and this linear layer has 2 output neurons. The number of inputs of the MLP is now independent of the embedding size and is sixteen times smaller when compared to the GPTransformer architecture. This MLP also has a different structure compared to the GPTransformer. It has two hidden layers with 512 and 128 neurons.

The second main modification was the introduction of random masking. It consists in replacing the encoded value from Figure 3.16 by a special token value with a given probability. The idea of the random masking is very similar to the dropout introduced earlier. However, the concept must be adapted to the input, as 0 represents a token. The token value is then set to vs^{ts}, which is the first value that is not used in the encoding.

The hyperparameters of the architecture are based on the values from [65]. The number of heads was increased to 4, and the number of transformer blocks was increased to 3. The number of neurons of the MLP in the transformer block was defined as four times the embedding size. As we did not change the embedding size, it was fixed to 128 neurons (as the embedding size was 32). No dropout was used: the dropout probability was set to 0. The AdamW optimizer was used with a weight decay set to 0.01. The learning rate was set to 10^{-4}.

3.7.2 Tokenization extensions

There are two possible ways to modify the tokenization. The first possible change we can make is the choice of the conversion. We will consider two more encoding possibilities: the first one is the simplest: instead of using letters based on the nucleotides at the marker locus, we keep the 0, 1, 2 values. The vocabulary size vs is thus of size three. The second is an extension of the conversion introduced in GPTransformer. This adaptation will separate the case where the marker value is 0 from the case where the marker is equal to 2. In the first case, the nucleotide letter will be in lowercase, while it will be in uppercase for the second case. The different possible values will be a, c, g, t, A, C, G, T, X, the vocabulary size will be of size 9.

The second parameter we can change is the token size ts. It controls the number of different possible tokens that will be in the dataset while also controlling the size of the dataset in tokens. The token size is thus a trade-off between the number of different tokens versus the number of tokens in the dataset. If the token size is big, there will be many possible tokens, and the size of the dataset will be smaller as more markers are grouped into a single token. If the token size becomes too big, then the number of tokens in the training dataset may not be sufficient to contain every possible token. As there will not be many samples of the vast majority of the tokens, their corresponding embedding will be very poor, which can impact the model predictions. On the contrary, when the token size is too small, then the sequences will be too big to be processed in an acceptable time frame. To give an idea, it takes forty minutes on the available hardware for one epoch of GPTransformer2 with a token size of four. If the token size is equal to one, it would take approximately one day to train during one epoch. Instead of taking less than a day to train a model, it will take nearly an entire month to train a single model. And there are six phenotypes in total, so six models to train. The value of the token size must then be chosen carefully to balance this tradeoff properly. We will test two values in this thesis: four and eight. These values were chosen as the number of markers is divisible by four and by eight. The value of sixteen was not considered, as even with the smallest vocabulary size of three, there would be more possible tokens ($3^{16} \approx 40000000$ tokens) than tokens in the training dataset ($13000 * \frac{36304}{16} \approx 29500000$ tokens) which was not a good idea.

Table 3.10: Tokenization names based on the token and vocabulary size

Token size / Vocabulary size	4	8
3	small	small-extended
5	medium	medium-extended
9	large	-

The combination of the two parameters gives us six possible tokenizations. In practice, we will not try the larger conversion with a vocabulary of 9 with a token size of 8 as it will create too many possible tokens ($9^8 \approx 43000000$ tokens) compared to the number of tokens in the training set ($13000 * \frac{36304}{8} \approx 59000000$ tokens). Table 3.10 introduces the naming convention for the different tokenizations that will be used in the rest of this work to avoid any confusion between the methods.

3.7.3 Experiments

The experiments based on this architecture were only done on the shoulder phenotype, as the training of all five tokenization models takes a lot of time. The same configurations will be used for every phenotype. Two experiments were done with this architecture. The first one consists of varying the batch size used during the training, and the second concerns the last difference between the GPTransformer and GPTransformer2. In the GPTransformer2, the activation function of the MLP inside the transformer block is not the ReLU used in GPTransformer but an alternative function, the GeLU.

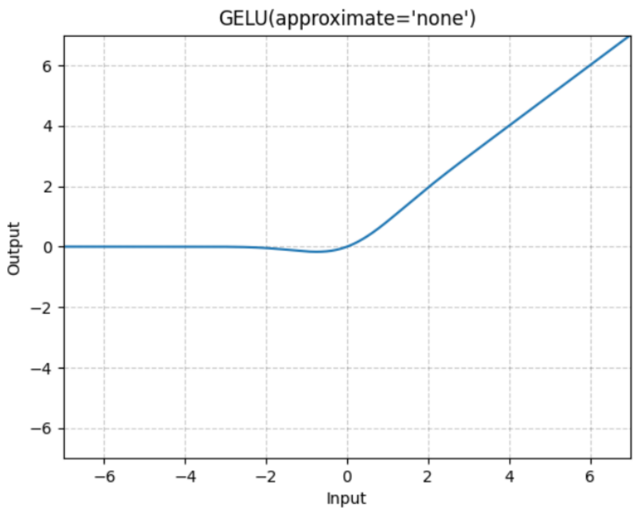


Figure 3.17: Graph of the GeLU function (taken from the documentation of [6])

This activation function is a smoother version of the ReLU that penalizes the small negative values as the output is not set to zero in this case. Note that this change only concerns the MLP inside the transformer block. For the final MLP that produces the prediction, the ReLU is still used as an activation function. Table B.11 summarizes the best results obtained for all tokenizations. There will not be any hyperparameter tuning for this architecture due to the same reasons as for the other models. Table A.12 summarizes the configuration for the five tokenization methods.

3.7.4 Results and discussion

Table 3.11: GPTransformer2 model results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.492	0.477	0.507	0.504	0.490	0.522
Small	0.395	**0.383**	0.390	0.426	**0.345**	0.436
Small-extended	0.357	0.341	0.351	0.370	0.233	0.406
Medium	**0.399**	0.359	0.374	0.423	**0.345**	0.453
Medium-extended	0.355	0.343	0.317	0.369	0.244	0.408
Large	0.382	0.344	**0.393**	**0.431**	0.315	**0.458**

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	5.347	1.100
Small	1.284	**2.649**	**1.071**	**1.076**	5.869	1.221
Small-extended	1.324	2.685	1.105	1.152	5.949	1.238
Medium	**1.279**	2.778	1.224	1.172	**5.724**	1.224
Medium-extended	1.314	2.691	1.094	1.121	6.705	**1.220**
Large	1.432	2.739	1.225	1.121	5.806	1.310

Table 3.11 summarizes the results of the best configuration on every phenotype for all tokenization methods. For more clarity, the best performance of the GPTransformer models for every phenotype is set in bold in the table. The first thing that can be seen is that, whatever the tokenization methods, the results are worse than those obtained with the GBLUP. It can also be seen that the performances of the models that use eight as token size, the "extended" methods, are lower than the ones that use a token size of four. This indicates that the size of the training dataset is really important. Using a token size of eight instead of four does reduce the number of tokens contained in the training size by a factor of two. The large difference in the results indicates that we are not close to the convergence and that the results may benefit from an increase in the training set size.

The second characteristic that we can evaluate is the impact of the conversion applied during the tokenization that defines the vocabulary size parameters. As the configuration was only tested on the shoulder phenotype, we will only consider this phenotype, as the others were not considered and are probably not the optimal results from all configurations tested during the experiments. It can be seen that the small and medium tokenization methods have very similar results, while the ones obtained with the large method are slightly worse. A possible explanation may be that using the large tokenization methods creates more possible tokens ($9^4 = 6561$), which is more than for the small and medium versions, which respectively create $3^4 = 81$ and $5^4 = 625$ tokens. The training set has thus fewer examples than for the other two methods, which would explain the lower results. Unfortunately, without any additional data, this is not possible to verify. To avoid any confusion, the tokenization method that will be used in Chapter 4 and Chapter 5, as the GPTransformer2 architecture will be the Medium one. The criterion used to make that choice is the fact that it achieved the best correlation on the single studied phenotype.

3.8 Multi-trait regression

This last section will evaluate the performance of the neural network architecture on the multi-trait regression task, like in the last section of the previous chapter. Every architecture but the GPTransformer and Deep-MLP ones were tested; the results of this experiment can be found in Table B.12. Note that the ResGS was not trained using the pipeline detailed in Section 3.4 and that the tokenization method used for the GPTransformer2 is the medium one. Based on these results, we can see that both LCLNN and the Shallow-MLP were the best models by a large margin. Unfortunately, as there was only time to tune the hyperparameter for a single architecture, a choice had to be made. The criterion was the summed correlation of the four phenotypes. Based on this criterion, the architecture that will be tuned is the Shallow-MLP. To avoid any confusion with the previous model, the name Multi-MLP will be used instead.

The hyperparameter range will be the same as the one from the first sweep for the MLP architecture described in Table 3.1a. As the architecture is the same, the most coherent choice is to take the same hyperparameters and the same set for each one of them. The best configuration will be chosen based on the model performing the best summed correlation for the four phenotypes. The summed correlation avoids making any biases toward any phenotypes. The best configuration can be found in Table A.13, and the performance of this model on the validation set can be found in Table B.13.

3.8.1 Results and discussion

Table 3.12: Multi-MLP model results

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)
GBLUP (VanRaden)	0.492	0.477	0.507	0.504
Multi-MLP	0.487	0.464	0.505	0.498

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)
GBLUP (VanRaden)	1.209	2.421	1.056	1.013
Multi-MLP	1.256	2.559	1.081	1.046

Table A.13 summarizes the results for the Multi-MLP results. It can be seen that the model performances are quite close to those of the baseline, but without surpassing it. It can also be seen that the performance for the Top phenotype is slightly worse than that of the other phenotypes. The same observations can be made on the mean absolute error metric.

Chapter 4

Global comparison of the models

The previous sections aimed to explain in detail all the architectures evaluated during this thesis. However, to avoid confusion, the different models were not compared with each other. This chapter aims to analyze the architectures more globally to better understand the interaction with machine learning methods and the genomic data. Every section will analyze the impact of a specific experiment using the results of the relevant information. The results of these evaluations will enable us to determine if the experiment was successful on the dataset or hurt the quality of the model's results. Note that we only display the result from the first run when several runs were performed for a given architecture, such that the results displayed are coherent for all models.

4.1 Evaluation of the multi-trait regression

Table 4.1: Maximum correlation of multi-trait models with their corresponding single-trait model

	Shoulder	Top	Buttock (side)	Buttock (rear)
Random forest	**0.380**	**0.358**	**0.394**	**0.375**
Multi-RF	0.379	0.352	0.365	0.374
XGBoost	**0.436**	0.413	0.454	**0.443**
Multi-XGB	0.433	**0.417**	**0.456**	**0.443**
Shallow-MLP	0.477	**0.464**	0.496	**0.498**
Multi-MLP	**0.487**	**0.464**	**0.505**	**0.498**

Table 4.1 shows the results on the correlation metric for the multi-trait regression and the single-trait ones. The best result for every family of the model is set in bold in the table. We can see that the results are almost the same between the single-trait model versus the multi-trait one. This conclusion is the same for all three models presented, despite the fact that they used very different algorithms. This could indicate that the conclusions made here could generalize to whatever architecture is used. However, we can not make this claim based on only three models. A separate experiment evaluating many more different models would be required before being able to claim that the usage of multi-trait regression does not impact the performance. Despite the lack of improvement in the results, the usage of the multi-trait regression has some advantages. For example, the training and the tuning of multi-trait models are reduced by a factor equal to the number of predicted phenotypes for the MLP, as only a single model has to be trained instead of one per phenotype. Future experiments using this dataset can use these results to train only multi-traits models to increase the efficiency of their work. However, using multi-trait regression may induce more processing of the dataset used for the training in order to avoid and manage the potential missing phenotypes.

4.2 Evaluation of the LCLNN model

Table 4.2: Maximum correlation obtained by the LCLNN compared to the Shallow-MLP

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Shallow-MLP	0.477	0.464	0.496	0.498	0.425	0.516
LCLNN	0.464	0.470	0.502	0.498	0.473	0.505

LCLNN is a new architecture that was not used before. It is thus important to analyze these results in order to evaluate this novel architecture. The architecture we will use for the comparison is the Shallow-MLP, as it is the closest architecture assessed in this thesis. Table 4.2 summarizes the results for both models. The results of both models are very similar (except for the size phenotype for the reasons explained in Section 3.2). When we look at the best configuration of the LCLNN, we can see that the locally connected layers are only a small part of the model compared to the linear model. Indeed, the number of parameters due to the LCL in the model is close to 300000 ($\approx 5 * 36304 + 3 * 36304$). When we compare to the configuration for the phenotype that used the smallest MLP in the LCLNN configuration, summarized in Table A.8, we can estimate that the number of parameters of the MLP is more than 2,3 million ($36304 * 64 = 2323456$) which is more or less eight times bigger than the contribution of the LCL. This contribution is even less for the other phenotypes that use more hidden neurons. The impact provided by the LCL layers is thus very limited and looks like an encoding layer for a classical MLP. Experiments done with the encoding done in Section 3.6 and Section 3.5 show that it does not significantly impact the model, and this comparison shows the same behavior. Therefore, the LCLNN architecture does not look like a very promising architecture for genomic selection. It can be seen as a modification of an MLP that does not improve the results compared to the version without the modifications. There is thus no reason to use the modified version instead of the original one.

4.3 Evaluation of the residual training pipeline

Table 4.3: Maximum correlation of the ResGS compared to the Ridge regression model

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Ridge	0.490	0.477	0.508	0.504	0.491	0.523
ResGS	0.482	0.477	0.505	0.497	0.492	0.525

In Section 3.4, a new training pipeline was derived based on the one proposed by [10]. In the version used, the first phase used the optimal ridge model introduced in Section 2.3. When training a model to predict residuals, like for ResGS, it is important to evaluate if the prediction of these residuals improves the results. Otherwise, the prediction of these residuals is useless, and only the base model should be used. Table 4.3 summarizes the relevant results. It can be seen that the performance of the Ridge regression and the ResGS architecture is very close. The addition of the ResGS does not seem to add any useful value to the model, despite being the biggest model evaluated during this thesis. Based on these results, we can conclude that there are no benefits in adding a ResGS model in order to predict the phenotype. However, these results do not mean that the method is not suited to the problem of genomic selection. Lots of variations are possible and may improve the results. For example, instead of using a tuned Ridge regression model, another linear model could have been used, or even a nonlinear model. This experiment concludes that the usage of a ResGS model to predict the residuals is not a good idea as there are very few differences with the ridge model used alone.

4.4 Evaluation of the transformer architectures

Table 4.4: Maximum correlation of the transformer architectures

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GPTransformer	0.415	0.370	0.453	0.452	0.312	0.450
GPTransformer2	0.399	0.359	0.374	0.423	0.345	0.453

During this thesis, we have evaluated two models based on the transformer architecture. GPTransformer evaluates different types of embeddings, while GPTransformer2 explores different variations for the tokenization. In this section, we will compare the results of both models to understand which method should be preferred to reduce the size of the sequence to compensate for the weakness of the attention mechanism used in transformer architectures. Table 4.4 shows that GPTransformer performs better in all phenotypes but the musculature and the size phenotype. However, as the gap is relatively small and the threshold on the feature selection was not tuned during the experiments done with the GPTransformer models, we cannot conclude that using mutual information feature selection yields better results than the usage of tokenization. Supplementary experiments are required with different thresholds for the mutual information to be able to conclude. Furthermore, there is no indication that the mutual information is the best feature selection method. To be as complete as possible, other feature selection methods must be tested to find the one that best suits the problem.

4.5 Impact of the size of the dataset

One of the main specificities of this thesis is the large number of individuals present in the dataset. However, the fact that the dataset is bigger than the ones used in the literature about genomic selection does not mean that the training set is sufficiently big for the tested architectures. As machine learning algorithms heavily depend on the size of the dataset, it is important to consider it in order to have a complete analysis. To do so, we will train the Shallow-MLP, GBLUP, and ResNet using a fraction of the training dataset to evaluate if the performance evolves according to the size of the dataset. To avoid confusion, we will only consider the Shoulder phenotype in this section. We will train the models using the first 500, 1000, 2000, 5000, and 10000 samples of the original training set using the best configuration for the shoulder phenotype. Note that this analysis was not performed on the transformer-based architecture, as the different tokenization methods already enable us to discuss the impact of the size of the dataset.
Figure 4.1 shows the results of these experiments. The abscissa axis uses a logarithmic scale to better illustrate the results. We can see that for the ResNet model, the correlation increases linearly with the logarithm of the training set size. For the Shallow-MLP, we can see the same linear behavior, but it is less clear for bigger training set values, where the increase looks to slow down a bit. The last point, which uses the whole training set, is less aligned. It can be explained by the fact that the configurations used in these experiments were chosen to optimize this value. The increase is bigger for the Shallow-MLP model as hyperparameter tuning was performed, which further optimizes the configuration compared to the ResNet, where only the configuration was chosen based on the experiments. The last model, GBLUP, has very similar results compared to those of the Shallow-MLP. Figure 4.1 shows that even if the size of the dataset is very big compared to the ones found in the literature, the correlation is still increasing as we increase the size of the dataset with either of the three models. It is thus not possible to draw general conclusions on the architectures, as this experiment shows that if we had more data available, the performance could be improved. With this given dataset, the ResNet architecture seems to underperform compared to the MLP, but this may be explained by the fact that the ResNet architecture requires more examples than the MLP to achieve its best performance. A natural follow-up to this thesis would consist of extending the

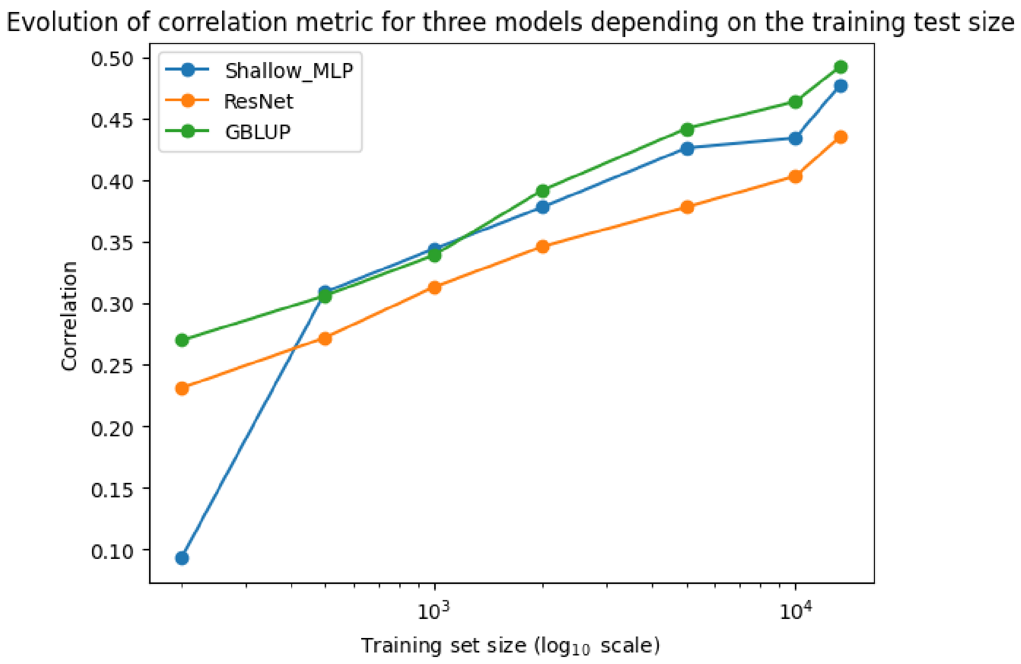


Figure 4.1: Impact of the training set size on the performance of GBLUP, Shallow-MLP, and ResNet models for the shoulder phenotype.

dataset to reach a point where increasing the training set would not increase the performance. Only then could conclusions be made on the real performances of the different architectures. Note that the increase of the dataset set must be big enough to matter: the linear scaling is on the logarithm of the training size. To have a significant upgrade of the dataset would require multiplying its size several times, which could be impossible in a reasonable amount of time. This is a major hurdle to be surpassed that may slow down the research in the genomic selection field.

4.6 Evaluation of all tested methods against the GBLUP baseline

Table 4.5 summarizes the results of all models tested during this thesis. For every table and every phenotype, the best results are set in bold. For the models where several runs were performed, only the performance of the first run is displayed in the table, such that the table contains only the performance of a single model. Compared to the GBLUP model, better results in the correlation metric were found in three out of the six phenotypes, and equal values were found for two other phenotypes. However, the improvement is nearly negligible as the correlation was only improved by at most five thousandths. For the mean absolute error, the GBLUP is only the best model for a single phenotype, but like for the correlation, the improvement is very small: the gap between the baseline and the best model is at most twelve thousandths. For the models defined in Chapter 2, the best models were the Ridge and the SVM, followed by the XGBoost, and finally the random forest. For the model defined in Chapter 3, the best models used the linear architecture, followed by the CNN, and finally the transformer-based models. To conclude, the results summarized from Table 4.5 show that we have not found a better model than the GBLUP; we have only trained models that give similar results at the best.

Table 4.5: Results of all models evaluated during this thesis

(a) Correlation

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	**0.492**	**0.477**	0.507	**0.504**	0.490	0.522
Ridge	0.490	**0.477**	0.508	**0.504**	0.491	0.523
Random forest	0.380	0.358	0.392	0.375	0.329	0.395
Multi-RF	0.379	0.352	0.365	0.374	/	/
Multi-XGB	0.433	0.417	0.456	0.443	/	/
XGBoost	0.436	0.413	0.454	0.443	0.377	0.460
SVM	0.486	0.475	**0.512**	0.500	0.490	0.518
Shallow-MLP	0.477	0.464	0.496	0.498	0.425	0.516
Deep-MLP	0.464	0.460	0.500	0.493	0.467	0.507
Multi-MLP	0.487	0.464	0.505	0.498	/	/
LCLNN	0.464	0.470	0.502	0.498	0.473	0.505
GPTransformer	0.415	0.370	0.453	0.452	0.312	0.450
GPTransformer2	0.399	0.359	0.374	0.423	0.345	0.453
ResNet	0.435	0.427	0.475	0.450	0.437	0.471
ResGS	0.482	**0.477**	0.505	0.497	**0.492**	**0.525**

(b) Mean absolute error

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	1.209	2.421	1.056	1.013	**5.347**	1.100
Ridge	**1.208**	2.421	1.053	**1.012**	5.352	**1.098**
Random forest	1.325	2.636	1.153	1.112	5.970	1.188
Multi-RF	1.331	2.636	1.165	1.124	/	/
Multi-XGB	1.383	2.612	1.103	1.088	/	/
XGBoost	1.258	2.534	1.089	1.057	5.757	1.138
SVM	1.211	2.430	**1.045**	1.013	5.391	1.105
Shallow-MLP	1.265	2.445	1.060	1.105	5.968	1.114
Deep-MLP	1.236	2.801	1.088	1.061	5.524	1.125
Multi-MLP	1.256	2.559	1.081	1.046	/	/
LCLNN	1.704	3.085	1.368	1.085	5.394	1.404
GPTransformer	1.262	2.589	1.088	1.060	5.894	1.187
GPTransformer2	1.279	2.778	1.224	1.172	5.724	1.224
ResNet	1.289	2.483	1.077	1.061	6.055	1.155
ResGS	1.209	**2.409**	1.057	1.015	5.349	1.105

Chapter 5

Conclusion

The goal of this thesis was to try different machine learning algorithms to find a better model than the GBLUP, one of the state-of-the-art models used in the genomic selection field. To do so, we have used a cattle dataset of a size much larger than the ones used in the literature. It enables testing methods that usually require much more data, like the transformer models. In Chapter 1, we started by introducing the main concepts, the dataset, and developing the GBLUP method. This chapter introduces the necessary concept and explains the context of the thesis by defining what genomic selection is and explaining the model that we will use as the baseline for all models tested.

In Chapter 2, we explained the classical machine learning models. We have found that the Ridge and the SVM were both models that achieved results that are very close to the GBLUP baseline. However, the SVM was not very practical to use as the training time for this model was much higher than the other ones due to the size of our dataset. XGBoost and random forest were also considered, but they do not reach the level of the baseline. We have also experimented the impact of multi-trait regression for the XGBoost and Random Forest and have found that it does not decrease the results, but hastens the development, as only one model for all phenotypes has to be tuned instead of training as many models as we have phenotypes.

In Chapter 3, we explained models based on three big families of neural networks. The first model examined was the MLP, where two architectures were considered. Shallow-MLP is a one-hidden-layer MLP, whereas Deep-MLP is a nine-hidden-layer MLP. We showed that both architectures achieved results close to the baseline, with Shallow-MLP being a little better. LCLNN was a new architecture based on the idea of [7] that uses a local connected layer: a convolutional layer that does not share its weights. This model performed similarly to the MLP, as the design of the architecture was very close to an MLP, and the addition of the LCL did not impact the model that much. For the convolutional neural network, we have used two different architectures, the ResGS and a custom ResNet. We have introduced a new training pipeline with ResGS that uses the Ridge model to provide a first estimation and then uses ResGS to predict the residual of the Ridge prediction. The results showed that this training method was not very effective, as their results were very close to those obtained with the ridge model only, which means the ResGS did not really influence the results. The impact of the pooling layer was studied with both architectures, and we showed that using maximum or average pooling, like in the classical CNN does decrease the performance of the models compared to the one using a strided convolution. The impact of a different encoding of the data was also tested with the ResNet, but it does not influence the results. The last two models are based on the Transformer architecture. The first model, GPTransformer, studied five different types of embedding for the model. The best one was the one defined in [11] that uses a linear layer as embedding. GPTransformer2, the second transformer-based architecture, introduced a new version of the tokenization procedure to evaluate a way to avoid the usage of feature selection to reduce the length of the input sequence that slows the transformer. The results obtained indicate that the encoding does not seem to influence the model, and that the number of tokens in the training size has a major influence on the performance. The big difference in results according to the number of tokens in the training set indicates that the models may benefit from more data.

The last chapter, Chapter 4, compared the different models with each other; the main experiment was the evaluation of the impact of the size of the training set on the performance of the Shallow-MLP, GBLUP, and ResNet models. It shows that we have not yet converged to the best performance despite the size of the dataset. This indicates that to be able to truly evaluate the potential of neural networks, more data is required before being able to conclude that one architecture is better than the others.

As this thesis's goal was to explore the possibilities, there are lots of possible follow-up works to be done. There are still many parameters still have to be tested for their impact, and different architectures have to be evaluated. The results observed in the thesis show that the classical methods used in other fields can not be applied blindly to predict phenotypes based on the genotype. Some examples from this work are the one-hot encoding and the max pooling, which are far from reaching the best performances. Defining an architecture that introduces inductive biases tailored to genomic selection may be a solution to increase the performance of neural networks while reducing the number of samples required to reach the best performance. An example would be the definition of an embedding method that can encode the sequence of SNP markers in a way that makes sense biologically. Another possible follow-up would be to use larger SNP marker sequences. However, this would induce new problems to be solved due to the increase in size. Models like the transformer would become very difficult to use as it is due to the increased sequence lengths. Another possible follow-up would be to study the impact of the choice of the loss function. This thesis only used the mean absolute error as loss, but tried only once to use the mean squared error for the GPTransformer. Table A.11 shows that some of the best GPTransformer models were trained using this metric. As the main metric, the Pearson correlation coefficient can not be used as a loss function. The question of the best loss function to use to train the model remains open. The mean absolute error is not the perfect choice, as we can see that for some results, the best correlation was obtained with a high mean absolute error. Finding a loss function that is more similar to the correlation metric may increase the results and ease the training of the models.

To conclude, the usage of machine learning models for genomic prediction is still at an early stage. In this thesis, a wide range of algorithms was evaluated, but without surpassing the linear baseline. However, the performed experiments enabled us to find some interesting information that could be used in further research in the field. This work can then be used to make further research on the usage of machine learning algorithms for genomic selection.

Bibliography

[1] Neogen Australasia : Demystifying DNA technology, a livestock breeders guide to genomic selection. https://www.genetics.mla.com.au/globalassets/genetics/documents/breeders-guide-to-genomics_a4_web.pdf. [Accessed 17–03-2025].

[2] Michael W. Davidson et Florida State University : Molecular expressions cell biology: Chromatin and chromosomes. https://micro.magnet.fsu.edu/cells/nucleus/chromatin.html, 2015. [Accessed 19-05-2025].

[3] Association Wallonne des Eleveurs : Genomique blanc bleu belge. https://www.awenet.be/awe/UserFiles/file/Newsletters/WE-GÃĨnomique_WEB.pdf, 2020. [Accessed 13-03-2025].

[4] Louis Wehenkel et Pierre Geurts : Slides of introduction to machine learning, 2024.

[5] Aston Zhang, Zachary C. Lipton, Mu Li et Alexander J. Smola : *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai et Soumith Chintala : Pytorch: An imperative style, high-performance deep learning library, 2019.

[7] Hyo-Jun Lee, Jun Heon Lee, Cedric Gondro, Yeong Jun Koh et Seung Hwan Lee : deepgblup: joint deep learning networks and gblup framework for accurate genomic prediction of complex traits in korean native cattle. *Genetics Selection Evolution*, 55(1):56, 2023.

[8] Sana Mujeeb, Turki Ali Alghamdi, Sameeh Ullah, Aisha Fatima, Nadeem Javaid et Tanzila Saba : Exploiting deep learning for wind power forecasting based on big data analytics. *Applied Sciences*, 9(20):4417, 2019.

[9] Gilles Louppe : Deep learning lecture slides. https://github.com/glouppe/info8010-deep-learning, 2024.

[10] Zhengchao Xie, Xiaogang Xu, Ling Li, Cuiling Wu, Yinxing Ma, Jingjing He, Sidi Wei, Jun Wang et Xianzhong Feng : Residual networks without pooling layers improve the accuracy of genomic predictions. *Theoretical and Applied Genetics*, 137(6):138, 2024.

[11] Sheikh Jubair, James R Tucker, Nathan Henderson, Colin W Hiebert, Ana Badea, Michael Domaratzki et WG Dilantha Fernando : Gptransformer: a transformer-based deep learning method for predicting fusarium related traits in barley. *Frontiers in plant science*, 12:761402, 2021.

[12] Contributors to Wikimedia projects : Mutation. https://en.wikipedia.org/wiki/Mutation, mai 2025. [Accessed 18-03-2025].

[13] Contributors to Wikimedia projects : Bovine genome. https://en.wikipedia.org/wiki/Bovine_genome, janvier 2025. [Accessed 19-03-2025].

[14] Contributors to Wikimedia projects : Jonagold. https://en.wikipedia.org/wiki/Jonagold, sep 30 2024. [Accessed 01-04-2025].

[15] Contributors to Wikimedia projects : Polyploidy. https://en.wikipedia.org/wiki/Polyploidy, nov 22 2024. [Accessed 01-04-2025].

[16] Wenqing Fu, Timothy D O'Connor et Joshua M Akey : Genetic architecture of quantitative traits and complex diseases. *Current opinion in genetics & development*, 23(6):678–683, 2013.

[17] Contributors to Wikimedia projects : Heritability. https://en.wikipedia.org/wiki/Heritability, février 2025. [Accessed 19-05-2025].

[18] Naomi Wray et Peter Visscher : Estimating Trait Heritability. https://www.nature.com/scitable/topicpage/estimating-trait-heritability-46889/, 2008. [Accessed 30-04-2025].

[19] Gang Wang, Ence Yang, Candice L Brinkmeyer-Langford et James J Cai : Additive, epistatic, and environmental effects through the lens of expression variability qtl in a twin cohort. *Genetics*, 196(2):413–425, 2014.

[20] Theo HE Meuwissen, Ben J Hayes et ME1461589 Goddard : Prediction of total genetic value using genome-wide dense marker maps. *genetics*, 157(4):1819–1829, 2001.

[21] Francois F Guillaume, Didier Boichard, Vincent Ducrocq et Sébastien S Fritz : Utilisation de la sélection génomique chez les bovins laitiers. *INRA Productions animales*, 24(4):363–368, 2011.

[22] Adriana García-Ruiz, John B Cole, Paul M VanRaden, George R Wiggans, Felipe J Ruiz-López et Curtis P Van Tassell : Changes in genetic selection differentials and generation intervals in us holstein dairy cattle as a result of genomic selection. *Proceedings of the National Academy of Sciences*, 113(28):E3995–E4004, 2016.

[23] George R Wiggans, John B Cole, Suzanne M Hubbard et Tad S Sonstegard : Genomic selection in dairy cattle: the usda experience. *Annual review of animal biosciences*, 5(1):309–327, 2017.

[24] PM VanRaden : Symposium review: How to implement genomic selection. *Journal of Dairy Science*, 103(6):5291–5301, 2020.

[25] Association wallonne des éleveurs : Rapport d'activités 2024. https://www.awenet.be/awe/UserFiles/file/RapportActivites/RA2024-web.pdf, 2024. [Accessed 18-05-2025].

[26] Herd-book belgian blue beef. https://www.hbbbb.be/en/pages/origin-evolution. [Accessed 18-05-2025].

[27] Can Yuan, Alain Gillon, José Luis Gualdrón Duarte, Haruko Takeda, Wouter Coppieters, Michel Georges et Tom Druet : Evaluation of genomic selection models using whole genome sequence data and functional annotation in belgian blue cattle. *Genetics Selection Evolution*, 57(1):10, 2025.

[28] Christopher Chang Shaun Purcell : Plink software. www.cog-genomics.org/plink/1.9/.

[29] Association wallonne des eleveurs. https://www.awenet.be/awe/commun/asbl/viande/index_taureaux_explication.php. [Accessed 18-05-2025].

[30] José Luis Gualdrón Duarte, Can Yuan, Ann-Stephan Gori, Gabriel CM Moreira, Haruko Takeda, Wouter Coppieters, Carole Charlier, Michel Georges et Tom Druet : Sequenced-based gwas for linear classification traits in belgian blue beef cattle reveals new coding variants in genes regulating body size in mammals. *Genetics Selection Evolution*, 55(1):83, 2023.

[31] Stuart Macgregor, Belinda K Cornes, Nicholas G Martin et Peter M Visscher : Bias, precision and heritability of self-reported and clinically measured height in australian twins. *Human genetics*, 120:571–580, 2006.

[32] George K Robinson : That blup is a good thing: the estimation of random effects. *Statistical science*, pages 15–32, 1991.

[33] Contributors to Wikimedia projects : Mixed model. `https://en.wikipedia.org/wiki/Mixed_model`, avril 2025. [Accessed 16-05-2025].

[34] Paul M VanRaden : Efficient methods to compute genomic predictions. *Journal of dairy science*, 91(11):4414–4423, 2008.

[35] Jian Zeng : Best linear unbiased prediction (blup). PowerPoint slides: `https://cnsgenomics.com/data/teaching/GNGWS24/module5/Lecture3_BLUP.pdf`, 2024. Slides credit: Ben Hayes. [Accessed 16-05-2025].

[36] CNS Genomics Group : Practical 3: Best linear unbiased prediction (blup). Practical session notes (PDF): `https://cnsgenomics.com/data/teaching/GNGWS22/module4/Practical3.pdf`, juin 2022. [Accessed 16-05-2025].

[37] Anıl Kasakolu et Seyrani Koncagül : Effects of different methods and genomic relationship matrices on reliabilities of genomic selection in dairy cattle. *Livestock Studies*, 62(2):58–64, 2022.

[38] Theo Meuwissen, Ben Hayes et Mike Goddard : Accelerating improvement of livestock with genomic selection. *Annu. Rev. Anim. Biosci.*, 1(1):221–237, 2013.

[39] Qianqian Zhang, Florian Privé, Bjarni Vilhjálmsson et Doug Speed : Improved genetic prediction of complex traits from individual-level data or summary statistics. *Nature communications*, 12(1):4192, 2021.

[40] Contributors to Wikimedia projects : Restricted maximum likelihood. `https://en.wikipedia.org/wiki/Restricted_maximum_likelihood`, nov 2024. [Accessed 18-05-2025].

[41] David Habier, Rohan L Fernando, Kadir Kizilkaya et Dorian J Garrick : Extension of the bayesian alphabet for genomic selection. *BMC bioinformatics*, 12:1–12, 2011.

[42] Fatima Shokor, Pascal Croiseau, Hugo Gangloff, Romain Saintilan, Thierry Tribout, Tristan Mary-Huard et Beatriz C.D. Cuyabano : Predicting nonlinear genetic relationships between traits in multi-trait evaluations by using a gblup-assisted deep learning model. *bioRxiv*, 2024.

[43] Xue Wang, Shaolei Shi, Md Yousuf Ali Khan, Zhe Zhang et Yi Zhang : Improving the accuracy of genomic prediction in dairy cattle using the biologically annotated neural networks framework. *Journal of Animal Science and Biotechnology*, 15(1):87, 2024.

[44] Pinar Demetci, Wei Cheng, Gregory Darnell, Xiang Zhou, Sohini Ramachandran et Lorin Crawford : Multi-scale inference of genetic trait architecture using biologically annotated neural networks. *PLoS genetics*, 17(8):e1009754, 2021.

[45] Osval Antonio Montesinos-López, Abelardo Montesinos-López, Paulino Pérez-Rodríguez, José Alberto Barrón-López, Johannes WR Martini, Silvia Berenice Fajardo-Flores, Laura S Gaytan-Lugo, Pedro C Santana-Mancilla et José Crossa : A review of deep learning applications for genomic selection. *BMC genomics*, 22:1–23, 2021.

[46] Kelin Wang, Muhammad Ali Abid, Awais Rasheed, Jose Crossa, Sarah Hearne et Huihui Li : Dnngp, a deep neural network-based method for genomic prediction using multi-omics data in plants. *Molecular Plant*, 16(1):279–293, 2023.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser et Illia Polosukhin : Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[48] Cuiling Wu, Yiyi Zhang, Zhiwen Ying, Ling Li, Jun Wang, Hui Yu, Mengchen Zhang, Xianzhong Feng, Xinghua Wei et Xiaogang Xu : A transformer-based genomic prediction method fused with knowledge-guided module. *Briefings in Bioinformatics*, 25(1):bbad438, 2024.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot et E. Duchesnay : Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] John C Whittaker, Robin Thompson et Mike C Denham : Marker-assisted selection using ridge regression. *Genetics Research*, 75(2):249–252, 2000.

[51] Vladimir N. Vapnik : *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer New York, New York, NY, 2 édition, 2000.

[52] The MathWorks, Inc. : Understanding Support Vector Machine Regression. `https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html#buytaw5`, mai 2025. [Accessed 22-05-2025].

[53] Contributors to Wikimedia projects : Radial basis function kernel. `https://en.wikipedia.org/wiki/Radial_basis_function_kernel`, avril 2025. [Accessed 22-05-2025].

[54] scikit-learn developers : Rbf svm parameters. `https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html`, 2025. [Accessed 22-05-2025].

[55] Ivor Tsang, James Kwok et Pak-Ming Cheung : Very large svm training using core vector machines. *In International Workshop on Artificial Intelligence and Statistics*, pages 349–356. PMLR, 2005.

[56] Jerome H Friedman : Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[57] Tianqi Chen et Carlos Guestrin : Xgboost: A scalable tree boosting system. *In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

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

[59] Lukas Biewald : Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[60] Omry Yadan : Hydra - a framework for elegantly configuring complex applications. Github, 2019.

[61] Kurt Hornik, Maxwell Stinchcombe et Halbert White : Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[62] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever et Ruslan R Salakhutdinov : Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition, 2015.

[64] Ilya Loshchilov et Frank Hutter : Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[65] Renqi Chen, Wenwei Han, Haohao Zhang, Haoyang Su, Zhefan Wang, Xiaolei Liu, Hao Jiang, Wanli Ouyang et Nanqing Dong : An embarrassingly simple approach to enhance transformer performance in genomic selection for crop breeding. *arXiv preprint arXiv:2405.09585*, 2024.

Appendix A

Model configurations

This section will detail all the parameters used for the models evaluated on the test set. Note that only the parameters required to construct the Python class of the models and important training hyperparameters will be displayed here. For the models created using scikit-learn models, the name of the parameters refers to the argument of the class.

Table A.1: Configuration of best Random forest models

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Max features	0.1	$\frac{1}{3}$	0.01	0.25	0.1	0.25
Max depth	35	33	17	23	25	15

Table A.2: Configuration of best Ridge regression models

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Alpha	55600	44500	26250	34000	20900	23950

Table A.3: Configuration of best SVM models

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
γ	$1.225*10^{-4}$	$1.225*10^{-4}$	$7.75*10^{-5}$	10^{-4}	10^{-4}	10^{-5}
C	3.25	7.25	3.25	5.5	21.25	3.25

Table A.4: Configuration of best XGBoost models

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Max depth	7	7	4	3	9	3
Learning rate	0.01	0.01	0.1	0.1	0.01	0.1
Subsampling	0.75	0.75	0.75	0.75	0.25	0.75

Table A.5: Configuration for multi-trait models

(a) Configuration for Multi-RF

	Max depth	Max features	Normalization
Multi-RF	21	$\frac{1}{3}$	False

(b) Configuration for Multi-XGB

	Max depth	Learning rate	Subsampling	Normalization
Multi-XGB	7	0.01	0.75	True

Table A.6: Configuration of best Deep-MLP architecture

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Hidden size	512	1024	64	512	1024	1024
Batch size	32	64	128	64	128	8
Learning rate	0.000556	0.000889	0.000445	0.000112	0.000223	0.000112
Dropout	0.25	0.4	0.4	0.45	0.5	0.25

Table A.7: Configuration of best Shallow-MLP architecture

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Hidden size	8192	32	512	4096	10240	256
Batch size	256	1024	32	256	1024	32
Learning rate	0.000112	0.000556	0.0005	0.005	0.0005	0.001
Dropout	0.5	0.45	0.15	0.8	0.5	0.5

Table A.8: Configuration of best LCNN architecture

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Learning rate	0.0001	0.0001	0.00001	0.00001	0.0001	0.0001
Dropout	0.4	0.5	0.4	0.3	0.4	0.3
Hidden size 0	128	64	1024	1024	2048	128
hidden size 1	32	8	4	64	64	4

Table A.9: Configuration of best ResGS architecture

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Learning rate	0.0001	0.01	0.00001	0.00001	0.0001	0.001
Dropout	0.2	0.15	0.1	0.2	0.1	0.3
Hidden size	1024	256	512	32	256	512
Target size	2400	8400	10400	12400	4400	10400

Table A.10: Configuration of best ResNet architecture

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Hidden size	512	128	128	512	512	512
Learning rate scheduler	LinearLR	None	None	LinearLR_40	None	LinearLR

Table A.11: Configuration of best GPTransformer architecture

Parameters	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Embedding	frequency	categorical	categorical	categorical	tokenization	categorical
Learning rate	$5 * 10^{-4}$	$5 * 10^{-4}$	$5 * 10^{-4}$	$5 * 10^{-4}$	10^{-5}	$5 * 10^{-4}$
Weight decay	0	0	0	0	0.005	0
Loss	MSE	MSE	MAE	MAE	MAE	MSE

Table A.12: Configuration of best GPTransformer2 architecture in function of the tokenization method

Parameters	Small	Small-extended	Medium	Medium-extended	Large
Batch size	170	256	170	170	180
Activation	ReLU	ReLU	GeLU	ReLU	GeLU

Table A.13: Configuration of best Multi-MLP architecture

Parameters	Multi-MLP
Hidden size	8192
Batch size	512
Learning rate	0.000334
Dropout	0.25

Appendix B

Validation set results

This section will display some results of the architectures on the validation set. These results are placed in this appendix to avoid any confusion with the results obtained on the test set presented in the main text of the thesis. Note that only the results mentioned in the thesis are present in this section to avoid overloading the document.

Table B.1: Maximal correlation obtained from the models in Chapter 2 compared to the GBLUP

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.471	0.464	0.498	0.526	0.447	0.550
Random forest	0.415	0.409	0.381	0.438	0.360	0.434
Ridge	0.475	0.468	0.499	0.526	0.448	0.551
SVM	0.479	0.462	0.502	0.535	0.446	0.555
XGBoost	0.445	0.450	0.460	0.478	0.386	0.507

Table B.2: Summed correlation of the best multi-trait models with and without normalization.

	With normalization	Without normalization
Multi-RF	1.597	1.611
Multi-XGB	1.816	1.795

Table B.3: Results of the normalization experiment on the MLP architectures

(a) Maximum correlation obtained without normalization

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Shallow-MLP	0.471	0.456	0.485	0.521	0.410	0.540
Deep-MLP	0.470	0.456	0.483	0.518	0.419	0.546

(b) Maximum correlation obtained with normalization

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Shallow-MLP	0.469	0.456	0.487	0.517	0.416	0.543
Deep-MLP	0.467	0.452	0.486	0.524	0.417	0.543

Table B.4: Maximal correlation obtained in the different sweeps compared to the GBLUP

	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.471	0.464	0.498	0.526	0.447	0.550
Shallow-MLP (first sweep)	0.477	0.466	0.498	0.524	0.428	0.556
Shallow-MLP (second sweep)	0.476	0.464	0.495	0.525	0.433	0.552
Deep-MLP	0.476	0.463	0.498	0.523	0.433	0.552

Table B.5: Maximal correlation obtained depending on the reduction method used in ResGS

Reduction method	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Max pooling	0.277	0.252	0.277	0.309	0.244	0.304
Strided convolution	0.347	0.329	0.365	0.366	0.293	0.378
Strided LCL	0.277	/	/	/	/	/

Table B.6: Maximal correlation obtained depending on the training pipeline with ResGS

Training pipeline	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Whole phenotype	0.483	0.474	0.506	0.531	0.458	0.553
With residuals	0.347	0.329	0.365	0.366	0.293	0.378

Table B.7: Maximal correlation obtained depending on the reduction method used in ResNet

(a) Result obtained using the fourth architecture

Reduction method	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Pooling	0.389	0.352	0.377	0.409	0.289	0.444
Strided	0.428	0.383	0.448	0.487	0.372	0.508
Strided + LCL	0.408	0.366	0.368	0.449	0.292	0.461

(b) Results obtained using the fifth architecture

Reduction method	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
Pooling	0.329	0.344	0.360	0.411	0.273	0.399
Strided	0.420	0.373	0.441	0.474	0.339	0.472

Table B.8: Maximal correlation obtained depending on the encoding used in ResNet

Encoding method	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
None	0.428	0.383	0.448	0.487	0.372	0.508
Convolutional	0.347	0.392	0.442	0.445	0.354	0.465
LCL	0.410	0.365	0.420	0.451	0.379	0.472

Table B.9: Maximal correlation obtained with all ResNet experiments

Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
0.458	0.432	0.475	0.503	0.411	0.533

Table B.10: Maximal correlation obtained with the GPTransformer models compared to the GBLUP

Training pipeline	Shoulder	Top	Buttock (side)	Buttock (rear)	Size	Musculature
GBLUP (VanRaden)	0.471	0.464	0.498	0.526	0.447	0.550
GPTransformer	0.453	0.399	0.451	0.478	0.312	0.493

Table B.11: Maximal correlation obtained with the GPTransformer2 according to the tokenization method

Tokenization method	Shoulder
Small	0.403
Small-extended	0.381
Medium	0.421
Medium-extended	0.381
Large	0.422

Table B.12: Maximal correlation obtained from the models in Chapter 3 using the multi-trait regression

	Shoulder	Top	Buttock (side)	Buttock (rear)
Shallow-MLP	0.481	0.445	0.476	0.512
LCLNN	0.483	0.446	0.467	0.512
ResGS	0.400	0.367	0.385	0.434
ResNet	0.397	0.370	0.395	0.456
GPTransformer2	0.412	0.380	0.424	0.453

Table B.13: Correlation obtained by the best Multi-MLP model during the hyperparmeter tuning

Shoulder	Top	Buttock (side)	Buttock (rear)
0.488	0.461	0.492	0.515