**File S1**

**Genomic best linear unbiased prediction (GBLUP) model**

We used GBLUP model to estimate the response variable and its prediction accuracy as a comparison to MLPs and CNNs:

$y =μ+ g +e$,

where $y$ is the response variable with phenotypes ($y=y\_{adj}$ in the real dataset), $μ$ is the population mean, $g \~ N(0, Gσ\_{g}^{2})$ is a vector of random genomic effects, $σ\_{g}^{2}$ is the genomic variance, and $e \~ N(0,Iσ\_{e}^{2})$ is a vector of residuals where $I$ is an identity matrix with 1s in the diagonal. The matrix $G=ZZ'$ is the genomic relationship matrix and $Z$ is a matrix with standardized allelic dosages (VanRaden 2008):

$Z\_{ij}= \frac{M\_{ij}-2p\_{j}}{\sqrt{m(2p\_{j}(1-p\_{j}))}}$,

where $M$is a matrix of allelic dosages, $p\_{j}$is the allelic frequency at SNP marker $j$, $i$ is the $i$th animal, and $m$is the number of markers.

**Multilayer perceptron**

Typical MLP models (Figure 1) consist of an input layer, a variable number of hidden layer(s), and an output layer. Each layer contains several neurons (also known as nodes). Depending on the type of layer, the nature of the nodes will change. For instance, the number of nodes in the input layer is equal to the number of predictor features. In this study, the input layer represents an individual’s genotype, and thus, the input layer will have as many nodes as SNP markers. In Figure 1, there are *M* nodes in the input layer, and its *kth* node will receive input as the allelic count at the *kth* SNP for the *nth* individual ($x\_{n,k}$). The output layer represents the prediction of the response variable produced by MLP. In this case, the output will contain the prediction of an individual’s phenotypic value ($\hat{y}\_{n}$), which can be a continuous outcome, an ordinal outcome, or a categorical outcome.

The nodes in one layer are connected to the nodes in the previous layer by a weighted sum operator. For instance, in Figure 1, the input of the *jth* node in hidden layer 1 is $z\_{j}^{1}=f(\sum\_{k=1}^{M}w\_{jk}^{0}x\_{k})$, where $x\_{k}$ represents the *kth* node from previous layer, the weights $w\_{jk}^{0}$ are unknown and connected to $x\_{k}$ (SNP *k*) and need to be determined through a learning process. $f()$ is the activation function that is specified by the user. It is worth noting that non-linear functions can be used as $f()$ and there is no need to assume linearity as with classic genomic prediction models. Activation functions are detailed in the *Hyperparameters* section further on. Likewise, nodes between layers are fully connected, which means that the input sum of each node in a layer will contain as many terms as there are nodes in the previous layer: $z\_{j'}^{i}=f(\sum\_{j=1}^{nneuron\_{i-1}}w\_{j,j'}^{i-1}z\_{j}^{i-1})$, where $j$ represents the nodes from layer $i-1$, $nneuron\_{i-1}$ is the number of nodes in hidden layer $i-1$, $i$ is the index of hidden layer $i$, $w\_{j,j'}^{i-1}$ is a weight connecting *jth* node ($z\_{j}^{i-1}$) in hidden layer $i-1$ and j’th node in hidden layer $i$ ($z\_{j'}^{i}$).

**Convolutional neural network**

In the context of genomic prediction, the input layer for a single observation in a CNN is a one-dimensional array. Similar to MLP, the input layer will contain an animal’s (*nth* individual’s) genotype and the number of units will be equal to the number of SNP markers. In Figure 2, there are *M* units in the input layer and the *kth* unit represents the allelic count at the *kth*SNP for the nth individual ($x\_{n,k}$). The output layer represents the predicted response value $\hat{y}\_{n}$ for the phenotype or breeding value of the nth individual. After the input layer, a CNN contains a variable number of convolutional layers followed by pooling layers. For instance, in Convolutional Layer 1 of Figure 2, several filters are applied to the nodes of the input layer, where filters are arrays containing certain number of weights to convolve the input. In this case, each filter has three weights $w\_{i,1}^{1}$, $w\_{i,2}^{1}$, and $w\_{i,3}^{1}$, where $i$ represents the $i$th filter defined by the user. These filters are applied to every three consecutive units of the input layer (filter size equal to three). Also, the stride of the filter is equal to its length, which means that the filter is applied to non-overlapping sets of three contiguous SNP. The length of the filter (kernel) is defined by the number of weights to include i.e. the number of units to be convolved by a filter in the input data. An arbitrary number of filters $i$ =1…$ I$ is applied in each convolution. The output of this process will be $I$ feature maps with length equal to $\frac{M-F}{S}+1$, where $M$ represents the number of SNP markers, $F$ is the length of the filter, and $S$ is the stride. In our case, because stride is equal to filter size, the length is simply $\frac{M}{3}$. Moreover, the input of the $j$th unit in feature map 1 is $c\_{j}^{1}=f(w\_{i,1}^{1}x\_{n,k}+w\_{i,2}^{1}x\_{n,k+1}+w\_{i,3}^{1}x\_{n,k+2})$, where $x\_{n,k}$, $x\_{n,k+1}$, and $x\_{n,k+2}$ are allelic dosages of individual$ n$ at three consecutive SNP markers. The weights in the filters are unknown and need to be determined through a DL optimization process. $f()$ is the activation function. In Convolutional Layer 1, the output of each convolution is saved in feature map 1, where the length of each feature map is $a\_{1}=\frac{M}{3}$ and the number of feature maps ($b\_{1}$) is equal to the number of filters (kernels) applied to the input layer (in this case b1=5 in Figure 2). A convolutional layer is followed by a pooling layer for the purposes of dimensionality reduction. In pooling layer 1 of Figure 2, $p^{1}=(p\_{1}^{1},p\_{1}^{2},…, p\_{a\_{1}/2}^{1})$ are elements that are summarized by every two consecutive units generated from the previous convolutional layer and the output will be $b\_{1}$ feature maps with a reduced length equal to $\frac{a\_{1}}{2}$ . Likewise, feature map 2 is followed by convolutional layer 2 where filters with three weights $w\_{i^{'},1}^{2}$, $w\_{i^{'},2}^{2}$, and $w\_{i^{'},3}^{2}$ are applied. In feature map 3, $b\_{2}$ features with a length of $a\_{2}$ are summarized into feature map 4 that has $b\_{2}$ features with length $\frac{a\_{2}}{2}$. If any value among $\frac{M}{3}$, $\frac{a\_{1}}{2}$, $\frac{a\_{1}/2}{3}$, or $\frac{a\_{2}}{2}$ has a remainder, the deficit unit(s) in the input data will be padded with zero(s). The last feature map (feature map 4) is re-arranged into a single vector that has $\frac{b\_{2} × a\_{2}}{2}$ elements. Each element in the re-arranged vector $z^{1}=(z\_{1}^{1},z\_{2}^{1},…, z\_{l}^{1},l=\frac{b\_{2} × a\_{2}}{2})$ is fully connected to a hidden layer (like the ones described in the MLP section of this paper) with $nneuron$ nodes, which are predictors for the output layer.

**Hyperparameters**

The number of hidden layers describes the depth of the network and DL requires at least one hidden layer, and it is known as the depth of the network. In the deep learning literature several studies found that the number of hidden layers, in similar sized problems, can often provide better results with a maximum upper bound of five (Bellot *et al.* 2018; Arifin *et al.* 2019). Thus, we optimized the number of hidden layers by selecting an integer ranging from one through five.

The number of neurons decides the number of units in a fully connected hidden layer, and it is known as the width of the network. Bellot et al. (2018) investigated the influence of neuron numbers on neural networks by varying neurons per layer in four scenarios: 16, 32, 64 and 128, and Pérez-Enciso and Zingaretti (2019) estimated the effect of the number of neurons in the first layer (8, 24, 32, 64, and 128). Both studies used discrete values for the number of neurons. In this study, we optimized the width of the neural network by selecting integers between 8 and 512. The number of neurons is optimized by the DE algorithm for every hidden layer of the MLP and the last hidden layer of the CNN.

Activation function is a function to transform the weighted sums from the previous layer. The aforementioned Pérez-Enciso and Zingaretti (2019) recommended “tanh”, “relu”, “selu” and “sigmoid”. In addition to their reccomendation, we further included “elu”, “softplus” as well as “linear” as possible activation functions.

In deep learning, an optimizer is an algorithm used to alter the attributes of the model e.g. weights and learning rates, where learning rates are coefficients applied to altered weights. Optimizer options included were “sgd”, “adam”, “adagrad”, “rmsprop”, “adadelta”, “adamax”, and “nadam”.

Dropout is used to avoid overfitting due to the large number of weights that need to be estimated. Dropout consists of and randomly sets a proportion (dropout rate) of the neurons in a layer for which their weights are not updated in the current iteration. The dropout rate may affect the predictive performance of a model and we included it in the hyperparameter optimization as a continuous parameter in a range of (0,1).

Another way to ease overfitting is to use weight regularization that adds constraints of weights to the loss function. For instance, in L2 regularization a squared penalty on the values of the weights is added to the loss function. This parameter also may have an effect on the model’s predictive performance. We optimized the L2 regularization as a parameter in a range of (0,1).

L2 Regularization is defined as:

$L\left(\hat{θ}, X, y\right)=\frac{1}{n} \sum\_{i}^{}(y\_{i}-f\_{\hat{θ}}(X\_{i}))^{2}+λ\sum\_{j=1}^{p}\hat{θ}^{2}$ ,

where $L()$ represents loss function, $X$ represents input data, $y$ is the observed response variable, $\hat{θ}$ consists of weights in the deep learning model, $f\_{\hat{θ}}()$ represents the deep learning model, and $λ$ is the L2 regularization parameter.

Epoch refers to the number of iterations where an entire training dataset is passed through the DL model to iteratively adjust weights. Within an epoch the training dataset is further divided into an actual training set for weight adjustment and a testing set that is used for performance evaluation. The number of epochs to be optimized was an integer between 21 and 50. We introduced an early-stopping rule when there is no improvement of the model training for ten consecutive epochs.

Batch size is used to determine the number of randomly partitioned training samples (within an epoch) utilized to update the weights. For the simulated datasets, we first optimized a continuous value α in the range of [0.001-0.01], while the range in the real pig dataset was [0.01, 0.1]. Then, the batch size was defined as the product of training sample size N and α. The number of samples utilized in each DL batch varied according to the training size (N=6539 for the simulated datasets and N=728 for the real pig dataset). This hyperparameter has a profound influence on the computing time and memory required by *TensorFlow* (Abadi *et al.* 2015, https://www.tensorflow.org/) to fit the model. We only optimized batch size in MLP while the batch size was fixed at 32 in CNN because larger batch sizes became computationally too onerous to fit CNN with the larger datasets (N=6539 and M=48,541).

The number of filters, filter (also known as kernel) size and pooling function are hyperparameters exclusive of convolutional neural networks (CNN). A filter is an array of weights used to convolve the input. Typically, multiple filters can be utilized in each layer. Pérez-Enciso and Zingaretti (2019) explored CNN architectures with 16, 32 and 64 filters while Bellot et al. (2018) varied the number of filters with 16, 32, 64 and 128. We optimized the number of filters in CNN by selecting an integer between 4 and 128.

The filter size of a 1d CNN is the number of weights in the filter. Pérez-Enciso and Zingaretti (2019) compared the predictive performance using kernel (filter) sizes of three, five and seven, while Bellot et al. (2018) used filter sizes of two, three, five and ten. In this study, we defined the sample space for filter size as an integer between two and 20.

A pooling layer is used to downsize the feature map that comes from the convolution operation by computing a summary statistical measure of several elements. The typical options for a pooling layer are average, minimum and maximum. Bellot et al. (2018) applied a 1×2 pooling to the feature maps. Similarly, we fixed the size of the pooling to two units in the feature map and optimized the pooling function by selecting between average value or maximum value of the two units. In other words, our models were optimized by selecting one of the pooling rules (average pooling or maximum pooling) to downsize the feature map that comes out of the convolutional operation by half (the two units were summarized into one).

It is necessary to point out that the hyperparameter space for sampling values or options varies across the literature, and it is up to the user to setup the adaptive architecture of the network. With differential evolution (DE) users can not only optimize the subset of hyperparameters used in this study, but can also optimize any other additional hyperparameters they deem relevant.

**Predictive performance of DL models and GBLUP**

Figure S2 shows the predictive performance (correlation between the predicted and the observed response variables in the external validation sets) for each method (optimized MLPs, optimized CNNs, and GBLUP). For the simulated pig dataset, all methods performed similarly, although GBLUP models were slightly better and CNN models were the worst. For the simulated cattle dataset, GBLUP models were better in partitions 1 and 5, while optimized MLPs and CNNs were slightly better in partitions 2, 3 and 4 (with tied performance of MLPs and CNNs). For the real pig dataset, the pattern was completely different, and the best models depended on the data partition. We did not notice a clear improvement in prediction accuracy for any of the models. Since deep learning model fitting results differ even with the same hyperparameters, we ran 30 external (cross) validations using the same hyperparameters and validation sets for all partitions across the three datasets. Each MLP and CNN was trained 30 times independently and repeatedly. Models in both the simulated pig and cattle datasets showed little variation through repetition. However, we observed more variation in the prediction performance for the real pig dataset.

**Literature cited**

Arifin, F., H. Robbani, T. Annisa, and N. N. M. I. Ma’Arof, 2019 Variations in the Number of Layers and the Number of Neurons in Artificial Neural Networks: Case Study of Pattern Recognition. J. Phys. Conf. Ser. 1413: 0–6.