Supplementary Discussion, Methods, Figures, Tables and References

Table of contents

Methods

Hardware

We performed the computation for this project on Harvard Medical School's compute cluster, with access to both central processing units [CPUs] and general processing units [GPUs] (Tesla-M40, Tesla-K80, Tesla-V100) via a Simple Linux Utility for Resource Management [SLURM] scheduler.

Software

We coded the project in Python [1] and used the following libraries: NumPy [2,3], Pandas [4], Matplotlib [5], Plotly [6], Python Imaging Library [7], SciPy [8–10], Scikit-learn [11], LightGBM [12], XGBoost [13], Hyperopt [14], TensorFlow 2 [15], Keras [16], Keras-vis [17], iNNvestigate [18]. We used Dash [19] to code the website on which we shared the results. We set the seed for the os library, the numpy library, the random library and the tensorflow library to zero.

Training, tuning and predictions

Data splitting

We split the 676,787 samples into ten data folds, while keeping all samples from the same participant in the same fold. To ensure this, we split the 502,211 participants' ids (referred to by UKB as "eid") into ten different buckets of the same size. To generate ten folds for each sub-dataset (e.g. brain MRIs), we took the intersection of the samples in each of the ten folds

with the samples for which the sub-dataset data was available. This method had however one important loophole, which is that we could not guarantee that the folds for the sub-datasets would be balanced. For example, brain MRI data was only recorded for 44,183 out of the 502,211 participants. Since the 502,211 participants are split into ten folds, a fold contains approximately 50,221 participants. Although unlikely, we could therefore not guarantee that all or most of the brain MRI samples would be attributed to the first data fold, leading to highly unbalanced folds for the brain MRI analysis. Unbalanced folds can lead to problems during the cross-validation (see further below), as models trained on a smaller number of samples will tend to generalize worse. One solution would have been to use a different split for each dataset, but this would have generated problems when building the ensemble models fold by fold (see Methods - Models ensembling). To mitigate this issue of unbalanced data folds, we developed the following heuristic. We randomly split the 502,211 participants into ten folds, 1,000 times. For each of these 1,000 splits, we computed for each sub-dataset the variance of the percentages of samples in each fold. We then scored each of the 1,000 splits using the maximum of the variance among the different sub-datasets. For example, if the brain MRI samples were not evenly split for the ith split out of the 1,000 splits (e.g. fold 1: 55% of the samples, every other fold: 5% of the samples), the variance of the sample proportions would be high, which would yield a poor score for the ith split. Finally, we selected the split with the lowest score as the final split for the main dataset, and for all the sub-datasets. This selected split had a score of 5.8e-4, which means that the most unbalanced sub-dataset had a variance in its sample size proportion between its ten folds of 5.8e-4.

Scalar data

Nested cross-validation

Cross-validation is a method to tune the regularization of models and prevent overfitting [20]. For the models inputting scalar data (Figure 1A in green), we tuned the hyperparameters and generated a testing prediction for each sample using a nested 10x9-folds cross-validation. We refer to the two nested cross-validations as the “outer” and the “inner” cross-validations. The outer-cross validation is used to generate an unbiased testing prediction for each sample, as opposed to a simple split of the data into a “training+validation” set on one hand, and a testing set on the other hand, which would only generate a testing prediction for one tenth of the dataset. The inner cross-validation is used to tune the hyperparameters more precisely, leveraging the full inner cross-validation dataset as a validation set, as opposed to a simple data split of the “training+validation” dataset into a training and a validation sets, which would only use one data fold as the validation set to estimate the performance associated with a specific combination of hyperparameters. The nested cross-validation is illustrated in Table S26.

Bayesian hyperparameters optimization

To tune the hyperparameters, we used the Tree-structured Parzen Estimator Approach [21] [TPE] of the hyperopt python package [22]. TPE is a sequential Bayesian hyperparameters optimization method that iteratively suggests the next most promising hyperparameters combination as a function of the hyperparameters combinations that have already been tested, by building a probabilistic representation of the objective function. We set the number of iterations to 30. For each model, 30 different hyperparameter combinations are iteratively tested before selecting the best performing one. The hyperparameters names and their ranges defining the

hyperparameters space can be found in Table S25. It might be of interest to other researchers that we initially tuned the hyperparameters using a random search [23] with the same number of iterations, and we did not observe a significant improvement in the model's performance after implementing the Bayesian hyperparameters optimization.

Example

For the sake of clarity, let us walk through a concrete example, which is illustrated in Table S26. Suppose we want to generate unbiased predictions for every sample in a dataset using an elastic net. First, let us generate the testing prediction for the data fold F9, which is performed by the first fold of the outer cross-validation (outer cross-validation fold 0). We select the data fold F9 out of the ten data folds as the testing fold, and we select the remaining nine data folds as "training+validation" folds for the inner cross-validation. We scale and center the target (age) and the predictors using the mean and standard deviation values of the variables on the "training+validation" dataset. We then enter the first inner-cross validation.

For the first inner cross-validation fold, we select the data fold F8 as the validation set, and the remaining eight "training+validation" data folds as the training set. We re-scale and center age and the predictors in the training and the validation sets using the mean and standard deviation values of the training set. We train the model on the eight training data folds with the first hyperparameters combination sampled by the TPE algorithm (one value for alpha and one value for l1_ratio) and generate validation predictions on the validation fold (data fold F8), which we unscale. This completes the first of the nine inner cross-validation folds (Inner CV fold 0). We then permute the nine inner data folds. We scale the age and the predictors using the mean and standard deviation computed on the new training set. Then we train the model with the same

first combination of hyperparameters on eight data folds, leaving aside the data fold F9 (still being used as the testing set for the outer cross-validation) and the data fold F7 (now being used as the validation set for the inner cross-validation). We then use the new trained model to generate validation predictions on the data fold F7, which we unscale. This completes the second of the nine inner-cross validation folds (Inner CV fold 1). We then reiterate these inner permutation and training processes seven more times, until every data fold in the nine "training+validation" data folds is used as the validation set once. At this point, we concatenate the validation predictions from these nine validation folds to obtain the overall validation predictions associated with the first hyperparameters combination, and compute the associated performance metric (e.g. RMSE). This completes the inner-cross validation for the first hyperparameters combination.

We then perform the same 9-folds inner cross-validation, this time with the second hyperparameters combination suggested by the TPE algorithm. We iterate this process 28 more times, until 30 different hyperparameters combinations have iteratively been tested. Next, we select the hyperparameter combination that yielded the best validation performance (e.g. minimum RMSE), and we retrain a model on the whole nine "training+validation" data folds (all data folds except for data fold #1), using this best performing hyperparameters combination. This completes the first inner cross-validation.

We then use the model to generate unbiased predictions on the unseen testing set (data fold F9) and record these predictions. By anticipation for the ensembling algorithm (see Methods - Models ensembling) we also need to compute validation predictions on the data fold F8. We do this by training a model on all the data folds aside from the validation fold (data fold F8) and the

testing fold (data fold F9), with the selected hyperparameters combination. We then use this trained model to compute predictions on the validation fold (data fold F8) and record these predictions, after unscaling them. This completes the first of the ten outer cross-validation folds (outer cross-validation 0).

We then complete the second outer cross-validation fold (outer cross-validation 1), this time using the data fold F8 as the testing dataset, to obtain unbiased testing predictions on this data fold, as well as validation predictions on the data fold F7. We reiterate the process eight more times to obtain the testing and validation predictions on the remaining data folds. We then concatenate the testing predictions from the ten data folds to obtain our final testing predictions for the model. Similarly, we concatenate the validation predictions from the ten data folds to obtain our final testing predictions for the model, which will later be used during ensemble models building and model selection (see Methods - Models ensembling).

The final validation and testing predictions for each data fold are therefore not necessarily associated with the same hyperparameters combination. It is also important to notice that we performed a single outer cross-validation, but that we performed a separate inner-cross validation for each outer cross-validation fold (hence the word “nested”), for a total of ten inner cross-validations per outer cross-validation fold.

Images

Hyperparameters tuning upstream of the cross-validation

The hyperparameters we tuned were the number of added fully connected dense layers, the number of nodes in these layers, their activation function, the optimizer, the initial learning rate, the weight decay, the dropout rate, the data augmentation amplitude and the batch size.

Repeatedly tuning the values of the hyperparameters for different deep neural networks architectures and on the different cross-validation folds would have been prohibitively time and resource consuming. Instead, we sequentially explored how each hyperparameter was affecting the training and validation performances for a single architecture (InceptionV3) on a single cross validation fold (fold #0, see Methods - Training, tuning and predictions - Images - Cross-validation for the detailed description of the cross-validation). We then extrapolated the hyperparameter values to the other architectures, datasets and cross-validation folds. The hyperparameters combinations tested during the tuning can be found in Table S27.

First, we maximized the batch size for each architecture. The maximum number of images per batch depends on the memory of the GPU and the size of the architecture, which itself depends on the dimensions of the image. We used a batch size of 32 for InceptionV3 and 8 for InceptionResNetV2.

Then, we tested the learning rates, including 1e-6, 1e-5, 1e-4, 1e-3, 1e-2 and 1e-1. We observed that learning rates larger than 1e-4 prevented the model from converging for some runs. Second, we did not observe significant differences between the results obtained with learning rates smaller than 1e-4. We therefore set the initial learning rate to be 1e-4 for all models to shorten the time to convergence while ensuring that the learning rate was small enough to allow convergence and the finding of a local minima for the loss function.

Then we tested three different optimizers to perform the gradient descent: Adam [24], Adadelta [25] and RMSprop [26]. We did not observe any significant differences between the optimizers, so we set the optimizer to be Adam.

We then added different numbers of fully connected layers between the base CNN and side CNN's concatenated outputs and the final activation layer. We set the number of nodes to be 1,024 in the first added layer and then decreased the number of nodes by a factor of two for each successive layer. For example, if we added three fully connected layers, the number of nodes was 1024, 512 and 256. We added zero, one and five layers. We did not observe significant differences in the performance of the different architectures, so we set the number of fully connected layers to one.

We then tested powers of two from 16 to 2,048 as the number of nodes in this single layer. We did not observe significant differences between these architectures, so we set the number of nodes to be 1,024 to keep the number close to the initial number of nodes in the imported CNN architectures, as these were initially used to perform classification between 1,000 categories.

We tested two different activation functions for the activation functions of the fully connected layers we added in the side neural network and before the final linear layer. We did not observe any significant differences between the rectified linear units [ReLU] [27] and the scaled exponential linear units [SELU] [28] as activation functions, so we used the more common ReLU.

We then tested different levels of data augmentation. We introduced a hyperparameter that we called "data augmentation factor". The data augmentation factor modulates the amount of variation introduced by the data augmentation, while preserving the ratio between the different transformations. For example, a data augmentation factor of one is equivalent to the default data augmentation (see Preprocessing - Data augmentation - Images), but a data augmentation

factor of two will double the ranges of the possible values sampled and the expected values for the vertical shift, the horizontal shift, the rotation and the zoom on the original images. We tested the following values for the data augmentation factor: 0, 0.1, 0.5, 1, 1.5 and 2. We found that different values for the data augmentation factor hyperparameter yielded similar results, as long as the data augmentation factor was not zero. We therefore set the data augmentation factor to be one when training the final models.

We then tuned the dropout rate for the fully connected layers we added. We tested the following values: 0, 0.1, 0.25, 0.3, 0.5, 0.75, 0.9 and 0.95. We observed that a dropout rate of 0.95 led to underfitting and that smaller values reduced overfitting on the training set but without improving the validation performance. As a consequence, we used a dropout rate of 0.5.

Finally, we tuned the weight decay. We tested the following values: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 5, 10 and 100. For the larger datasets, we found that weight decay values as low as 0.4 could lead to underfitting. We found that lower weight decay values reduced overfitting on the training set without significantly improving the validation performance. We set the weight decay to 0.1.
Altogether, we found that hyperparameter tuning had little effect on the validation performance as long as extreme hyperparameters values were not selected.

Cross-validation

Training deep convolutional neural networks on images and videos is too time and resource consuming to perform a nested cross-validation. Therefore, we tuned the hyperparameters during the preliminary analysis, as described above. After hyperparameters tuning, we performed a simple outer cross-validation to obtain a testing prediction for each sample of the

datasets, but we replaced the inner cross-validation with a simple split between the training fold and the validation fold (Table S28). Although the hyperparameters were already tuned, a validation set was still required for two reasons: (1) to perform early stopping [29], a form of regularization. (2) to generate a set of validation predictions that are necessary for efficient ensemble building (see Methods - Models ensembling) and model selection. During the cross-validation, we scaled and centered the target variable (chronological age) as well as the side predictors (sex and ethnicity) around zero with a standard deviation of one, using the training summary statistics. Scaling the target and the input helps prevent the issues of exploding and vanishing gradients [30,31].

Cross-validation example

For the sake of clarity, let us walk through an example. Let us say that we want to generate unbiased predictions for every sample in a dataset using a CNN. First, we select the data fold #0 as the validation set, the data fold #1 as the testing set, and the remaining data folds (#2-9) as the training set. Then we scale and center the target (age), and the side predictors (sex and ethnicity) using the training mean and standard deviation: for each of the variables, we subtract the training mean to the variable on both the training, the validation and the testing set, and we divide it by the training standard deviation. We then train the model on the training set until convergence and select the architecture's parameters (also known as "weights") associated with the epoch that yielded the lowest validation RMSE. We then use the optimal weights to generate validation predictions for the data fold #0 and testing predictions on the data fold #1. Finally, we unscale the validation and testing predictions by multiplying them by the initial age training standard deviation before adding the initial age training mean to them. This completes the first cross-validation fold.

We then reiterate the process, this time using the data fold #1 as the validation set, the data fold #2 as the testing set, and the remaining data folds (#0 and #3-9) as the training set. We use the optimized weights to generate the validation predictions on the data fold #2, and the testing predictions on the data fold #3. We unscale the validation and testing predictions. This completes the second cross-validation fold. We reiterate the process eight more times to complete the cross-validation. We then concatenate the validation predictions from the ten data folds to obtain the final validation predictions, and the testing predictions from the ten data folds to obtain the final testing predictions.

Generating average predictions for each participant

We generated an average prediction for each individual, reported to UKB's instance 0. We walk through an example. Let us assume a participant had two brain MRI samples collected from them in instances 2 and 3, respectively at age 70 and 80. Let us assume that the age predictions were respectively 64 and 78, so the residuals are respectively -6 years and -2 years, for an average of -4 years. However, we still need to take into account the bias in the residuals, defined as the difference between the participant's chronological age and the prediction. As explained in more details under Methods - Biological age definition, we observed a bias in the residuals as a function of chronological age. Participants on the younger end of the chronological age distribution tend to be predicted older than they actually are, whereas participants on the older end of the distribution tend to be predicted younger than they actually are. We need to properly account for this bias when translating a prediction from a more recent instance to an older instance. Let us assume that the average bias in the residuals for participants who are 70 and 80 years old is respectively -2 years and -4 years. After correcting

for this bias, the predictions are now respectively 64-(-2)=66 and 78-(-4) = 82. Therefore, the corrected residuals for this participant are respectively -4 years and +2 years, for an average of -1 years. Finally, let us assume that the participant was 60 years old in instance 0. We will assign a single prediction of 60-1=59 years to the participant, but we still need to un-correct for the bias in residuals. Let us assume that the average bias for the residuals at age 60 is +5 years. We will assign a final prediction for the participant of 59+5=64 years. This new set of predictions reported on the instance 0 is more likely to have a non-zero sample size overlap with other predictors based on datasets collected on instance 0 (e.g. cognitive biomarkers) and can therefore be leveraged by the ensemble builder.

A key point we would like to highlight here is that we did not actually correct for the bias in the residuals at this step of the pipeline. Instead, we corrected then un-corrected the predictions that we translated from different instances to the instance 0. The actual correction for the residual biases takes place when defining the biological age phenotypes (see Methods - Biological age definition).

To distinguish between raw predictions on the instance 0, and the average predictions reported to the instance 0, we created a new instance which we named instance “*”. We refer to these predictions as “participants predictions”, as opposed to “samples predictions”.

Interpretability of the predictions

Scalar data-based predictors

For elastic nets, we interpreted the models using the values of the regression coefficients. Large absolute values for these coefficients means they played an important role when generating the predictions. For gradient boosted machines we used the feature importances, which are based on the number of times a tree selected each of the variables. Variables with high feature importances were selected more often and are therefore likely to play a key role in predicting chronological age. For neural networks, we estimated the importance of each feature by permuting it randomly between samples before computing the performance of the model. The score of each feature is the difference between the R-Squared value before and after the random permutations. Features whose random permutation leads to a large decrease in the model's performance are estimated to be important predictors of chronological age.

We estimated the concordance between the three different algorithms by computing the Pearson and the Spearman correlations between their feature importances.

Image-based predictors

To interpret the CNNs built on images, we first used saliency maps [32], which we coded using the keract python library. For each input sample, a saliency map uses the gradient of the final prediction with respect to each individual input pixel to estimate whether changing the value of this pixel would affect the prediction. Pixels for which the gradient is close to zero are not important, whereas pixels with a large gradient are estimated to be important.

We then built a second attention map using a custom version of the Gradient-weighted Class Activation Mapping [Grad-CAM] algorithm [33] adapted to regression rather than multi-class classification: Gradient-weighted Regression Activation Mapping [Grad-RAM]. The intuition behind Grad-CAM maps is that they are similar to saliency maps [33], but instead of computing the gradient with respect to the input image, they compute it with respect to the activation of the last convolutional layer. As convolutional layers maintain the spatial organization of the input image, Grad-CAM can still identify which region of the image is driving the predictions. Because Grad-CAM does not have to backpropagate the gradient all the way back to the input image, it is considered a less noisy alternative to the saliency maps. In the same way that saliency maps need to combine the attention maps generated in the different input channels (e.g. RGB) into a single activation map, Grad-CAM must combine the attention maps generated on the different filters of the last convolutional layer. For example, the last convolutional layer for InceptionResNetV2 has 1,792 filters. Grad-CAM combines these 1,792 attention maps into a single attention map using a linear combination. In the initial Class Activation Mapping [CAM] algorithm [34], generating CAM activation maps required to retrain the model after modifying the architecture and replacing all the fully connected layers after the final convolutional layer with a global max pooling operation, which converted each filter into a scalar feature. The intuition behind this substitution was that each filter could be interpreted as detecting a specific feature, and global max pooling yielded a scalar that could be interpreted as the presence (high value) or absence (low value) of the feature anywhere on the image. The scalar values were then linearly combined and activated using the softmax function to yield the probabilities of belonging to different classes. To obtain the activation map for a specific class, the filters of the last convolution layer were linearly combined using the weights connecting the scalar features

obtained after the max pooling operation to the final prediction score for that class. CAM was later improved to become Grad-CAM [33]. Grad-CAM saves the need for modifying the architecture of the model and retraining it by approximating the linear regression weight for each final convolutional filter by the mean activation gradient over the pixels of the filter. The intuition behind this approximation is that a filter's pixel is important if changing its value affects the final prediction, so a high average gradient over the pixels of the filter justifies that this filter should be given a higher weight when merging all the filters into a single attention map. To adapt Grad-CAM to our regression task we (1) computed the derivatives of the chronological age prediction rather than a class' prediction and (2) removed the ReLU activation applied to the weighted sum of the last convolutional filters, which we replaced by an absolute value. The rationale is that for (Grad-)CAM maps, we only want to highlight the regions of the picture which are associated with a high probability for the class. In contrast, for (Grad-)RAM we care as much about the regions of the input image that can strongly increase the chronological age prediction as about the regions that can strongly decrease it. Because the filters in the last convolutional layer are the result of the processing of the input image by several convolutional layers with possibly negative weights, the sign of the last convolutional layer's pixels and regression weights cannot be linked to either accelerated aging or decelerated aging, only to the magnitude of the shift that would affect the prediction if each region of the input image was modified. Regression Activation Mapping (RAM) was mentioned as a possible extension of CAM in the original CAM publication [34] and has been used to interpret models CNNs built on retinal images [35] and cortical surfaces [36], but we are to our knowledge the first to describe the generalization of Grad-CAM to a regression task. One notable difference between our implementation and Wang and Yang.'s implementation [35] is that we are taking the absolute value of the final attention map, as mentioned above. We found that not taking the absolute value led to misleading attention

maps for participants with high chronological age predictions. The attention map highlights important areas with negative values, which are therefore depicted in blue, a color otherwise associated with unimportant regions in traditional CAMs. Inversely, regions on the input image for which the attention map has a slight positive value are spuriously considered to be the most important and are highlighted in red. We therefore advise that RAM or Grad-RAM be implemented using an absolute value. We coded Grad-RAM using the get_activations and get_gradients_of_activations functions of the keract python library.

It is important to understand that unlike the feature importances described under “Scalar data-based predictors”, which describe the model itself, attention maps are sample specific. In other words, they can be used to explain which features drove the predictions for a specific inputted sample but cannot provide an explanation for the way the model is performing predictions in general.

For each aging subdimension, we generated the attention maps for the best performing CNN architecture. We selected representative samples for which we computed the different attention maps. We computed attention maps for the two sexes (female and male), for three age ranges (ten youngest ages, ten middle ages and ten oldest ages of the chronological age distribution) and for three aging rates (accelerated agers, normal agers, decelerated agers). For each intersection of the three categories listed above, we selected the ten most representative samples (e.g. the ten most accelerated agers among young males). The figures in this paper only present the first, most representative of these ten samples. The complete set of samples can be found on the website.

Non-genetic correlates of accelerated aging

Unlike DNA, biomarkers, phenotypes, diseases, family history, environmental variables and socioeconomics can change over life. As a consequence, we compared each biomarker, phenotype and environmental variable with the accelerated aging of the participant at the time the exposure was measured and we used the “Samples predictions”, as opposed to the “Participants predictions” that we used for the identification of genetic correlates (see Methods - Models ensembling - Generating average predictions for each participant).

Imputation of the non-genetic X-variables

Most X-variables were not collected on all four instances. Additionally, no X-variables were collected at the same time as the accelerometer data was collected. To identify the non-genetic correlates of accelerated aging, we had to impute the values of the X-variables for the ages of the participants for which they were not available. We considered two imputation methods, which we refer to as the “cross-sectional” and the “longitudinal” imputations.

For the cross-sectional imputation, we computed a linear regression for each X variable as a function of age, adjusting for sex. We then used the slope of the linear regression to extrapolate the value of the XWAS variable at different ages.

For the longitudinal imputation, we first selected, for each X variable, all the participants that had at least two measures taken for this X variable. We then performed a linear regression for each participant. We then averaged the slope of the linear regressions over all the participants of the same sex. Finally, we used this slope to extrapolate the value of the XWAS variable at different

ages for all participants depending on their sex, in the same way we did it for the cross-sectional imputation.

It is important to notice that for both the cross-sectional imputation and the longitudinal imputation, data can only be imputed when the XWAS variable has been measured at least once for the participant. This raw measure is then used to extrapolate which value the X variable was likely taking a couple years earlier and/or later.

The advantage of the cross-sectional imputation is larger sample sizes. The advantage of the longitudinal method is that it corrects for generational effects. For example, old people have shorter legs than young people on average [37]. This is not because human legs shrink as we grow older. Instead, people who are old today already had shorter legs when they were young. If the cross-sectional regression is used to impute the length of the participants on instances where it was not measured, it will spuriously assign smaller values to the older samples. In contrast, the longitudinal regression learns the regression coefficient by comparing each participant to themselves as they age and will therefore not capture the generational effect. When used to predict the participants legs' length, it will impute constant values over time. To evaluate which of the two imputation methods should be preferred, we used them to predict X-variables for which we knew the actual values and computed the R-Squared values associated with the predictions. We found that, even with sample sizes as small as 200 samples, longitudinal imputation outperformed cross-sectional imputation. We therefore used longitudinal imputation.

X-Wide Association Studies

First, we tested for associations in an univariate context by computing the partial correlation between each X-variable and brain aging dimensions. To compute the partial correlation between an X-variable and an aging, we followed a three steps process. (1) We ran a linear regression on each of the two variables, using age, sex and ethnicity as predictors. (2) We computed the residuals for the two variables. (3) We computed the correlation between the two residuals and the associated p-value if their intersection had a sample size of at least ten samples. We used a threshold for significance of 0.05 and corrected the p-values for multiple testing using the Bonferroni correction. We plotted the results using a volcano plot. We refer to this pipeline as an X-Wide Association study [XWAS].

In the supplementary tables and the results, we rank the X-variables subcategories by decreasing percentage of variables associated with accelerated aging (note that the ranking is therefore biased towards categories with fewer variables). For each subcategory, we list the three most associated variables, based on the absolute value of the correlation coefficient. For the exhaustive list, please refer to https://www.multidimensionality-of-aging.net/xwas/univariate_associations.

Supplementary Figures

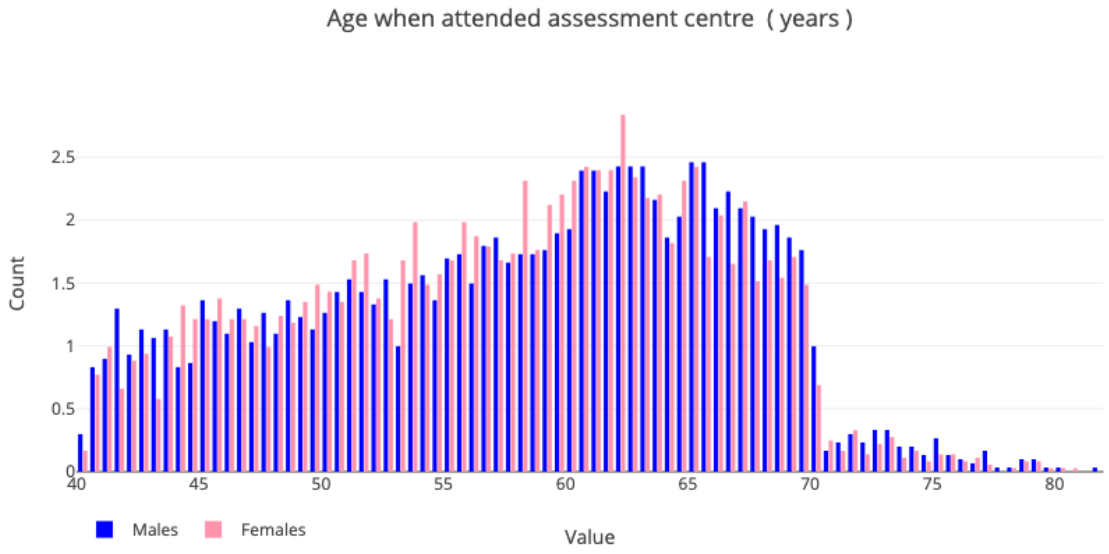

Figure S1: Demographics of the UK Biobank cohort

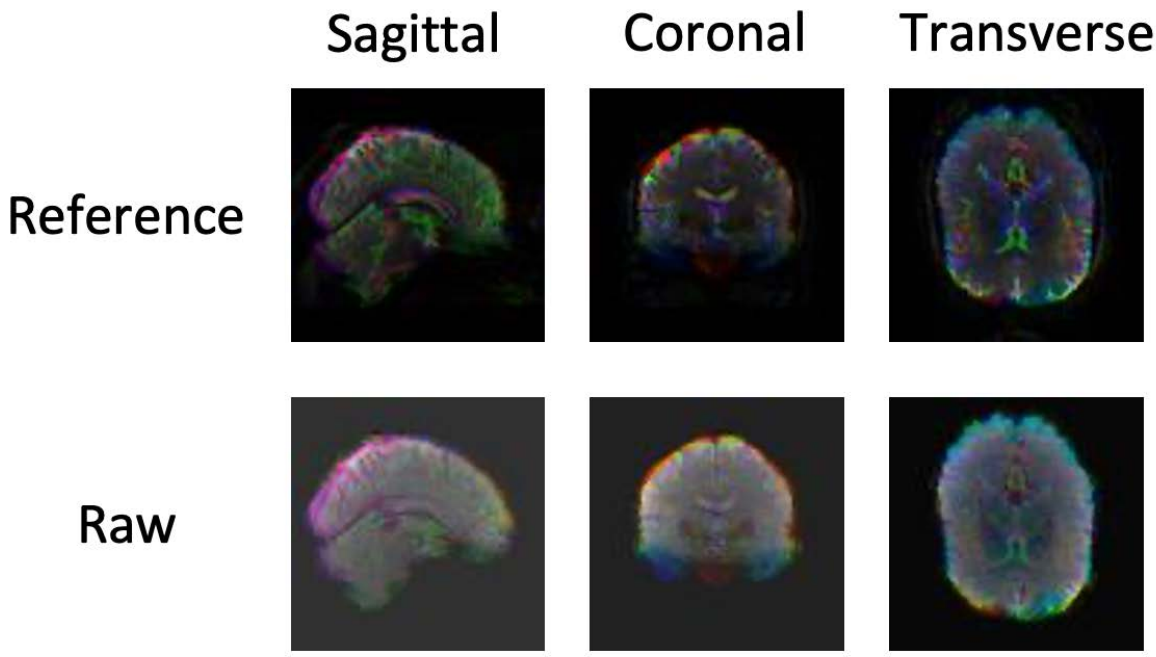

Figure S2: **Six images (three views and two preprocessings) generated from a brain MRI 3D video**

The participant is a 60-65-year-old male.

Supplementary Tables

Table S1: Comparison between the models trained on scalar features

Brain dimension	Subdimension	Number of Predictors (non-demographics)	ElasticNet (R-Squared)	GBM (R-Squared)	Neural Network (R-Squared)
Anatomical	GreyMatterVolumes	139	0.494±0.012	0.505±0.009	0.511±0.011
Anatomical	SubcorticalVolumes	14	0.274±0.006	0.297±0.008	0.284±0.009
Anatomical	dMRIWeightedMeans	243	0.528±0.014	0.531±0.014	0.542±0.020
Anatomical	AllScalars	396	0.622±0.011	0.620±0.010	0.627±0.017
Cognitive	ReactionTime	14	0.112±0.004	0.158±0.003	0.145±0.003
Cognitive	MatrixPatternCompletion	4	0.136±0.011	0.138±0.010	0.133±0.011
Cognitive	TowerRearranging	2	0.083±0.009	0.084±0.009	0.074±0.018
Cognitive	SymbolDigitSubstitution	2	0.230±0.012	0.248±0.011	0.237±0.013
Cognitive	PairedAssociativeLearning	46	0.072±0.010	0.078±0.011	0.064±0.014
Cognitive	ProspectiveMemory	14	0.122±0.017	0.176±0.003	0.157±0.010
Cognitive	NumericMemory	3	0.045±0.021	0.092±0.006	0.089±0.007
Cognitive	FluidIntelligence	37	0.045±0.003	0.048±0.003	0.045±0.002
Cognitive	TrailMaking	4	0.128±0.011	0.241±0.008	0.221±0.010
Cognitive	PairsMatching	10	0.082±0.004	0.138±0.003	0.134±0.004
Cognitive	AllScalars	135	0.340±0.014	0.378±0.012	0.308±0.060
All	Scalars	531	0.655±0.013	0.648±0.013	0.615±0.034

Table S2: Feature importances for the models built on all cognitive test features

See supplementary data

Table S3: Feature importances for the models built on all anatomical scalar features

See supplementary data

Table S4: Pearson correlations between the feature importances for different scalar features-based algorithms

Brain dimension	Subdimension	Correlation vs. ElasticNet	Correlation vs. GBM	Correlation vs. NeuralNetwork	ElasticNet vs. GBM	ElasticNet vs. NeuralNetwork	GBM vs. NeuralNetwork
Anatomical	GreyMatterVolumes	0.236	0.348	0.079	0.836	0.861	0.662
Anatomical	SubcorticalVolumes	0.656	0.699	0.024	0.72	0.095	-0.048
Anatomical	dMRIWeightedMeans	0.11	0.119	0.173	0.428	0.74	0.308
Anatomical	AllScalars	0.107	0.037	0.085	0.427	0.753	0.296
Cognitive	ReactionTime	0.917	0.885	0.871	0.895	0.954	0.939
Cognitive	MatrixPatternCompletion	0.873	0.657	0.451	0.687	0.434	0.253
Cognitive	TowerRearranging	0.938	0.958	0.429	0.899	0.377	0.358
Cognitive	SymbolDigitSubstitution	0.901	0.982	0.629	0.841	0.754	0.544
Cognitive	PairedAssociativeLearning	0.695	0.568	0.4	0.473	0.33	0.495
Cognitive	ProspectiveMemory	0.884	0.738	0.265	0.862	0.221	0.318
Cognitive	NumericMemory	0.658	0.399	0.561	0.849	0.614	0.744
Cognitive	FluidIntelligence	0.635	0.139	0.515	0.42	0.522	0.187
Cognitive	TrailMaking	0.941	0.954	0.344	0.888	0.326	0.24
Cognitive	PairsMatching	0.771	0.783	0.727	0.736	0.477	0.411
Cognitive	AllScalars	0.737	0.722	0.665	0.707	0.904	0.551
All	Scalars	0.24	0.346	0.151	0.507	0.757	0.568

Table S5: Spearman correlations between the feature importances for different scalar features-based algorithms

Brain dimension	Subdimension	Correlation vs. ElasticNet	Correlation vs. GBM	Correlation vs. NeuralNetwork	ElasticNet vs. GBM	ElasticNet vs. NeuralNetwork	GBM vs. NeuralNetwork
MRI	GreyMatterVolumes	0.27	0.388	0.151	0.714	0.836	0.548
MRI	SubcorticalVolumes	0.722	0.681	0.403	0.725	0.36	0.415
MRI	dMRIWeightedMeans	0.048	0.098	0.184	0.427	0.532	0.24
MRI	AllScalars	0.126	0.028	0.102	0.347	0.72	0.271
Cognitive	ReactionTime	0.7	0.535	0.732	0.58	0.731	0.818
Cognitive	MatrixPatternCompletion	0.647	0.531	0.659	0.464	0.312	0.54
Cognitive	TowerRearranging	0.611	0.603	0.681	0.383	0.352	0.89

Cognitive	SymbolDigitSubstitution	0.635	0.628	0.237	0.516	0.305	0.484
Cognitive	PairedAssociativeLearning	0.555	0.571	0.611	0.397	0.37	0.71
Cognitive	ProspectiveMemory	0.653	0.56	0.617	0.488	0.38	0.643
Cognitive	NumericMemory	0.833	0.652	0.684	0.581	0.616	0.826
Cognitive	FluidIntelligence	0.452	0.341	0.318	0.395	0.187	0.699
Cognitive	TrailMaking	0.641	0.666	0.346	0.566	0.293	0.432
Cognitive	PairsMatching	0.725	0.849	0.831	0.776	0.515	0.783
Cognitive	AllScalars	0.585	0.671	0.618	0.566	0.728	0.596
All	Scalars	0.307	0.411	0.271	0.563	0.688	0.53

Table S6: **List of biomarkers by subcategories for the Biomarkers Wide Association Study [BWAS]**

See supplementary data

Table S7: Biomarkers most associated with accelerated aging for each brain aging dimension

See supplementary data

Table S8: Biomarkers most associated with decelerated aging for each brain aging dimension

See supplementary data

Table S9: List of clinical phenotypes by subcategories for the Clinical Phenotypes Wide Association Study [CWAS]

See supplementary data

Table S10: Clinical phenotypes most associated with accelerated aging for each brain aging dimension

See supplementary data

Table S11: Clinical phenotypes most associated with decelerated aging for each brain aging dimension

See supplementary data

Table S12: List of diseases by subcategories for the Diseases Wide Association Study [DWAS]

See supplementary data

Table S13: Diseases most associated with accelerated aging for each brain aging dimension

See supplementary data

Table S14: Diseases most associated with decelerated aging for each brain aging dimension

See supplementary data

Table S15: List of family history variables by subcategories for the Family History Phenotypes Wide Association Study [FWAS]

See supplementary data

Table S16: Family history variables most associated with accelerated aging for each brain dimension

See supplementary data

Table S17: Family history variables most associated with decelerated aging for each brain dimension

See supplementary data

Table S18: List of environmental variables by subcategories for the Environmental Wide Association Study [EWAS]

See supplementary data

Table S19: Environmental variables most associated with accelerated aging for brain each aging dimension

See supplementary data

Table S20: Environmental variables most associated with decelerated aging for each brain aging dimension

See supplementary data

Table S21: List of socioeconomic variables by subcategories for the Socioeconomics Wide Association Study [SWAS]

See supplementary data

Table S22: Socioeconomic variables most associated with accelerated aging for each brain aging dimension

See supplementary data

Table S23: Socioeconomic variables most associated with decelerated aging for each brain aging dimension

See supplementary data

Table S24: Comparison between our age predictors and the literature in terms of prediction performance

Brain dimension	Our model				Model(s) in the literature					
	R-Squared (%)	RMSE (years)	Sample size	Age range (years)	R-Squared (%)	RMSE (years)	Algorithm	Sample size	Age range (years)	Authors
Cognitive	37.9±0.4	5.95±0.02	34,055	47.0-82.7	9	4.77	XGBoost	537	N/A. 69.75±5.08 (mean age)	Anatürk et al.
MRI	76.4±0.3	3.73±0.01	38,815	45.1-82.5	92; 61.4	5.31; 3.63 (MAE)	3D CNN; 3D CNN	2,001; 12,395	18-90; 46-79	Cole et al.; Jonsson et al.
EEG	N/A				67.24; 37±6	7.6 (MAE); 8.46±0.59	Linear regression (after feature engineering); SVR with radial kernel	18-80; 18-58	4506; 468	Sun et al.; Zoubi et al.

Table S25: Hyperparameter space for scalar features-based models Bayesian optimization

Algorithm	Hyperparameter	Scale	Low	High
Elastic net	alpha	loguniform	-10	0
	l1_ratio	uniform	0	1
Gradient Boosted Machine	num_leaves	quniform	5	45
	min_child_samples	quniform	100	500
	min_child_weight	loguniform	-5	4
	subsample	uniform	0.2	0.8

	colsample	uniform	0.4	0.6
	reg_alpha	loguniform	-2	2
	reg_lambda	loguniform	-2	2
	n_estimators	quniform	150	450
Neural network	learning_rate_init	loguniform	-5	-1
	apha	loguniform	-6	3

Table S26: Nested Cross-Validation pipeline

METHODS:

General comments

The Nested Cross-Validation is a normal double split train/val/test method, for which not just one, but the two splits are replaced with a CV instead.

Outer Cross-Validation

The Outer Cross-Validation is used to replace the split between train + val on one side, vs. test on the other side. Thanks to the Outer Cross-Validation, an unbiased testing prediction can be obtained for every sample.
Three common sources of confusion clarified:
Yes, using an Outer Cross-Validation does mean that the predictions do not all come from the same model. (For exemple for testing, the prediction for each fold comes from a different model.)
No, the Outer Cross-Validation has **"nothing"** to do with hyperparameter tuning or model selection.
Tuning a model only requires one Outer Cross-Validation. In contrast, it requires N_CV_folds distinct Inner Cross-Validations, one for each fold of the Outer Cross Validation. (see below)

Inner Cross-Validation

The Inner Cross-Validation is used to replace the split between the typical training set and validation set.
Thanks to the Inner Cross-Validation, hyperparameter selection can be decided based on a performance calculated on the full training set and validation set sample size, instead of the sole validation set sample size.
Two common sources of confusion clarified:
#1 - The whole purpose of performing an Inner Cross-Validation is **"hyperparameter selection"**.
The Inner Cross-Validation tunes the model that will then be used to generate predictions on the left out testing fold.
Therefore, the Inner Cross-Validation does **"not"** generate a testing prediction for more samples. That is the role of the Outer Cross-Validation.
One last time, the Inner Cross-Validation is **"only"** about improving the hyperparameter tuning of the model.
#2 - Tuning a model requires N_CV_folds distinct Inner Cross-Validations, one for each fold of the Outer Cross Validation. In contrast, it only requires one Outer Cross-Validation. (see above)

Special Fold 0

Special "All folds" permutation is only here for feature importance purposes.
We could extract the features importance from the 10 models used to generate the predictions on the testing set, but we would have to take the average of features importances between several model.
Additionally each of these models only saw 90% of the data (10% was left aside for testing purposes).
If we only care about feature importance, we actually do not need to leave samples aside for testing. We can use 100% of the data to train a model using a 10-fold (not 9-fold this time, since the test fold can be used too!) **"Inner"** Cross-Validation to train a single model.
This model will never be tested, but it does not matter: its role is to provide the best estimate for the feature importance, which it does well.
Two common sources of confusion clarified:
#1 - The Nested Cross-Validation pipeline and the Special "All Folds" permutation pipeline are **"not"** intertwined and serve different purposes.
The purpose of the Nested Cross-Validation pipeline is to generate accurate and a unbiased testing prediction for every sample, whereas the purpose of the Special "All Folds" pipeline is to generate feature importance.
#2 - **Yes**, it does mean that the feature importance reported do not correspond to any of the 10 models (trained in the 10 Outer Cross-Validation folds) that generated the final testing predictions.
The feature importance comes from a model that was never used to generate predictions.

RESULTS:

	Example: cells corresponding to the values used for each prediction set	**Explanation:**
Training	"E17=G17+I17+K17+M17+O17+Q17+S17+ E29=G29+I29+K29+M29+O29+Q29+U29+..."	To obtain the training prediction for the Data Fold F1, take the mean of every training prediction available on this data fold (10*8=80 of them).
Validation	"E25"	To obtain the validation prediction for the Data Fold F1, take the prediction outputted by the model trained on the remaining 8 data folds with the best hyperparameters values obtained for this Outer Cross-Validation fold. These validation predictions can then be used to tune the ensemble models. /!\Warning /!\: This can only be properly done if data folds split used for the Outer Cross-Validation and the Inner Cross-Validation is the same.
Testing	"C26"	No extra step is needed for the testing prediction for the Data Fold 1. Simply take the only testing prediction available for this fold.
Feature importance:	N/A	Extract the features importance from the special "outer fold" "All". If want to compute the standard deviation for the feature importance as well, we can leverage the feature importance computed on each of the Outer Cross-Validation folds and take the square root of their variance.

SUPPLEMENTARY:

Comparison between Nested Cross-Validation and other validation methods:
#1-Single split train/test.
Split train/test works when there are no hyperparameters to tune. If one uses a simple train/test split and tune the hyperparameters on the testing set, some overfitting is to be expected.
#2-Double split train/validation/test.
Because of the above, a train/validation/test double split is commonly used when hyperparameters tuning is required.
The training is performed on the training set, the tuning of the hyperparameters on the validation set, and the evaluation of the unbiased model performanc on the untouched test dataset.
#3-Cross-Validation(train/val) and split (train+val)/test.
With the pipeline #2 (double split train/validation/test), the tuning of the hyperparameters is not leveraging the dataset as much as with a Cross-Validation, as only one data fold is used to tune the hyperparameters.
Therefore machine learning practitioners usually replace the "inner split" (the training/validation split), by a Cross-Validation instead, but they keep the "outer split" (between the training+validation dataset used for their Cross-Validation and the testing set).
#4-Split train/val and Cross-Validation(train+val)/test).
Pipeline #3 only generate an unbiased testing prediction for a fraction of the dataset (the testing set).
To generate an unbiased prediction on every sample of the dataset, pipeline #4 can be used.
However, because there is a simple split between the training set and the validation set (one for each Cross-Validation fold), that means that the hypperparametier tuning only leverages a fraction of the dataset.
Pipeline #4 is preferred when generating an unbiased testing prediction for every sample is needed but the models are too time consuming to train to be able to afford a Nested Cross-Validation (#5).
Pipeline #4 is therefore the pipeline we used to predict chronological age using medical images or videos, for example.
#5-Nested Cross-Validation.
To fully leverage the whole dataset, a Nested Cross-Validation replaces both the outer split (train+val/test) and the inner splits (train/val) by a Cross-Validation, hence the nesting.
It requires the training of a large number of models, and it is therefore usually reserved for models that can be trained relatively quickly.
In our case, we used it to train our scalar features-based models (elastic nets, light gradient boosted machines, and shallow fully connected neural networks).

Table S27: Hyperparameters tuning upstream of the cross-validation for images-based models

See supplementary data

Table S28: Outer Cross-Validation with inner split pipeline

The values displayed are validation RMSE values. Lower values are associated with better hyperparameter tuning. When two values are displayed (value1/value2), the second value corresponds to the training RMSE. The architecture used was InceptionV3, with an initial learning rate of 0.001. The model was trained on the data folds 2-9, and validated on the data fold #0. The data fold #1 was set aside as the testing set and was not used.

Data Fold	N=100	Outer CV folds 0	Outer CV folds 1	Outer CV folds 2	Outer CV folds 3	Outer CV folds 4	Outer CV folds 5	Outer CV folds 6	Outer CV folds 7	Outer CV folds 8	Outer CV folds 9
F0	N=10	Train	Train	Train	Train	Train	Train	Train	Train	Validation	Test
F1	N=10	Train	Train	Train	Train	Train	Train	Train	Validation	Test	Train
F2	N=10	Train	Train	Train	Train	Train	Train	Validation	Test	Train	Train
F3	N=10	Train	Train	Train	Train	Train	Validation	Test	Train	Train	Train
F4	N=10	Train	Train	Train	Train	Validation	Test	Train	Train	Train	Train
F5	N=10	Train	Train	Train	Validation	Test	Train	Train	Train	Train	Train
F6	N=10	Train	Train	Validation	Test	Train	Train	Train	Train	Train	Train
F7	N=10	Train	Validation	Test	Train	Train	Train	Train	Train	Train	Train
F8	N=	Validati	Test	Train	Train	Train	Train	Train	Train	Train	Train

	10	on									
F9	N=10	Test	Train	Train	Train	Train	Train	Train	Train	Train	Validation

Supplementary References

1. Van Rossum, G. & Drake, F. L. *The Python Language Reference Manual*. (Network Theory Limited, 2011).
2. Oliphant, T. E. *A guide to NumPy*. vol. 1 (Trelgol Publishing USA, 2006).
3. Walt, S. van der, van der Walt, S., Chris Colbert, S. & Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering* vol. 13 22–30 (2011).
4. McKinney, W. & Others. Data structures for statistical computing in python. in *Proceedings of the 9th Python in Science Conference* vol. 445 51–56 (Austin, TX, 2010).
5. Hunter, J. D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **9**, 90–95 (2007).
6. Inc, P. T. Collaborative data science. *Montreal: Plotly Technologies Inc Montral* (2015).
7. Clark, A. Pillow Python Imaging Library. *Pillow—Pillow (PIL Fork) 5. 4. 1 documentation* (2018).
8. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* vol. 17 261–272 (2020).
9. Oliphant, T. E. Python for Scientific Computing. *Computing in Science Engineering* **9**, 10–20 (2007).
10. Millman, K. J., Jarrod Millman, K. & Aivazis, M. Python for Scientists and Engineers. *Computing in Science & Engineering* vol. 13 9–12 (2011).
11. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* **12**, 2825–2830 (2011).
12. Ke, G. *et al.* LightGBM: A Highly Efficient Gradient Boosting Decision Tree. in *Advances in*

Neural Information Processing Systems 30 (eds. Guyon, I. et al.) 3146–3154 (Curran Associates, Inc., 2017).

13. Chen, T. & Guestrin, C. XGBoost: A Scalable Tree Boosting System. in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 785–794 (Association for Computing Machinery, 2016).
14. Bergstra, J., Yamins, D. & Cox, D. D. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. in *Proceedings of the 12th Python in science conference* vol. 13 20 (Citeseer, 2013).
15. Abadi, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems. (2015).
16. Chollet, F. & Others. keras. (2015).
17. Kotikalapudi, R. & Others. keras-vis. 2017. *URL https://github. com/raghakot/keras-vis* (2019).
18. Alber, M. *et al.* iNNvestigate neural networks. *J. Mach. Learn. Res.* **20**, 1–8 (2019).
19. Hossain, S., Calloway, C., Lippa, D., Niederhut, D. & Shupe, D. Visualization of Bioinformatics Data with Dash Bio. in *Proceedings of the 18th Python in Science Conference* 126–133 (2019).
20. Kohavi, R. & Others. A study of cross-validation and bootstrap for accuracy estimation and model selection. in *Ijcai* vol. 14 1137–1145 (Montreal, Canada, 1995).
21. Bergstra, J. S., Bardenet, R., Bengio, Y. & Kégl, B. Algorithms for Hyper-Parameter Optimization. in *Advances in Neural Information Processing Systems 24* (eds. Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. & Weinberger, K. Q.) 2546–2554 (Curran Associates, Inc., 2011).
22. Bergstra, J., Yamins, D. & Cox, D. Making a Science of Model Search: Hyperparameter

Optimization in Hundreds of Dimensions for Vision Architectures. in (eds. Dasgupta, S. & McAllester, D.) vol. 28 115–123 (PMLR, 2013).

23. Bergstra, J. & Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **13**, 281–305 (2012).
24. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv [cs.LG]* (2014).
25. Zeiler, M. D. ADADELTA: An Adaptive Learning Rate Method. *arXiv [cs.LG]* (2012).
26. Hinton, G. Slide 29 of Lecture 6, Geoffrey Hinton coursera's class. *http://www.cs.toronto.edu* http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
27. Nair, V. & Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. (2010).
28. Klambauer, G., Unterthiner, T., Mayr, A. & Hochreiter, S. Self-Normalizing Neural Networks. in *Advances in Neural Information Processing Systems 30* (eds. Guyon, I. et al.) 971–980 (Curran Associates, Inc., 2017).
29. Prechelt, L. Early Stopping - But When? in *Neural Networks: Tricks of the Trade* (eds. Orr, G. B. & Müller, K.-R.) 55–69 (Springer Berlin Heidelberg, 1998).
30. Hochreiter, S. Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München* **91**, (1991).
31. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. & Others. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. (2001).
32. Alqaraawi, A., Schuessler, M., Weiß, P., Costanza, E. & Berthouze, N. Evaluating saliency map explanations for convolutional neural networks: a user study. in *Proceedings of the 25th International Conference on Intelligent User Interfaces* 275–285 (Association for Computing Machinery, 2020).

33. Selvaraju, R. R. *et al.* Grad-cam: Visual explanations from deep networks via gradient-based localization. in *Proceedings of the IEEE international conference on computer vision* 618–626 (2017).
34. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. & Torralba, A. Learning deep features for discriminative localization. in *Proceedings of the IEEE conference on computer vision and pattern recognition* 2921–2929 (2016).
35. Wang, Z. & Yang, J. Diabetic Retinopathy Detection via Deep Convolutional Networks for Discriminative Localization and Visual Explanation. *arXiv [cs.CV]* (2017).
36. Duffy, B. A. *et al.* Regression activation mapping on the cortical surface using graph convolutional networks. (2019).
37. Le Goallec, A. & Patel, C. J. Age-dependent co-dependency structure of biomarkers in the general population of the United States. *Aging* **11**, 1404–1426 (2019).