COMP9417 Project: MAPnet

Forrest Koch z3463797

August 10, 2019

1 Introduction

There is significant interest in the medical community for a generalized "brain age" measure as a tool to communicate overall health to patients. Think of the doctor informing a smoker that they have "the lungs of a 70 year old" despite only being 40. Brain health is expected to decline naturally with age; however, this decline is by no means constant across a population, and is heavily influenced by genetic and lifestyle factors (Kalaria et al. 2008). Therefore, a tool which allows individuals to track their "brain age" relative to their peers could prove an immensely valuable tool.

In addition to providing a general indication of health, there is speculation that this approach may also be helpful for creating "brain aging profiles" for various brain pathologies. For example, Cole, Leech, and Sharp (2015) found that traumatic brain injuries result in an accelerated rate of brain aging. Alzheimers and Dementia are well known for their accelerated rate of cognitive decline and cortical atrophy (McKhann et al. 2011), and so a brain aging model would be of clear benefit here as well.

1.1 Description of the Problem

Predicting brain age refers to the estimation of an individual's age using only physiological and/or anatomical information about their brain. In order to be generally applicable, we also require that any methods used to acquire this information be non-invasive. That is, they do not involve the introduction of medical equipment into the body. Magnetic Resonance Imaging (MRI) is one such technology that meets these requirements and will be the focus of this work. Its worth noting that other methods such as Electroencephalography (EEG) and Magnetic Resonance Spectroscopy (MRS) could be used to these means as well, however, their use is beyond the scope of this work.

MRI is a medical imaging technique that utilizes strong magnetic fields and gradients in order to measure various properties of brain tissues. There are many approaches, commonly referred to as modalities, each measuring different aspects of the brain. For example, T1 and T2 FLAIR images are commonly used to construct detailed anatomical images, whereas fMRI is used to measure changes in blood oxygen levels and is said to be reflective of brain activity. Reconstructed images are typically 3 or 4 dimensions and are on the order of 1 million voxels, or volumetric pixels, per 3D volume.

The large number of datapoints within a single image can present signficant complications when it comes to making predictions from MRI data. As a result, most approaches have involved feature extraction to reduce the data's high dimensionality. Both Valizadeh et al. (2017) and Aycheh et al. (2018) used measurements of anatomical regions (e.g cortical volume and thickness) derived from T1 MRI images to predict brain age. However, feature extraction can be a complicated process requiring in-depth domain knowledge and extensive quality control proceedures. Furthermore, potentially relevent information is likely to be lost during the feature extraction process which can have negative impacts on the predictive power of a model.

1.2 Proposed Solution

From handwritten digit recognition (Ciresan et al. 2011) to object recognition (Krizhevsky, Sutskever, and Hinton 2012), 2D convolutional neural networks (CNNs) have been successfully utilized to solve many complicated computer vision tasks in recent years. This success extends to medical imaging, where 3D CNNs have been used for the segmentation of brain tumors (Pereira et al. 2016) as well as the classification of Alzheimer's Disease (Payan and Montana 2015) from preprocessed MRI data. CNNs learn features directly from the training data. This effectively sidesteps the feature development process and problems mentioned above albeit by introducing the complication of training a deep learning model.

The aim of this work is to develop a neural network utilizing 3D convolutional layers for the purpose of predicting age from MRI data alone. An experimental framework will be developed to fascilitate the exploration of various network architectures and training parameters. Subsequently, a range of experiments will be conducted to determine how performance changes when certain parameters are adjusted. The results of these experiments will guide the model development. Finally, model performance will be assessed on a held-out set of data not previously used during model development.

2 Methods

2.1 About the Data

We will use Diffusion Weighted Imaging (DWI) scans from the UK Biobank Study. This long-term population study has collected physical and health data from over 500,000 participants, a subset of which also underwent an MRI acquisition (Sudlow et al. 2015).

DWI is an MRI modality that measures the diffusion of water from multiple directions. It is useful for determining the structural integrity of white-matter, and is highly affected by ageing processes (Sullivan and Pfefferbaum 2006). Images are 104 x 104 x 72 voxels where each voxel is 2mm x 2mm x 2mm.

2.2 Preprocessing

Fractional Anisotropy (FA), Mean Diffusivity (MD), Axial Diffusivity (AD), and Radial Diffusivity (RD) images were first derived by fitting the Diffusion Tensor model to the DWI data. Descriptions of these measures are available in Table 1.

Table 1: Description of DWI measures

Measure	Description
FA	How restricted the diffusion is
MD	Average diffusion in all directions
AD	Magnitude of diffusion along the main axis
RD	Magnitude of diffusion perpendicular to the main axis

Images were then transformed to a common MNI space template to ensure consistent alignment between subjects. This step is necessary to prevent the model from having to accomodate relative shifts and rotations of the brain between subjects.

Finally, each image is scaled to have a range between 0 and 1 by the following formula:

$$\forall v_i \in V : v_i' = \frac{v_i - \min(V)}{\max(V) - \min(V)}$$

Where V is the set of all non-zero voxels, and v_i is the i^{th} voxel. This ensures that each of the four input channels are similar in scale which is important for successful training of neural networks.

2.3 Subjects Demographics

The cohort has a mean age of 63.1397548 with a standard deviation of 7.5289271. It worth noting that a model which predicts the population mean would have an expected Mean-Squared Error (MSE) of 56.6828946.

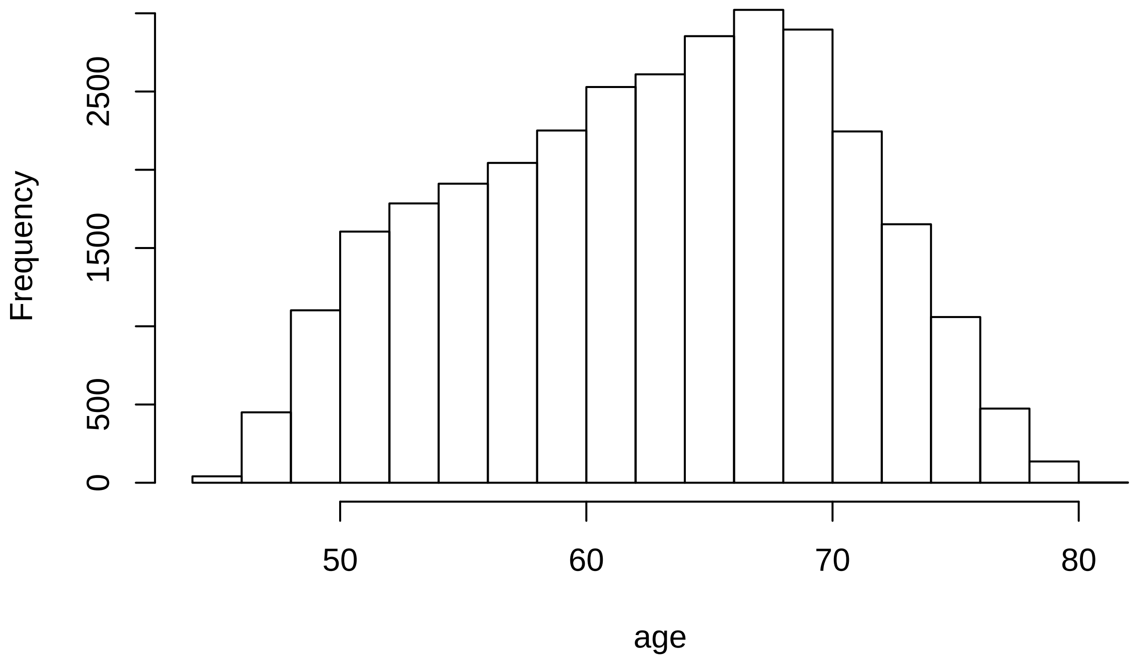


Figure 1: Histogram of Subject Ages

2.4 Train, Test, and Validation Sets

A total of 17915 subjects have been used for this project. This has been randomly partitioned into train, test, and validation sets with respective sizes of 13915, 2000, and 2000. The training set will be used to train the model, the test set will be used to asses model performance during model development, and the valiation set will be used to assess the final model performance.

2.5 Experimental Framework

The `train.py` script allows the user to specify a multitude of options relating to network architecture and training parameters. It was written in order to facilitate rigorous testing to inform model development. Listing 1 shows the help message for this script and summarizes the options available.

Listing 1: 'Help message for `train.py` script'

```
usage: train.py [-h] [--datapath [str]] [--scale-inputs] [--workers [int]]
                [--savepath [str]] [--save-freq [str]] [--load-model [str]]
                [--conv-layers [int]] [--kernel-size int [int ...]]
                [--dilation int [int ...]] [--padding int [int ...]]
                [--even-padding] [--stride int [int ...]]
                [--filters int [int ...]] [--weight-init [str]]
                [--conv-actv str [str ...]] [--fc-actv str [str ...]]
                [--pooling [str]] [--model-output [str]] [--lr [float]]
                [--decay [float]] [--reduce-on-plateau] [--batch-size [int]]
                [--epochs [int]] [--update-freq [int]] [--cuda] [--loss [str]]
                [--optim [str]] [--weight-decay [float]] [--beta1 [float]]
                [--beta2 [float]] [--rho [float]] [--momentum [float]]
                [--dampening [float]] [--amsgrad] [--nesterov]
                [--debug-size   ] [--silent] [--test-model [str]]
                [--encode-age]

optional arguments:
  -h, --help            show this help message and exit
  --datapath [str]      Path to data folder (default: data)
  --scale-inputs        Set flag to scale input images (default: False)
  --workers [int]       Number of workers in DataLoader (default: 8)
  --savepath [str]      Folder where model checkpoints should be saved -- if
                        None model will not be saved. If savepath is
                        specified, models will be saved in a new folder named
                        according to the date and time it is run. If mutliple
                        instances are being run in parallel, each instance
                        should have a different savepath to avoid overlap.
                        (default: None)
  --save-freq [str]     How often model checkpoints should be saved (in
                        epochs) (default: 1)
  --load-model [str]    Specify a saved model to load and train. Other
                        arguments relating to model paremeters (padding,
                        kernel-size, etc..) will be ignored. Training
                        parameters (learning rate, update frequency, etc ...)
                        may still be specified. (default: None)
  --conv-layers [int]   Number of Conv3d layers (default: 3)
  --kernel-size int [int ...]
                        Kernel size of each filter (default: [5])
  --dilation int [int ...]
                        Dilation factor for each filter (default: [1])
  --padding int [int ...]
                        Zero padding to be used in Conv3d layers (default:
                        [2])
  --even-padding        Calculate padding vectors to ensure even perfect
                        overlap with kernel applications. Layers with stride =
                        1 will have input dimensions preserved. The '--
                        padding' argument is ignored when this flag is set
                        (default: False)
  --stride int [int ...]
                        Stride between filter applications (default: [3])
  --filters int [int ...]
                        Filters to apply to each channel -- one entry per
                        layer (default: [4, 4, 4])
  --weight-init [str]   Weight initialization method [normal, uniform, xavier-
                        normal, xavier-uniform, kaiming-normal, kaiming-
                        uniform, leaky-kaiming-normal, leaky-kaiming-uniform]
                        (default: kaiming-uniform)
  --conv-actv str [str ...]
```

```
                        Activation functions to be used in convolutional
                        layers -- must be 1 or n_conv_layers [sigmoid,
                        softmax, tanh, relu, elu, leaky-relu, rrelu] (default:
                        ['relu'])
--fc-actv str [str ...]
                        Activation functions to be used in convolutional
                        layers -- must be 1 or n_conv_layers [sigmoid,
                        softmax, tanh, relu, elu, leaky-relu, rrelu] (default:
                        ['relu'])
--pooling [str]         Which pooling method to apply in between convolution
                        layers. If this argument is not specified, then no
                        pooling will be performed. ['max','avg'] (default:
                        None)
--model-output [str]    Specify what type of output the model should produce.
                        'value': model is trained to predict a single value
                        (e.g age). 'scaled-value': same as 'value', but scaled
                        down by a factor of 100. 'single-class': ages are
                        treated as individual classes to be predited.
                        'ordinal-class': a class should be predicted if it is
                        <= target age. 'gaussian': model is trained to predict
                        a range of outputs centered around the target age.
                        ['value','scaled-value','single-class','ordinal-
                        class','gaussian'] (default: scaled-value)
--lr [float]            Learning rate paramater (default: 0.001)
--decay [float]         Learning rate decay (multiplicative factor). Unless '
                        --reduce-on-plateau is set, this decay rate is applied
                        every epoch (default: 1.0)
--reduce-on-plateau     Learning rate will decay after performance on the
                        train set plateaus as opposed to every epoch (default:
                        False)
--batch-size [int]      Number of samples per batch (default: 32)
--epochs [int]          Number of epochs to train over (default: 20)
--update-freq [int]     How often (in epochs) to asses test set accuracy
                        (default: 1)
--cuda                  Set flag to use cuda device(s) (default: False)
--loss [str]            Specify a loss function. [L1, L2, SmoothL1, BCE,
                        Wasserstein] (default: L2)
--optim [str]           Specify optimizer to use. [adam, adamw, adamax,
                        adagrad, sgd] (default: adam)
--weight-decay [float]
                        weight decay parameter for relevant optimizers
                        (default: 0)
--beta1 [float]         beta1 parameter for relevant optimizers (default: 0.9)
--beta2 [float]         beta1 parameter for relevant optimizers (default:
                        0.999)
--rho [float]           rho parameter for relevant optimizers (default: 0.9)
--momentum [float]      momentum parameter for relevant optimizers (default:
                        0.0)
--dampening [float]     dampening parameter for relevant optimizers (default:
                        0.0)
--amsgrad               whether to use the amsgrad variant (default: False)
--nesterov              whether to use the nesterov variant (default: False)
--debug-size            Print out the expected architecture. 4 Integers should
                        be supplied to this argument [channels, dimx, dimy,
                        dimz]. Program execution will terminate afterwards
                        (default: None)
--test-model [str]      Instead of training the loaded model, it's performance
                        will be assessed on either the test or train set.
                        ['test','train'] (default: None)
```

3 Experiments & Results

3.1 Initial Model

The initial model consisted of 5 3D convolutional layers with a dilation of 1, stride of 1, no padding and max pooling between layers. Kernel sizes for each layer were 5, 4, 4, 2, and 2 respectively, and filters per channel at each layer are 2 so that there are 128 channels after the last convolutional layer. These channels are flattened and concatenated with 50% dropout being applied. Three fully connected layers then gradually reduce the number of nodes down to 1, which is used as the output of the model. Exponential linear units (elu) were used as the activation function for each layer.

For training, target ages were scaled by dividing by 100, and Mean Squared Error (MSE) loss was used alongside an Adam optimizer with default settings of 0.99 and 0.999 for β_1 and β_2 respectively. Weight initialization was performed by the `kaiming-uniform` method. Model performance is assessed on the training set at the end of each epoch.

See Listing 2 for a summary of the model. Overall there were 162,985 parameters.

Listing 2: 'Initial model summary'

```
datapath:              dwi_data/
scale_inputs:          True
workers:               6
savepath:              models/
save_freq:             1
load_model:            None
conv_layers:           5
kernel_size:           [5, 4, 4, 2, 2]
dilation:              [1]
padding:               [2]
even_padding:          True
stride:                [1]
filters:               [2, 2, 2, 2, 2]
weight_init:           kaiming-uniform
conv_actv:             ['elu']
fc_actv:               ['elu']
pooling:               max
lr:                    0.001
decay:                 1.0
reduce_on_plateau:     False
batch_size:            32
epochs:                10
update_freq:           1
cuda:                  True
debug_size:            None
silent:                False
test_model:            None
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
  ConstantPad3d-1       [-1, 4, 104, 104, 72]               0
         Conv3d-2       [-1, 8, 100, 100, 68]           1,008
      MaxPool3d-3         [-1, 8, 50, 50, 34]               0
  ConstantPad3d-4         [-1, 8, 50, 50, 34]               0
         Conv3d-5        [-1, 16, 47, 47, 31]           1,040
      MaxPool3d-6        [-1, 16, 24, 24, 16]               0
  ConstantPad3d-7        [-1, 16, 24, 24, 16]               0
         Conv3d-8        [-1, 32, 21, 21, 13]           2,080
      MaxPool3d-9         [-1, 32, 11, 11, 7]               0
 ConstantPad3d-10         [-1, 32, 11, 11, 7]               0
```

```
          Conv3d-11         [-1, 64, 10, 10, 6]            576
       MaxPool3d-12          [-1, 64, 5, 5, 3]              0
   ConstantPad3d-13          [-1, 64, 5, 5, 3]              0
          Conv3d-14         [-1, 128, 4, 4, 2]          1,152
       MaxPool3d-15         [-1, 128, 2, 2, 1]              0
         Dropout-16                  [-1, 512]              0
          Linear-17                  [-1, 256]        131,328
          Linear-18                  [-1, 100]         25,700
          Linear-19                    [-1, 1]            101
================================================================
Total params: 162,985
Trainable params: 162,985
Non-trainable params: 0
```

3.2 Learning Rate and Batch Size Grid Search

The ideal learning rate is highly dependent upon batch size. This is because larger batch sizes result in more stable training, and allow for larger learning rates to be applied. However, larger batch sizes may result in slower learning and also require more memory in order to compute backwards gradients. To determine the ideal combination of paramters, a grid search was performed by testing every pairwise combination of `batch_size = {16, 32, 64, 96, 128}` and `learing_rate = {0.01, 0.001, 0.0001, 0.00001}`. Each model was trained for 10 epochs, after which MSE loss on the test set was used for comparison. The results are sumarized by the heatmap shown in Figure 2.

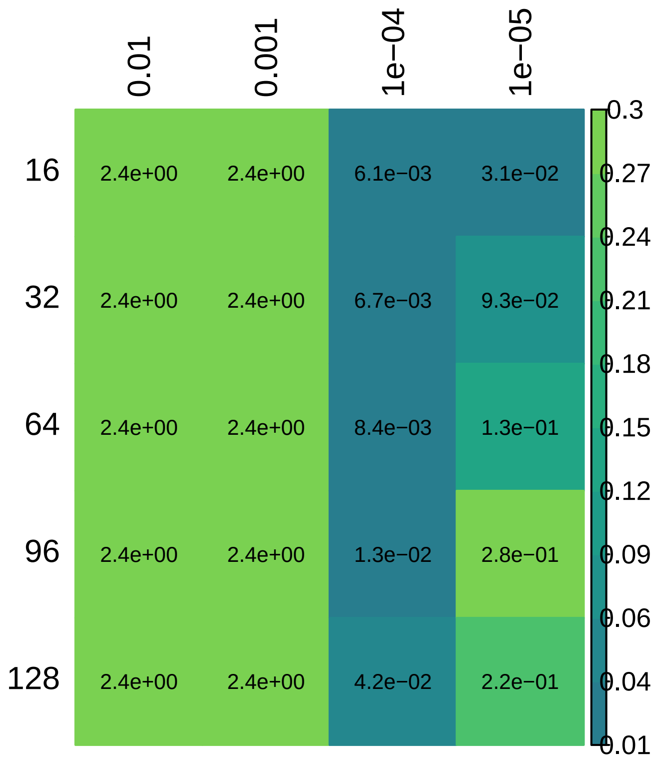


Figure 2: Heatmap of MSE after 10 epochs with batch size on the y-axis and learning rate on the x-axis

A learning rate of 1e-04 provided the best reduction in test set MSE after 10 epochs with larger batch sizes showing worse performance. A batch size of 64 was selected for further models despite the slightly worse performance at this stage. The same performance should be able to be obtained with extra training, and the model will be less prone to overfitting.

At this point, the selected model has an MSE loss on the test set of 8.4e-03 whereas the best observed model had an MSE of 6.1e-03. Recall that the expected MSE loss from predicting the cohort mean was 56 years. After accounting for scaling of the target value, this is equivalent to 5.6e-03, so none of the models perform as well as just predicting the mean.

3.3 Ideal Activation Functions Grid Search

Next, the effect of different activation functions on model performance was explored. The activation functions considered in this experiment were:

- Rectified Linear Units (relu)

  $$\text{relu}(x) = \max(0, x)$$

- Exponential Linear Units (elu)

  $$\text{elu}(x) = \max(0, x) + \min(0, exp(x) - 1)$$

- Random rectified linear units (rrelu)

  $$\text{rrelu}(x) = \begin{cases} x & \text{if } x \geq 0 \\ ux & \text{otherwise} \end{cases} \qquad \text{where } u \sim U(0, 1)$$

- Leaky-relu

  $$\text{leaky-relu}(x) = \max(0, x) + \min(0, -0.1 * x)$$

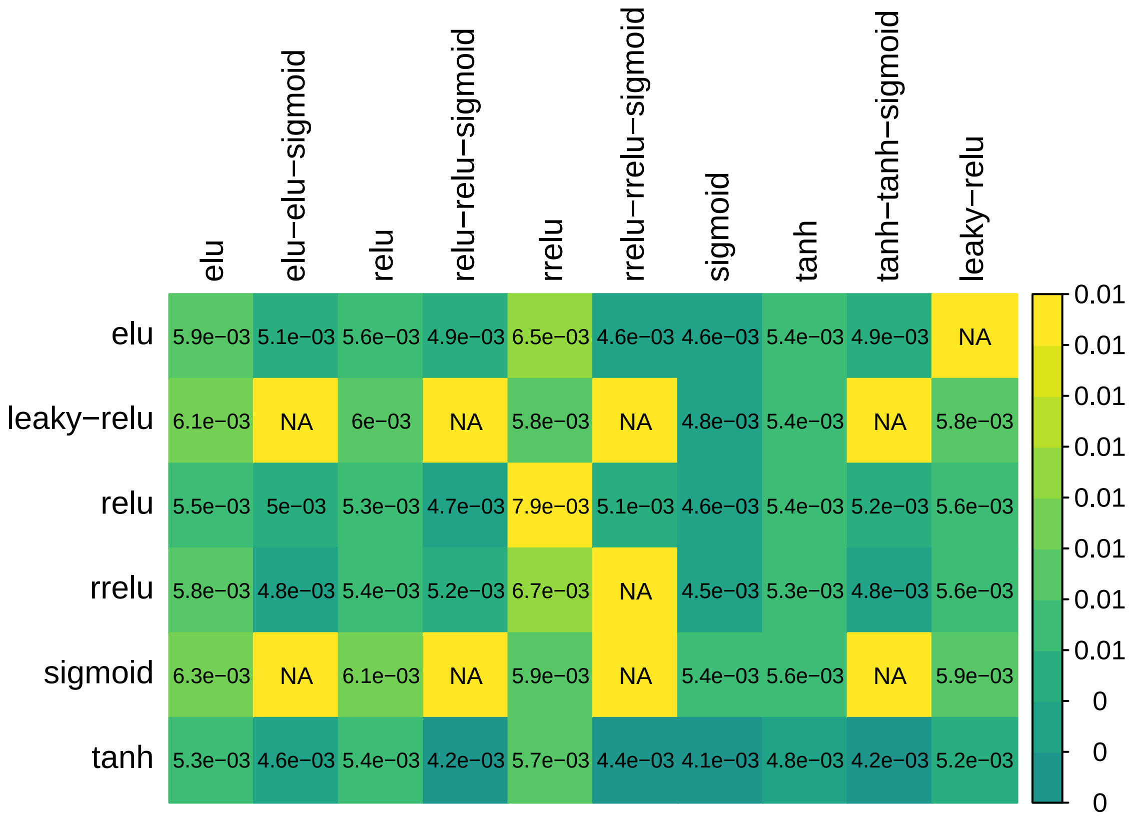
- Sigmoid

  $$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

- Tanh

  $$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

Convolutional and fully connected layers often require different activation functions, and so a grid search was performed comparing various pairwise combinations of the above activation functions. This included using multiple activation functions in the fully connected layers (e.g `relu-relu-sigmoid`). This time, models were trained for 20 epochs, and the best MSE value on the test set was used for comparison. The results are summarized in Figure 3.



	elu	elu−elu−sigmoid	relu	relu−relu−sigmoid	rrelu	rrelu−rrelu−sigmoid	sigmoid	tanh	tanh−tanh−sigmoid	leaky−relu
elu	5.9e−03	5.1e−03	5.6e−03	4.9e−03	6.5e−03	4.6e−03	4.6e−03	5.4e−03	4.9e−03	NA
leaky−relu	6.1e−03	NA	6e−03	NA	5.8e−03	NA	4.8e−03	5.4e−03	NA	5.8e−03
relu	5.5e−03	5e−03	5.3e−03	4.7e−03	7.9e−03	5.1e−03	4.6e−03	5.4e−03	5.2e−03	5.6e−03
rrelu	5.8e−03	4.8e−03	5.4e−03	5.2e−03	6.7e−03	NA	4.5e−03	5.3e−03	4.8e−03	5.6e−03
sigmoid	6.3e−03	NA	6.1e−03	NA	5.9e−03	NA	5.4e−03	5.6e−03	NA	5.9e−03
tanh	5.3e−03	4.6e−03	5.4e−03	4.2e−03	5.7e−03	4.4e−03	4.1e−03	4.8e−03	4.2e−03	5.2e−03

Figure 3: Heatmap of MSE after 20 epochs with Conv layer activations on the y-axis and FC layer activations on the x-axis. Entries with NA were either not run, or crashed due to memory consumption

The best performing model used tanh activations on the convolution layers and sigmoid activations on the fully connected layers. It obtained an MSE of 4.1e-03 after 20 epochs, which is an improvement over predicting the cohort mean (expected MSE of 5.6e-03).

3.4 Experimenting with Learning Rate Decay

The next experiment investigated the effect of learning rate decay on model performance. Starting with an initial learning rate of 1e-04, the learning rate was decreased by a multiplicitive factor. That is

$$\eta_{i+1} = \lambda \eta_i, 0 < \lambda \leq 1$$

where η_i is the learning rate at the i^{th} epoch and λ is the decay factor. Models with $\lambda = 0.90, 0.905, 0.91, ..., 0.995, 1.0$ were trained and their best MSE on the test set across 40 epochs was used for comparison with the other models. Figure 4 – Left summarizes the results.

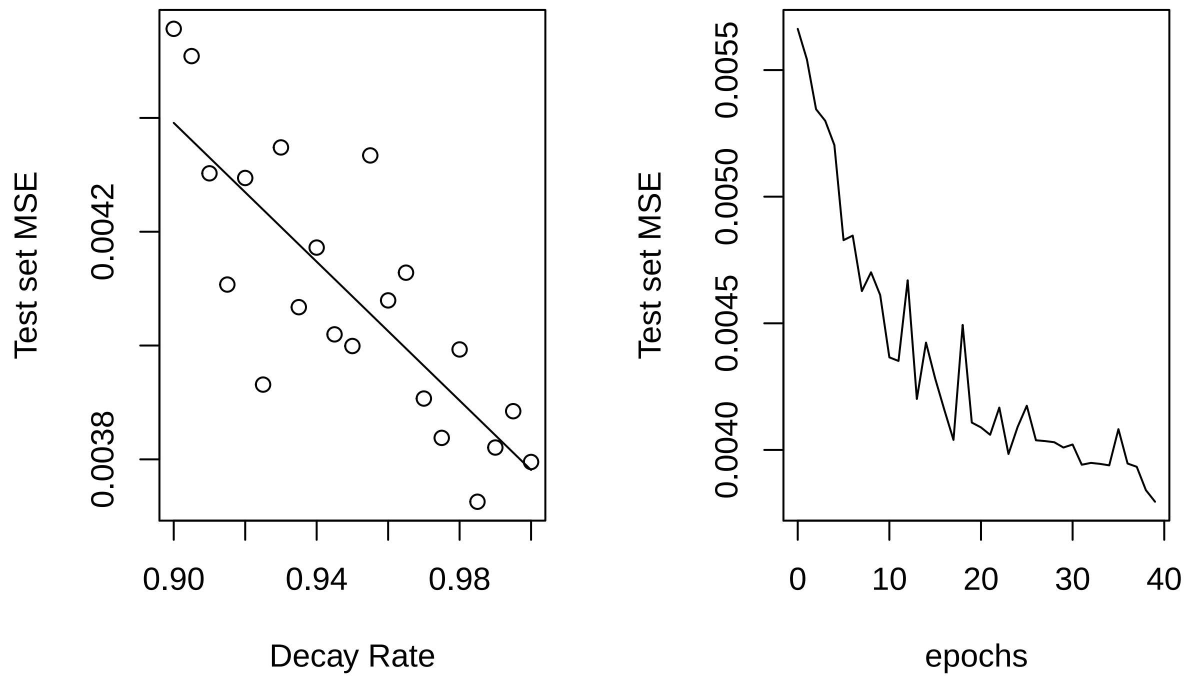


Figure 4: (Left) Learning rate decay against Test set MSE (Right) Number of epochs against Test set MSE without learning rate decay

Model performance degraded as learning rate decay became more aggressive. It was expected that small to moderate amounts of decay would improve performance; however, this was not observed. One reason for this could be that the learning curve has not flattened enough for a reduced learning rate to be of any benefit (See Figure 4 – Right). For this reason, a decay rate of 0.995 was selected as it did not appear to harm performance; however, it may still provide benefits during longer periods of training.

3.5 Experimenting with Different Numbers of Filters

Crudely speaking, each filter corresponds to a different feature the network attempts to learn. Understandably then, increasing the number of filters can help to improve predictive power. However, this also increases the number of trainable parameters in the network resulting in a higher propensity for overfitting as well as increased computational complexity.

At each convolutional layer in the network, a certain number of filters is learned for each input channel. Therefore, if there are n input channels and f filters per channel, then nf channels will be output to the next layer. Up until this point, models have been trained with 2 filters per channel per layer, so the number of channels in the convolutional layers follows the sequence $\{4, 8, 16, 32, 64, 128\}$. The purpose of this next experiment is to attempt to determine the ideal number of filters at each layer. Due to the combinatorial nature of possibilities here, it is not practical to perform an exhaustive search through the problem space. Table 2 summarizes the combinations that were tried. As before, each model was trained for 40 epochs and the best MSE score was used to compare with other models. The results are summarized in Table 3.

Table 2: Various filter sequences explored by experiment

1-1-1-1-1	1-1-1-1-2	1-1-1-2-2	1-1-2-2-2	1-2-2-2-2
	2-1-1-1-1	2-2-1-1-1	2-2-2-1-1	2-2-2-2-1
2-2-2-2-2	2-2-2-2-3	2-2-2-3-3	2-2-3-3-3	2-3-3-3-3
	3-2-2-2-2	3-3-2-2-2	3-3-3-2-2	3-3-3-3-2
	2-2-2-2-4	2-2-2-4-4	2-2-4-4-4	2-4-4-4-4
	4-2-2-2-2	4-4-2-2-2	4-4-4-2-2	4-4-4-4-2
3-3-3-3-3				

Table 3: Test set MSE results for filter sequence tests in order of performance

Filters	MSE
3-3-3-3-3	0.0032968
4-4-4-2-2	0.0033321
3-3-3-3-2	0.0033436
2-4-4-4-4	0.0033552
3-3-3-2-2	0.0033719
2-2-4-4-4	0.0034023
2-2-3-3-3	0.0034474
4-4-2-2-2	0.0034615
2-2-2-4-4	0.0034832
2-3-3-3-3	0.0035150
2-2-2-2-3	0.0035693
3-3-2-2-2	0.0036006
2-2-2-3-3	0.0036273
4-2-2-2-2	0.0036744
2-2-2-2-2	0.0037212
3-2-2-2-2	0.0037980
1-2-2-2-2	0.0038299
2-2-2-2-4	0.0038584
2-2-2-2-1	0.0040126
2-2-2-1-1	0.0041815
2-2-1-1-1	0.0042759
1-1-2-2-2	0.0046774
1-1-1-2-2	0.0049667
1-1-1-1-1	0.0049741
2-1-1-1-1	0.0052045
1-1-1-1-2	0.0053083
4-4-4-4-2	NA

The best performing model had 3 filters per channel at each layer and obtained an MSE of 3.297e-03 within the first 40 epochs. In general, models with higher numbers of total filters, and therefore parameters, performed better. Furthermore, models with a higher number of filters in the beginning layers outperformed those with the same number of filters, but introduced later on. That is, 3-3-3-2 (rank 3) is preferable to 2-3-3-3-3 (rank 10). Unfortunately, filter numbers could not be increased beyond the amounts tested due to limitations on hardware.

3.6 Experimenting with Weight Decay (L2 Regularization)

Weight decay is a technique that involves adding an extra penalization term to the base loss function to encourage the network to keep the weights small. In the case of L2 Regularization with MSE loss, the new loss function becomes:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \frac{1}{2}c||\boldsymbol{w}||^2$$

where c is a constant term controlling how heavily large weights should be penalized.

Larger values of c can help the model generalize better to unseen data and prevent the model from relying too heavily on any one feature or input; however, if set too high, weight decay can prevent the model from learning. Several models were trained to test the effect of setting c to 0 (no decay), 1e-05, 5e-05, 1e-04, 5e-04, and 1e-03. Models were trained for 60 epochs, and the results are summarized in Figure 5.

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 1 x value <= 0 omitted
## from logarithmic plot
```

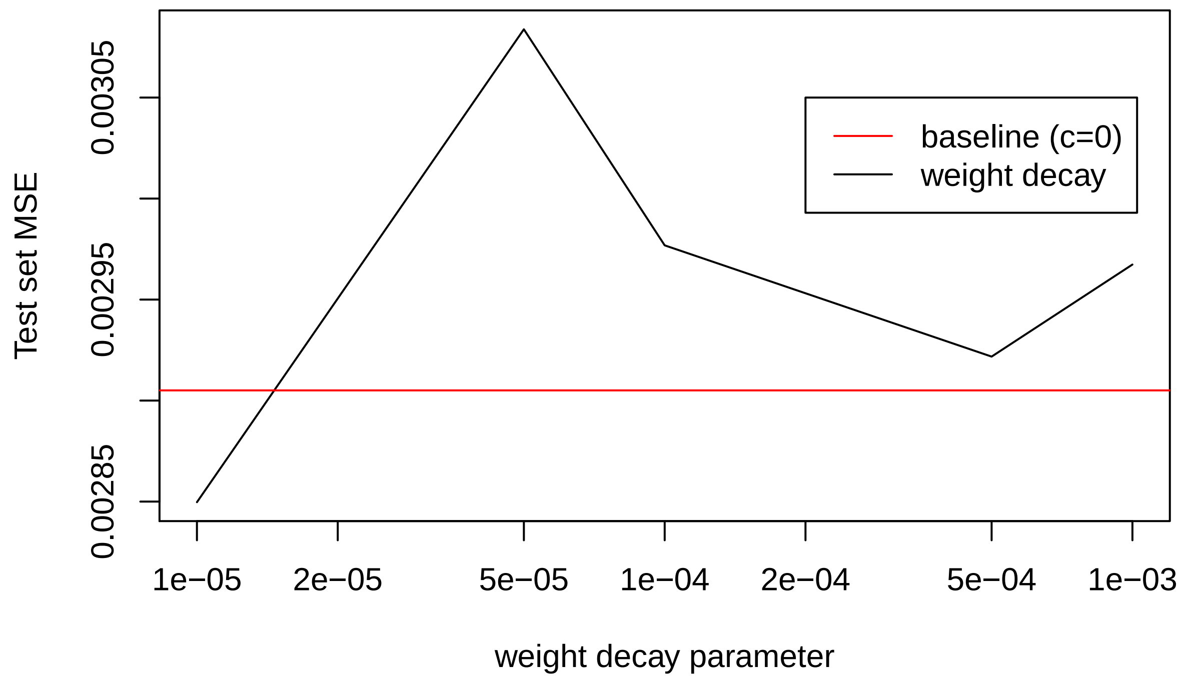


Figure 5: The effect of weight decay on Test set MSE. The red line shows the baseline model without any decay, and the black line shows the results of varying weight decay

The effect of weight decay on model performance is unclear, and no obvious trend was observed in the collected data. Only one model with weight decay outperformed the model without weight decay, however, it is unclear whether this is a significant effect.

This "top-performing" model had c = 1e-05 and obtained an MSE of 2.85e-03 on the Test set. This is in comparison to the baseline model's performance of 2.91e-03. Although the effect of weight decay is unclear,

c = 1e-05 will be used for the final model as it did technically outperform the baseline model, and the utility of weight decay is well documented in existing literature.

3.7 Final Model

The final model to be used will be the top performing model from section 3.6. Listing 3 summarizes the parameters used to train the model as well as the model archicture.

Listing 3: 'Final Model Specification'

```
scale_inputs:        True
conv_layers:         5
kernel_size:         [5, 4, 4, 2, 2]
dilation:            [1]
padding:             [2]
even_padding:        True
stride:              [1]
filters:             [3, 3, 3, 3, 3]
weight_init:         kaiming-uniform
conv_actv:           ['tanh']
fc_actv:             ['sigmoid']
pooling:             max
model_output:        scaled-value
lr:                  0.0001
decay:               0.995
batch_size:          64
epochs:              60
cuda:                True
loss:                L2
optim:               adam
weight_decay:        1e-05
beta1:               0.9
beta2:               0.999
amsgrad:             False

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
  ConstantPad3d-1       [-1, 4, 104, 104, 72]               0
          Conv3d-2     [-1, 12, 100, 100, 68]           1,512
       MaxPool3d-3       [-1, 12, 50, 50, 34]               0
   ConstantPad3d-4       [-1, 12, 50, 50, 34]               0
          Conv3d-5       [-1, 36, 47, 47, 31]           2,340
       MaxPool3d-6       [-1, 36, 24, 24, 16]               0
   ConstantPad3d-7       [-1, 36, 24, 24, 16]               0
          Conv3d-8      [-1, 108, 21, 21, 13]           7,020
       MaxPool3d-9       [-1, 108, 11, 11, 7]               0
  ConstantPad3d-10       [-1, 108, 11, 11, 7]               0
         Conv3d-11       [-1, 324, 10, 10, 6]           2,916
      MaxPool3d-12         [-1, 324, 5, 5, 3]               0
  ConstantPad3d-13         [-1, 324, 5, 5, 3]               0
         Conv3d-14         [-1, 972, 4, 4, 2]           8,748
      MaxPool3d-15         [-1, 972, 2, 2, 1]               0
        Dropout-16                 [-1, 3888]               0
         Linear-17                 [-1, 1944]       7,560,216
         Linear-18                  [-1, 100]         194,500
         Linear-19                    [-1, 1]             101
================================================================
Total params: 7,777,353
```

```
Trainable params: 7,777,353
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 11.88
Forward/backward pass size (MB): 133.74
Params size (MB): 29.67
Estimated Total Size (MB): 175.29
----------------------------------------------------------------
```

This model was trained for 60 epochs and MSE on the Test set was calculated at the end of each epoch. The lowest MSE on the Test set occurred after 56 epochs acheiving an MSE of 2.85e-03. Figure 6 shows the training curve.

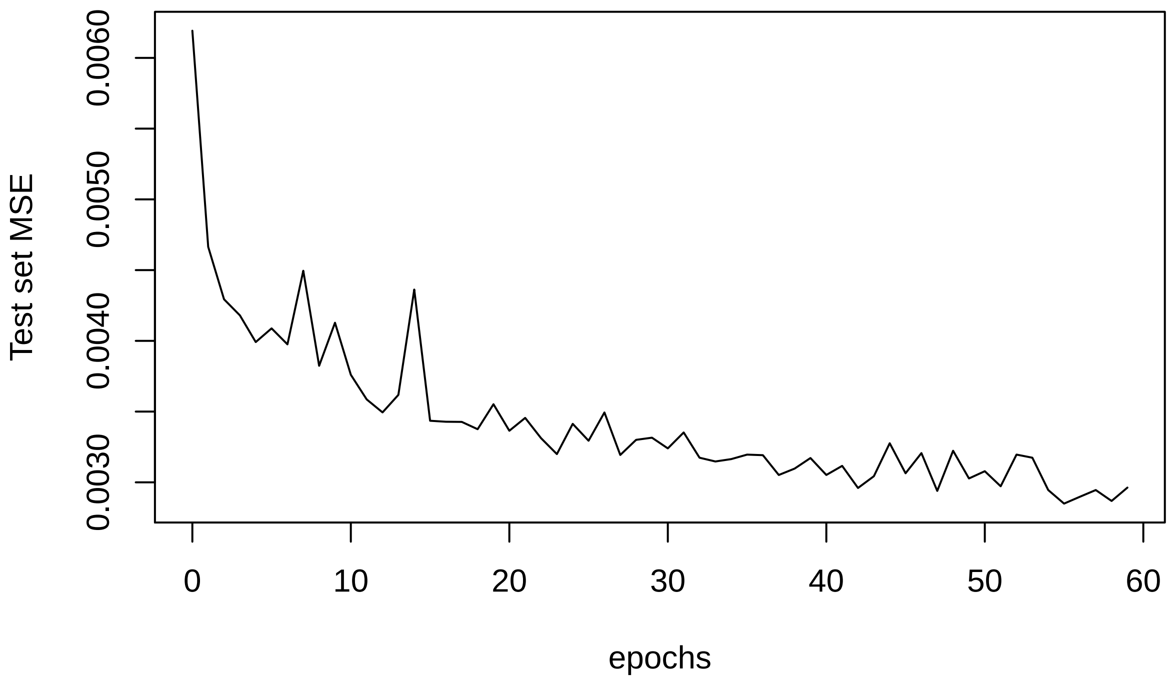


Figure 6: Training Epochs against MSE on the Test for the final model

As a final measure of performance, the model was tested on the validation set which hasn't been used in training up to this point. Figure 7 shows a scatterplot of predicted age versus actual age for this test.

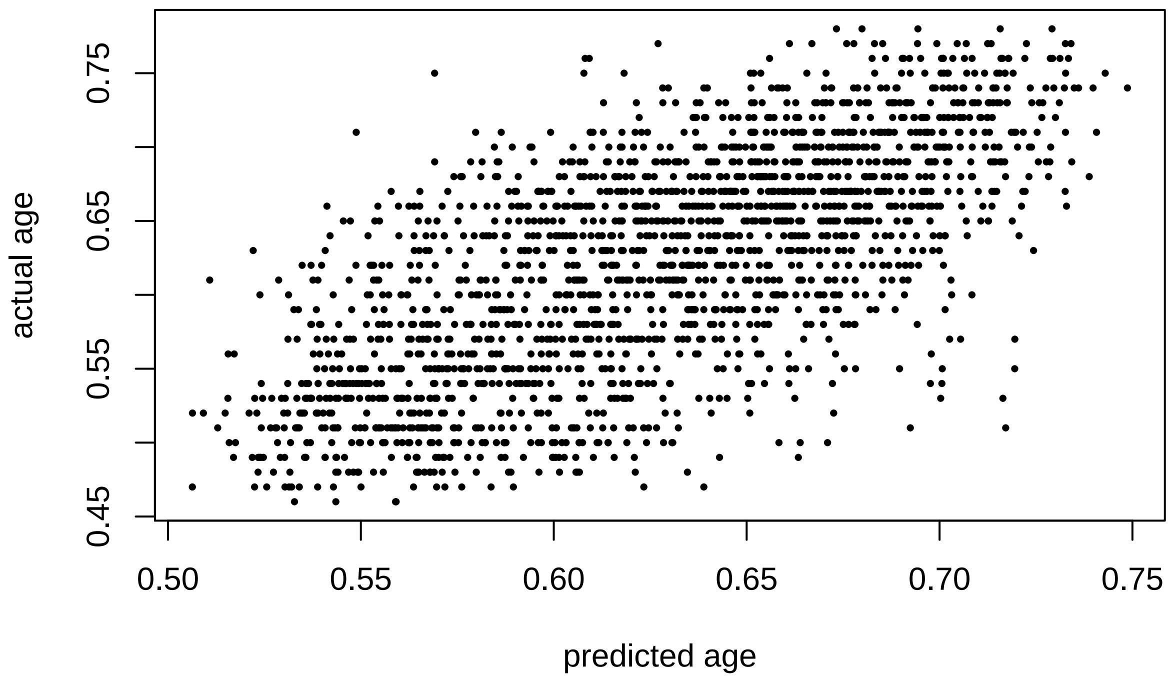


Figure 7: Scatterplot of predicted age against actual age

When assessed against the validation set, this model achieved a mean squared error of 0.0030884, a mean absolute error of 0.0444153 (or 4.4415315 years), and an R^2 of 0.4655911.

4 Conclusions

4.1 Comparison with "Predicting the Mean"

Simply predicting the mean would result in a MSE of

4.2 Comparison with State-of-the-Art

Several other studies have also created brain age prediction models:

- Aycheh et al. (2018) used a gaussian process regression model and obtained a mean absolute error of 4.05 years
- Wang and Pham (2011) used hidden markov models to predict age with "an average normalized age-gap error of two to three years"
- Li, Satterthwaite, and Fan (2018) used CNNs trained on fMRI connectivity data and obtained an absolute mean error of 2.15 years
- Cole, Leech, and Sharp (2015) obtained an R^2 of 0.85 using a gaussian process regression when trained on healthy subjects
- Valizadeh et al. (2017) achieved an R^2 of 0.84 using neural networks and support vector machines

Although this model was unable to match the performance of any of the above studies, it is worth noting that they each required extensive processing pipelines in order to extract relevent features. The processing for this model was very minimalistic only requiring standard DTI maps and alignment to a common template.

4.3 Limitations and Further Work

In the results of 3.5, we observed that a higher number of filters introduced early on was preferable to introducing the same number of filters but more slowly. Further work may investigate whether this phenomenon holds in more extreme cases (e.g 3-3-3-3-3 vs 243-1-1-1-1).

Bibliography

Aycheh, Habtamu M., Joon-Kyung Seong, Jeong-Hyeon Shin, Duk L. Na, Byungkon Kang, Sang W. Seo, and Kyung-Ah Sohn. 2018. "Biological Brain Age Prediction Using Cortical Thickness Data: A Large Scale Cohort Study." *Frontiers in Aging Neuroscience* 10. https://doi.org/10.3389/fnagi.2018.00252.

Ciresan, D. C., U. Meier, L. M. Gambardella, and J. Schmidhuber. 2011. "Convolutional Neural Network Committees for Handwritten Character Classification." In *2011 International Conference on Document Analysis and Recognition*, 1135–9. https://doi.org/10.1109/ICDAR.2011.229.

Cole, James H., Robert Leech, and David J. Sharp. 2015. "Prediction of Brain Age Suggests Accelerated Atrophy After Traumatic Brain Injury." *Annals of Neurology* 77 (4): 571–81. https://doi.org/10.1002/ana.24367.

Kalaria, Raj N, Gladys E Maestre, Raul Arizaga, Robert P Friedland, Doug Galasko, Kathleen Hall, José A Luchsinger, et al. 2008. "Alzheimer's Disease and Vascular Dementia in Developing Countries: Prevalence, Management, and Risk Factors." *The Lancet Neurology* 7 (9): 812–26. https://doi.org/10.1016/S1474-4422(08)70169-8.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." In *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, 1097–1105. Curran Associates, Inc. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Li, H., T. D. Satterthwaite, and Y. Fan. 2018. "Brain Age Prediction Based on Resting-State Functional Connectivity Patterns Using Convolutional Neural Networks." In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, 101–4. https://doi.org/10.1109/ISBI.2018.8363532.

McKhann, Guy M., David S. Knopman, Howard Chertkow, Bradley T. Hyman, Clifford R. Jack, Claudia H. Kawas, William E. Klunk, et al. 2011. "The Diagnosis of Dementia Due to Alzheimer's Disease: Recommendations from the National Institute on Aging-Alzheimer's Association Workgroups on Diagnostic Guidelines for Alzheimer's Disease." *Alzheimer's & Dementia* 7 (3): 263–69. https://doi.org/10.1016/j.jalz.2011.03.005.

Payan, Adrien, and Giovanni Montana. 2015. "Predicting Alzheimer's Disease: A Neuroimaging Study with 3D Convolutional Neural Networks." *arXiv:1502.02506 [Cs, Stat]*, February. http://arxiv.org/abs/1502.02506.

Pereira, S., A. Pinto, V. Alves, and C. A. Silva. 2016. "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images." *IEEE Transactions on Medical Imaging* 35 (5): 1240–51. https://doi.org/10.1109/TMI.2016.2538465.

Sudlow, Cathie, John Gallacher, Naomi Allen, Valerie Beral, Paul Burton, John Danesh, Paul Downey, et al. 2015. "UK Biobank: An Open Access Resource for Identifying the Causes of a Wide Range of Complex Diseases of Middle and Old Age." *PLOS Medicine* 12 (3): e1001779. https://doi.org/10.1371/journal.pmed.1001779.

Sullivan, Edith V., and Adolf Pfefferbaum. 2006. "Diffusion Tensor Imaging and Aging." *Neuroscience & Biobehavioral Reviews*, Methodological and Conceptual Advances in the Study of Brain-Behavior Dynamics: A Multivariate Lifespan Perspective, 30 (6): 749–61. https://doi.org/10.1016/j.neubiorev.2006.06.002.

Valizadeh, S. A., J. Hänggi, S. Mérillat, and L. Jäncke. 2017. "Age Prediction on the Basis of Brain Anatomical Measures." *Human Brain Mapping* 38 (2): 997–1008. https://doi.org/10.1002/hbm.23434.

Wang, Bing, and Tuan D. Pham. 2011. "MRI-Based Age Prediction Using Hidden Markov Models." *Journal of Neuroscience Methods* 199 (1): 140–45. https://doi.org/10.1016/j.jneumeth.2011.04.022.