Undergraduate Final Year Project

Final Report

~ April 2018 ~

Prediction of Individual Brain Ages
from MRI Data Sets
Using Deep Learning

Lv Ruyi
3035142560

Supervisors:
Prof. Ed X. Wu
Dr. Edward S. Hui

Table of Contents

Acknowledgments

I would like to thank Professor Ed X. Wu for his expert advice and guidance throughout this project. I would also like to thank Yilong Liu and Karim El Hallaoui for their support.

I would also like to express my gratitude to International Neuroimaging Data-sharing Initiative for providing the MRI images. This project would be impossible without them.

Abstract

Index terms: deep learning; brain age estimation; MRI.

Deep learning can accurately predict healthy individuals' chronological age from T1-weighted MRI brain images. By feeding novel data into the model, the resulting bio-marker, termed brain age, has the potential to investigate brain maturation and degeneration, as well as detect brain diseases in early phases. In this project, in order to evaluate the robustness of the brain age estimation system, four Convolutional Neural Network models were developed using different datasets. The training efficiency of the model is noticeable compared to previous attempts.

Images were selected from an archive containing 2072 samples to form four datasets. By training separately on the datasets, the model's performance with or without rotated images, with or without non-brain tissue interference, with or without dropout during testing, and with different data quality was compared. Input data were raw MRI images from individuals without known brain diseases with the mean image of the training data subtracted.

With the best performance model, the correlation between brain age and chronological age is evaluated as r=0.90, RMSE=9.66 years. The model is capable of dealing with images with different orientations but performs poorly when non-brain tissues are involved. Removing dropout during testing phase shrinks the estimated chronological age, with strong correlation intact, suggesting the estimated value is a sum of several indexes. The image quality of 65×65×55 is enough for the sound performance of the model.

By integrating with deep learning techniques such as feature visualization and inversion, brain age has the potential to deepen our understanding of brain development processes. Furthermore, data from individuals with known brain diseases can turn the biomarker into a valid tool for clinical evaluation.

1. Introduction

1.1. Need for Accurate Brain Age Estimation

1.1.1 Detection of Diseases

The human brain experiences significant changes in an individual's lifespan. Although these changes are not directly pathological, the cognitive decline associated with these changes is a good indicator of early-stage neurodegenerative diseases. However, the outset ages for such diseases have shown great variability between different brains. Therefore, a highly individualized brain age estimation algorithm is crucial for accurate early-stage detection of age-associated diseases, such as Alzheimer's disease (AD) or Parkinson's disease. Additionally, the diseases that cognitive decline is associated with are not limited to neurodegenerative ones. A brain age prediction model with precision has great clinical significance.

Accelerated brain aging has been found to be highly associated with disease severity and prospective worsening of Alzheimer's disease. The signs of cognitive decline usually appear years before the diagnosis of Alzheimer's disease. Specifically, the degree of global cognitive decline of the subjects was found to be 15 times that of normal five to six years before the actual diagnosis of AD [1]. Furthermore, the severity of the disease is usually correlated with the atrophy rates in certain brain regions of the subjects [2]. If a successfully developed brain age estimation model can take these factors into account, the risk of Alzheimer's development can be earlier detected and thus better coped with.

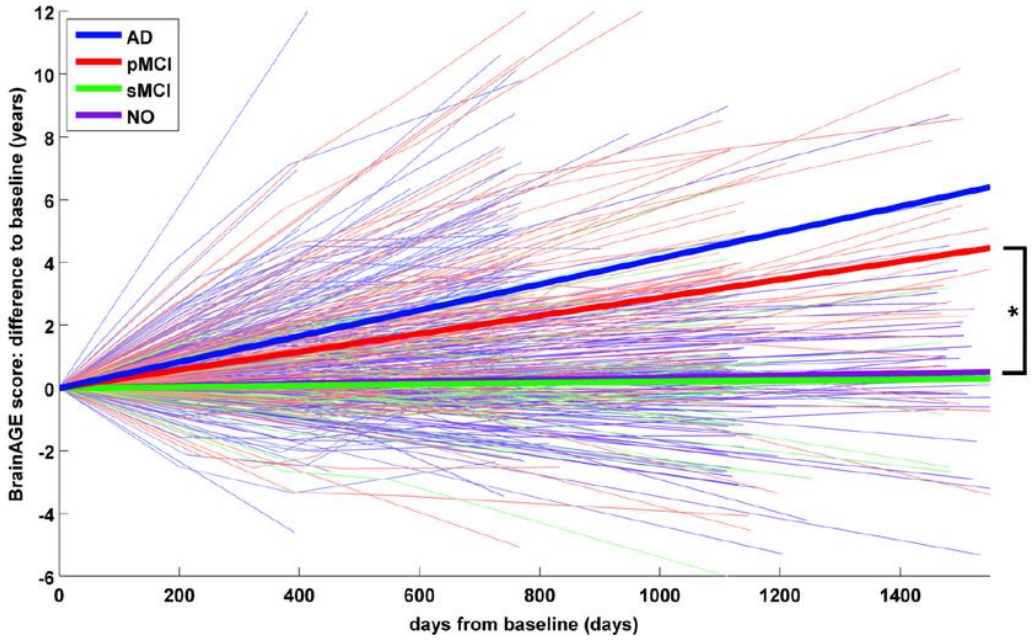

Figure 1.1 Longitudinal brain changes for AD-diagnosed individuals and control groups (retrieved from [1])

Cognitive decline has also been found to be highly associated with HIV infection. Abnormal brain aging was recognized in HIV-positive individuals even with adequate viral suppression [3]. The brains of individuals with type two diabetes, especially those with longer diabetes duration, have been found to display accentuated brain aging patterns [4]. Apart from these, cognitive decline pattern, estimated by brain-predicted age, is also found to be different in individuals with schizophrenia, epilepsy, Down's syndrome, and after traumatic brain injury [5].

1.1.2 Neural Degeneration Evaluation

Apart from disease detection, brain-predicted age may act as a pivot point for advancing understanding in brain development and quality of life. Previously developed brain-age prediction models have shown great robustness in classifying underage brains and adult brains. However, the understanding of their interim processes of such models needs to be further demonstrated by inversion assessment. By doing this, both the aging process of maturation and degeneration can be delved into. In many of the previous models, the underage samples were intentionally excluded in order to achieve a better performance in the estimation of adult

samples. This separation, however, renders an inability to compare wide-spread data in a single model, which may be necessary for a comprehensive understanding of brain aging process. The comeback of interest in convolutional neural networks for learning image data representatives, deep learning, offers a comprehensive method for data analysis and feature extraction. A large amount of features available in a model theoretically enables it to master the whole aging process during brain development.

An objective of health-improving and aging prevention is the improvement in the quality of life. The brain-age prediction model can thus be integrated with the existing quality of life models to evaluate these health-improving process. For example, in a study that targeted meditators with age over fifty, meditators' brain-ages were predicted to be 1 month and 22 days younger than their chronological age [6]. Experiments like this, especially those that integrate factors more than one, require an accurate age-prediction model whose reliability and efficiency are well-demonstrated.

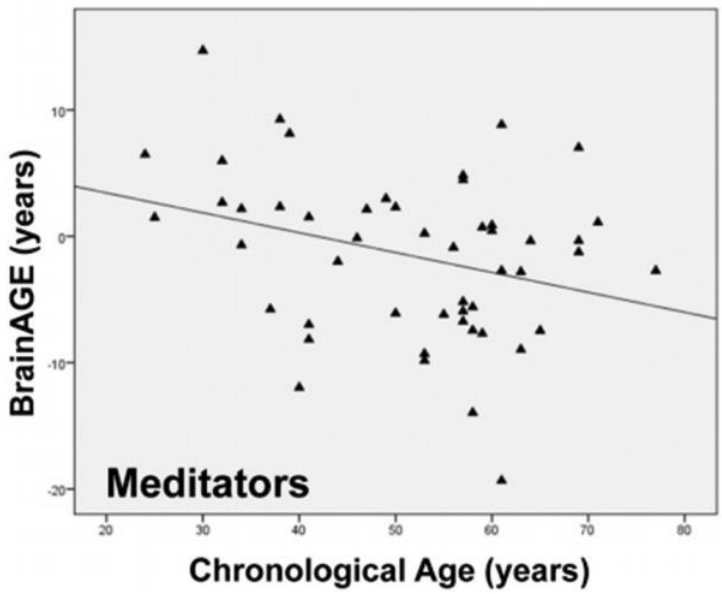

Figure 1.2 Estimated brain age and chronological age of meditators group

1.1.3 Bio-marker Reliability and Time Efficiency

If a biomarker is to be validated, no matter in research or clinical settings, the retest reliability of the marker has to be established. Part of the problem of evaluating brain-age estimation models is that related to the sample size and the selection of the data. Therefore, comparing the same index used in different models may not have statistical significance. Furthermore, one feature of the bio-images analysis is that the raw data is usually bulky and hard to generate. As a result, the collected brain images are usually from different repositories. If this is the case, between-repository and between-scanner reliability should also be evaluated. This is important for the whole purpose of the model is to evaluate brains that have not been used to train. The high between-repository reliability ensures that the model is still appropriate for images with similar but not exactly the same settings.

The time efficiency problem is usually encountered in the clinical setting. The decisions that clinicians have to make happen in less than a few minutes. The image pre-processing that the previous brain-age estimation models require, however, can take up to a few days. These pre-processing processes include but not limited to the removal of non-brain data, interpolation, clipping, smoothing, as well as gray matter and white matter extraction. In order to meet the clinical needs, an algorithm that accepts images in nearly raw format is needed. This can be achieved by learning data representations, specifically deep learning. Furthermore, recent studies have shown that the pre-processing methods that the traditional models use may not meet all the assumptions required and thus become a source of error [7]. This is especially problematic in deep learning, as the specific function of each layer in deep learning is usually unknown and the deviation of input data is thus harder to be detected.

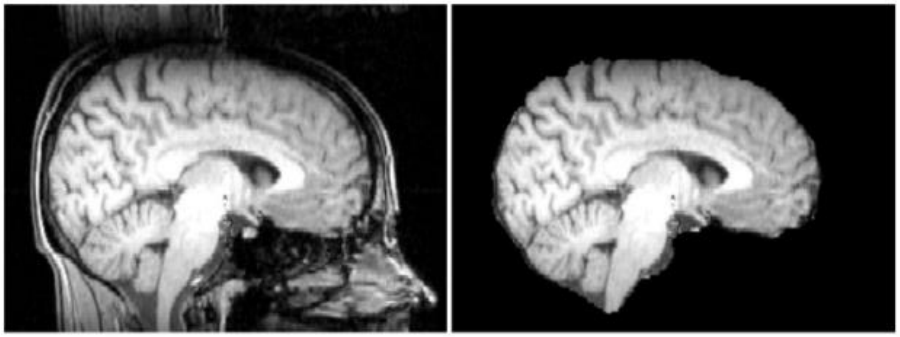

Figure 1.3 Removal of Non-brain Tissue

1.4 Previous Attempts

The first attempt towards such brain-based age prediction utilized the method of machine learning, specifically kernel regression method [8]. The method is based on the relevance vector regression, RVR, introduced by Tipping. The model demonstrated the use of a probabilistic Bayesian learning framework to maximize the performance of support vector machines (SVM). Accurate estimation can be achieved by greatly fewer basis functions [9]. The study focused on the brain maturation process during childhood and adolescence only. Using MRI images obtained in 1.5T field strengths, the maturation curve predicted elucidated 87% of the sample variance. The study performed affine registration with a large smoothing kernel to preprocess the data. The human sample used in the study was 394 and came from six scanner conditions. The age prediction, called brainAGE in the study, has an average correlation coefficient of 0.93 with the actual chronological age and an MAE of 1.2 years. More detailed evaluations for each scanner condition is shown below. It can be seen that although the raw correlation coefficient of each scanner condition was shown, the between-scanner reliability was not further demonstrated. Furthermore, the small sample size and the limited age range utilized rendered the model rather narrow in application.

	Scanner site (as test sample)					
1	2	3	4	5	6	
Scanner	GE Genesis Signa	GE Genesis Signa	GE Genesis Signa	GE Genesis Signa	Siemens Sonata	Siemens Magneton Vision
Number of subjects	53	76	71	75	48	70
Age range (years)	4.8-17.8	4.8-18.5	5.1-18.6	4.8-18.0	5.0-17.8	6.2-18.4
Mean (SD) age (years)	10.7 (4.0)	10.5 (3.8)	11.8 (4.1)	10.4 (4.0)	9.8 (3.8)	10.8 (3.4)
r	0.95	0.92	0.92	0.93	0.91	0.90
MAE (years)	1.1	1.2	1.3	1.2	1.3	1.2

Figure 1.4 BrainAGE estimation across different MRI scanner Sites

The first attempt towards such end using a different approach was fairly recent [5]. The model was developed using deep structured learning, or deep learning, to extract and transform features from the brain MRI images. A pattern called supervised learning was selected. The study also conducted reliability and genetically-influenced tests on the brain-predicted age. The ages predicted from grey matter volumetric maps have an r=0.96 with the chronological ages. The mean absolute error is 4.16 years. Apart from deep learning, the study also used GPR (Gaussian processes regression) for comparison. The main results are summarized in the table below.

Method	Input data	MAE (years)	r	R^2	RMSE
CNN	GM	4.16	0.96	0.92	5.31
	WM	5.14	0.94	0.88	6.54
	GM+WM	4.34	0.96	0.91	5.67
	Raw	4.65	0.94	0.88	6.46
GPR	GM	4.66	0.95	0.89	6.01
	WM	5.88	0.92	0.84	7.25
	GM+WM	4.41	0.96	0.91	5.43
	Raw	11.81	0.57	0.32	15.10

MAE = mean absolute error, r = Pearson's r from correlation between chronological age and brain-predicted age, RMSE = root mean squared error, GM = Grey Matter, WM = White Matter.

Figure 1.5 Chronological age prediction accuracy

1.5 Introduction to Deep Learning

Deep structured learning, or hierarchical learning, belongs to a larger family of methods called machine learning. These methods learn data representations and extract features from them. There are three types of deep learning: supervised, semi-supervised, or unsupervised [10]. This project only makes use of supervised deep learning. In a deep learning system, artificial neural networks are systems that learn from the input data and do specific tasks. A classic example would be an artificial neural network that learns from pictures that either have a cat or have a dog. These pictures are manually labeled as ‘cat’ or ‘dog’ (in a real system this is usually numpy array [0 1] and [1 0]). After some time of training, a good artificial neural network should be able to identify cats in images that are not fed into the system previously with high accuracy. As we

can see, the input layer of this artificial neural network takes in image data and the output layer gives out the classification decision.

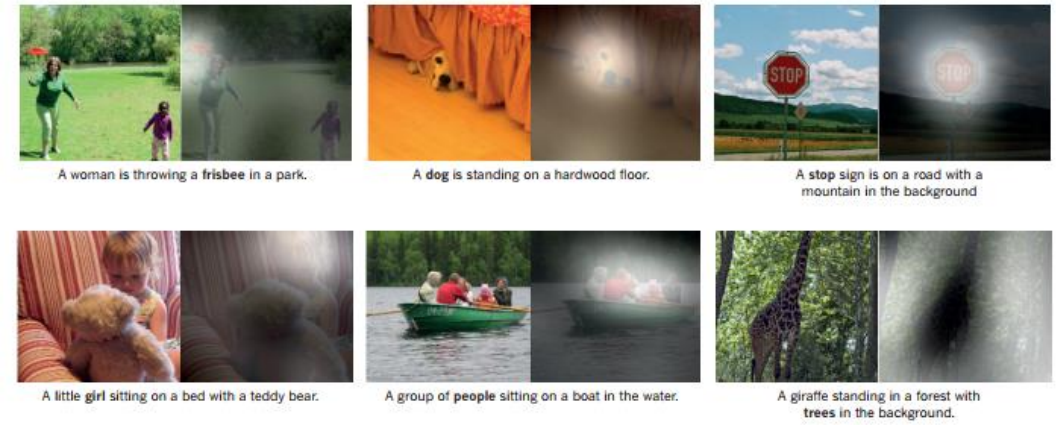

Figure 1.6 Deep Learning For Image Captioning (retrieved from [10])

If a neural network not only has input and output layers but also has multiple hidden layers in between, then it is called a deep neural network. The corresponding learning process and system is called deep learning. The layers are made up of neurons, which have learnable weights and biases. All the neurons in the system receive part of the input, execute dot product operations and update the weights and biases [11]. By doing this, the whole system 'learns'. The hidden layers enable the system to model non-linear relationships by adding activation functions. The extra amount of layers help the system to extract information from the features already extracted from the shallow layers. Recently, as the GPU capacity (Graphics processing unit) of the computer systems keep improving, deep learning has again gained public attention. Applications include CNNs (convolutional neural networks) for image recognition, acoustic modeling and RNNs (recurrent neural networks) for natural language processing.

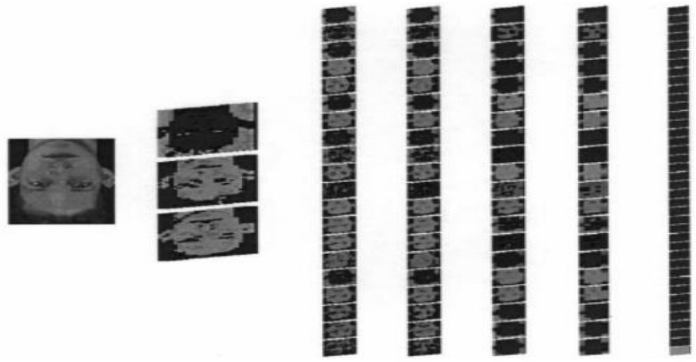

Figure 1.7 Face Recognition using Deep Learning (retrieved from [12])

1.6 Introduction to Convolutional Neural Networks

Convolutional neural networks, usually abbreviated as CNNs or ConvNet, are similar to the neural networks introduced above as that they are also formed by neurons with learnable weights and biases. However, CNN only takes images as input. This assumption allows more efficient implementations and encoding, dramatically reducing the number of parameters needed to achieve a certain level of performance. The typical types of layers in a convolutional neural network include convolutional layer, pooling layer, fully connected layer, and activation layer.

A convolutional layer performs convolution operation to the input. It is worth noting that the 'convolution' here is not the mathematical operation also called 'convolution' that is widely used in probability, statistics, and signal processing. The parameters of a convolutional layer have several learnable filters (we use 8 here as an example). A typical filter size for a three-dimensional gray-scale image is 3×3×3. Therefore, the parameters of the convolutional layer have a shape of 3×3×3×8 (3 pixels for length, width, and height, and 8 for 8 filters). During the forward path, the filters are slid across the width, length, and height of the input volume. The results are activation maps that are composed of dot products of the filter entries and the input at the certain position. As can be seen in the image, each neuron in the neural network is only connected to a certain local region of the input volume. The limit of the connectivity can be tuned by a hypermeter called receptive field. This concept is similar to the receptive field in brain physiology.

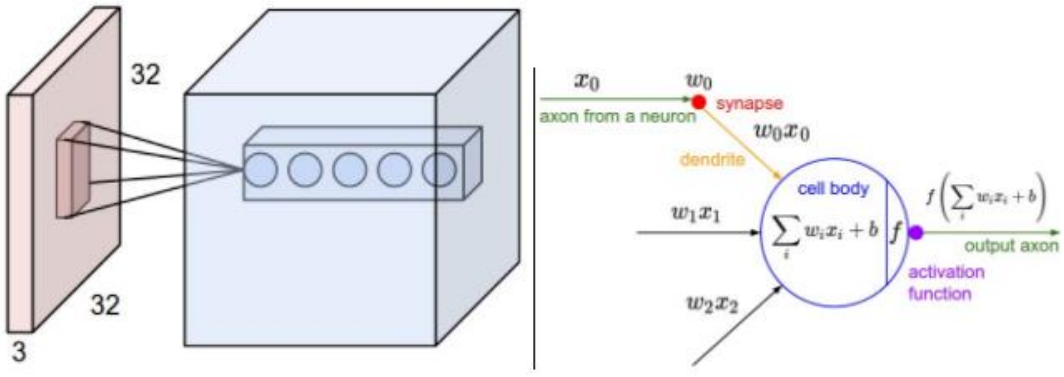

Figure 1.8 Convolutional Layer and a Neuron (retrieved from [13])

It is typical to insert a layer called a pooling layer between successive convolutional layers. In this project, the input of the convolutional neural network is either a 130×130×110 matrix or a 65×65×55 matrix and the output is always a scalar, the estimated chronological age. Therefore, the size of the representative and the number of parameters need to be reduced, and that is the function of a pooling layer. The pooling layers usually function by performing the MAX operation. For instance, the pooling layers used in this project have a common size of 2×2×2 with a stride 1. The output of such pooling layers down-sample the input by 2, discarding 87.5% of the activations. If a 3D pooling layer with a spatial extent F and a stride S accepts a volume of size W×H×D, the output would have a size of W'×H'×D', where W'=(W-F)/S+1, H'=(H-F)/S+1, and D'=(D-F)/S+1.

A fully-connected layer, usually the last layer of a convolutional neural network, has complete connections to all activations in the previous layer. The activations of the fully-connected layer can be calculated with the weight and the bias of itself, producing the final result. The activation layer is used to increase the nonlinearity of the whole system, thus improving the system complexity without disrupting the appropriate receptive fields.

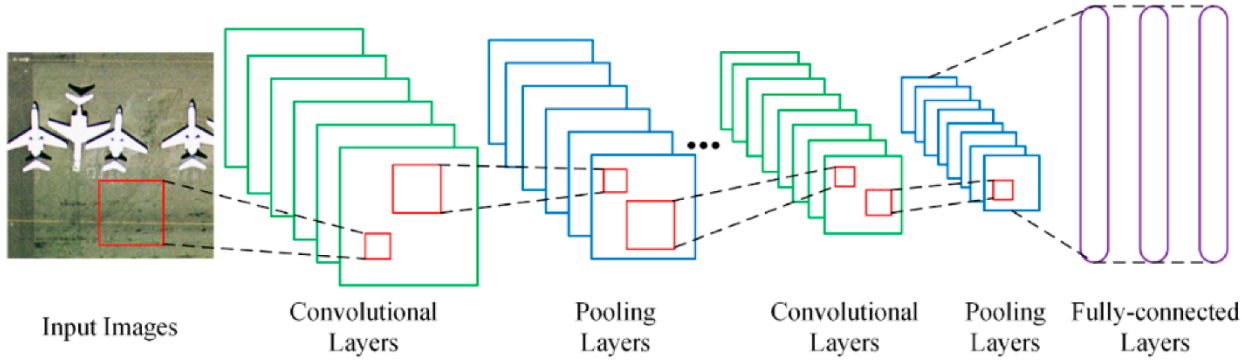

Figure 1.9 Typical Convolutional Neural Network Layers (retrieved from [14])

2. Methodology

2.1. Deep Learning

2.1.1 The Neural Viewpoints and Backpropagation

In a human nervous system, roughly 86 billion neurons are connected with about 10^{15} synapses. In the scale of a single neuron, the input is from the dendrites of the neuron and the output is sent to the axon of the neuron. The axon is connected to dendrites of other neurons via synapses. Synaptic plasticity is defined as the ability of synapses to strengthen or weaken over time [15]. The whole foundation of the neural networks is that the synaptic plasticity, or the weights and biases of the neurons in neural networks, are learnable. The cell body, where dendrites send the signal to, decides whether the signal is large enough to fire. The firing rate of the cell body becomes the activation function of the layers described in the previous section. This analogy takes the assumption that only the frequency of the firing is significant.

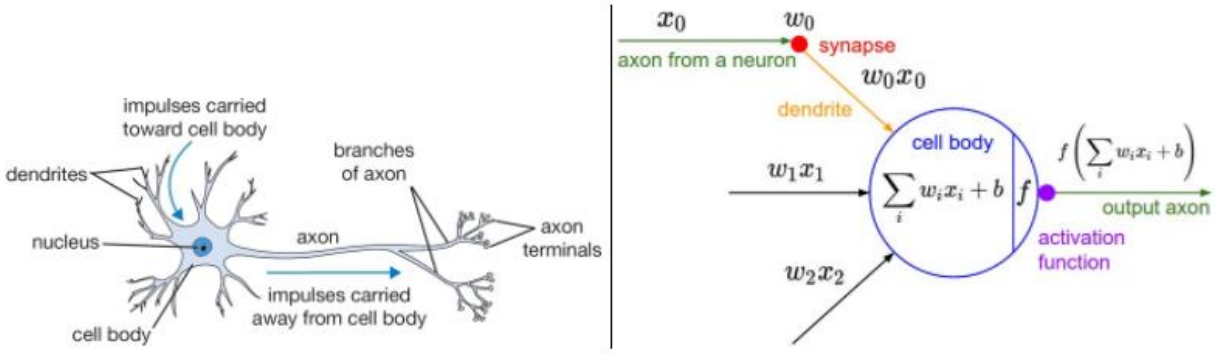

Figure 2.1 Biological Neuron and its model (retrieved from [13])

With a specific error function and a neural network, the backpropagation computes the gradients of expressions with respect to the weights in the specified neural network. It is essentially a recursive application of the chain rule. The ‘back’ in the name stems from the fact that the gradient is calculated backward through the neural network, from the last layer to the first. The partial gradient of a layer is reused for the calculation of gradient in the next layer (in terms of the flow of gradient, or the previous layer in terms of the network structure). It is worth noting that although the use of chain rule renders a single gate of the neuron complex and aware of the entire neural network, the calculation of a single gradient is local and does not require any

external information. The specifics of backpropagation is complex and depends on the activation function selected, and thus and explained in detail here.

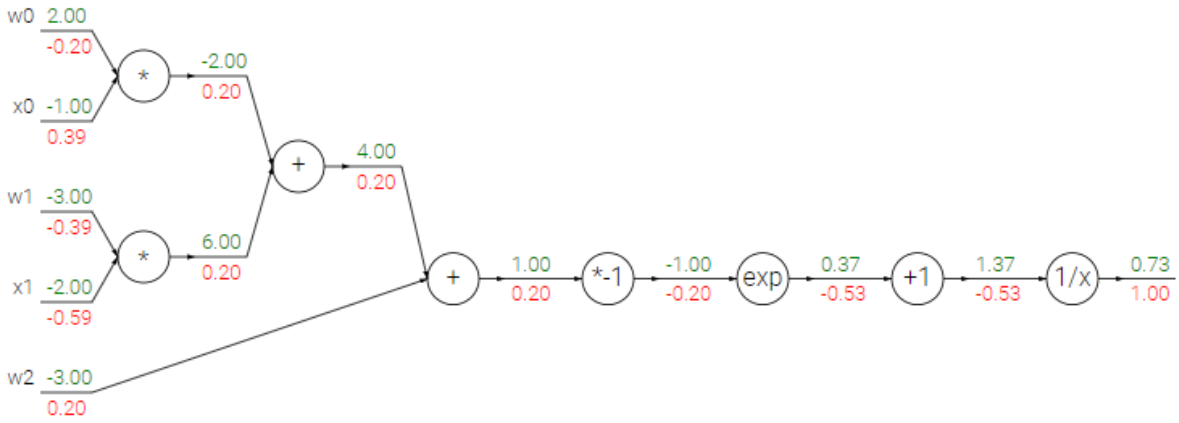

Figure 2.2 Backpropagation example for a simple 2D network, note that the gradients are calculated backward (retrieved from [13])

2.1.2 Cost Function

In this project, the change of the weights and biases in the networks are determined by the backpropagation described in the previous section. However, when deciding the direction where the weights and biases should be headed, an index that describes our unhappiness with the outcomes is needed. This is where the loss function comes into play. In this project, since the last layer is a regression rather than classification, the loss function defined is simply a mean square error between the estimated age and the actual chronological age. When this value is large, we are not satisfied with the weights and biases. Therefore, by minimizing the cost function, the system learns.

2.1.3 Optimization

The goal of a well-performing optimization is to find the weight combination that minimizes the cost function described in the previous section. For the illustrational purpose, the optimization strategy of stochastic gradient descent (SGD) is first introduced here and then the actual optimization strategy used in this project, Adam, is then analyzed.

Given an objective cost function Q, which is the least square in this project, the cost function over the entire sample range is simply $Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$. To minimize this, a standard gradient descent method takes the form $w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$, where η is the learning rate of the system. After performing the algorithm for multiple training representatives, the sum-gradient is evaluated and the weights are updated towards the direction of minimizing the cost function. As we can see, the learning rate is kept constant across the whole iteration. This makes the system extremely sensitive to the learning rate, which is usually defined empirically. When the defined learning rate is too large, the system will not reach the minima and the loss function fluctuate drastically. When the learning rate is too small, the system needs longer time to converge, which is usually problematic due to the heavy computational load of deep learning. After a few generations of update, an optimization method called Adam is developed. In Adam, the learning rate is divided by the moving average of the recent gradients as well as the second moments of the gradients. If we denote the parameter as w, the cost function as L, the Adam's update rule is given by

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$
$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^t}$$
$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^t}$$
$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

(Retrieved from [16])

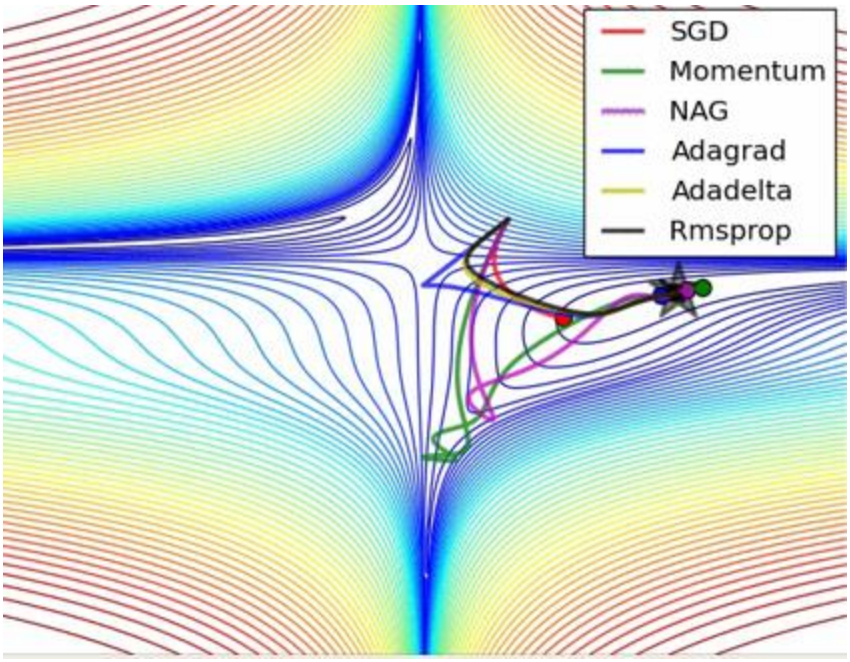

Figure 2.3 Optimization Strategies Take Different Paths (retrieved from [13])

2.1.4 Activation Functions and Weights Initialization

The activation layer is usually inserted right after the convolutional layer. Each activation function in the activation layer performs the defined mathematical operation on the input. For historical reason, a function called sigmoid with a form of $\sigma(x) = 1 / (1 + e^{-x})$ was used in many neural networks. The function compresses the input into the range between 0 and 1. Specifically, large negative or positive numbers become 0 or 1. This particular activation function was utilized for its similarity with biological neuron firing rate relationships. However, the function has two main defects. The first one is that saturation occurs when the input is large in magnitude. The direct result of this is the killing of gradient and the signal flowing of the specific neuron. Not being symmetrical with respect to the origin is the function's second drawback. During gradient descent, the gradient calculated will be either all positive or negative if the input is with the same sign. This will introduce an undesired zig-zagging motion during optimization.

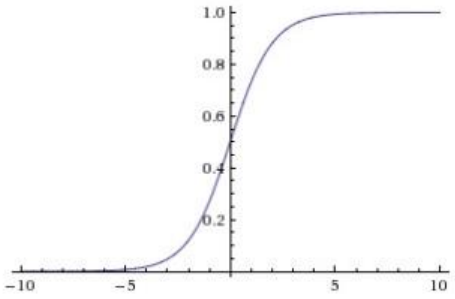

Figure 2.4 Sigmoid Function (retrieved from [13])

The activation function that this project adopts is called the Rectified Linear Unit, or ReLU. It simply sets all negative input to zero. Mathematically, this is simply f(x) = max(0,x). This function has been found empirically to accelerate convergence and decrease computational burden. However, if some part of the data is knocked off the data manifold, it will never be activated again, and thus dead. This problem usually happens when the learning rate is set too high and did not occur in this project.

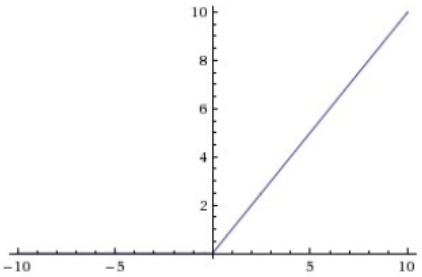

Figure 2.5 ReLU Function (retrieved from [13])

Assuming that the data is properly normalized (though not applicable to this project), it is reasonable to infer that about half of the weights in the neural network will be positive and another half negative. In this case, initializing all weights to zero seems to be a sound idea. However, the strategy fails because it leads to the situation where all neuron computes the same output, generates the same gradient and experiences the same change at all time. This is obviously undesirable for the neurons are supposed to stand for different features.

In this project, two initialization methods were investigated. In the first phase, the objective of estimating brain ages was simplified as a deep learning classification problem, where the chronological ages were divided into 9 or 18 groups. The task of the neural network is simply finding the right group given a specific MRI brain image. In this phase, the small random number initialization method was used. All parameters, including weights and biases, were initialized as very small numbers to avoid the problem encounter in zero initializing described above. In the second phase, the problem is treated as a regression problem. As the outputs of regression problems are less confined, more complicated initialization is required to avoid gradient explosion. In this case, Xavier initialization was used. Xavier initialization sets the weights in the neural network by drawing them from a distribution with zero mean and variance $Var(w) = \frac{1}{n_{in}}$ where w is the initialization distribution and n_{in} is the number of neurons [18]. This ensures that the weights are set too large and thus the gradients will not explode.

2.1.5 Regularization

During the preparation for a deep learning project, the data is usually divided into two parts, training data and test data (or three parts training, validation, test in some cases). A common scenario encountered after adequate training is that the error on the training data is impelled to a rather small value, but the error on the test data is relatively large. In other words, the model has 'memorized' the information presented in the training data, but it has not learned the generalization, the common features of the training set and test set. This problem is termed overfitting.

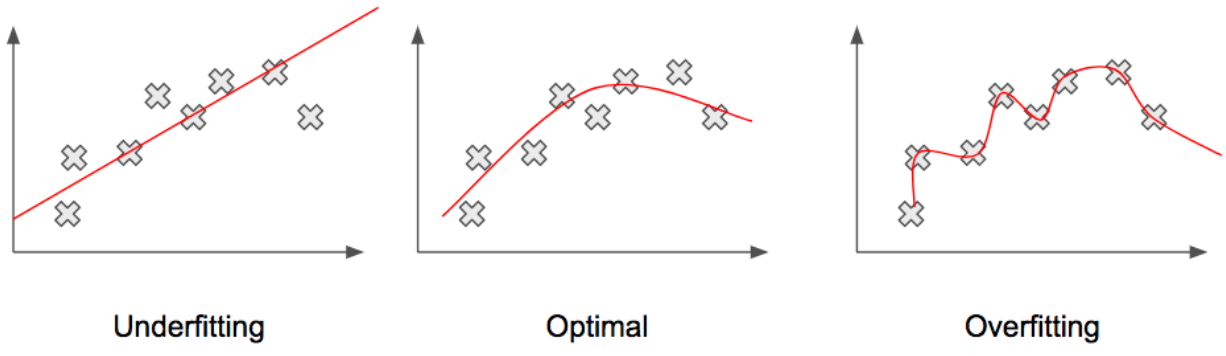

Figure 2.6 the problem of overfitting (retrieved from [22])

In order to solve this, several techniques have been proposed to increase generalization. Among them, L2 regularization is probably most commonly used. By adding 1/2 $\lambda\omega^2$ to the objective of every ω in the neural network designed, where λ is termed the regularization strength, L2 regularization forces the weights to decay linearly during gradient descent. The advantages of this method include spurring the neural network to utilize all its inputs instead of part of the inputs.

This project makes use of a relatively new regularization technique called dropout. During training, a neuron is only kept active (not set to zero) with a probability of p, which is a hyperparameter to be tuned during training phase [17]. That is to say, only p of the whole graph remains. If the output of a single neuron is x before drop out, then the output would be px + (1-p) × 0 after the implementation of dropout. The power of dropout has been proved empirically. The source of its power, however, is still an active area of research.

This project implements dropout after each convolutional layer to prevent overfitting. The hyperparameter p has been tuned separately to achieve the optimal performance under each circumstance (different input size, learning rate, repository selection).

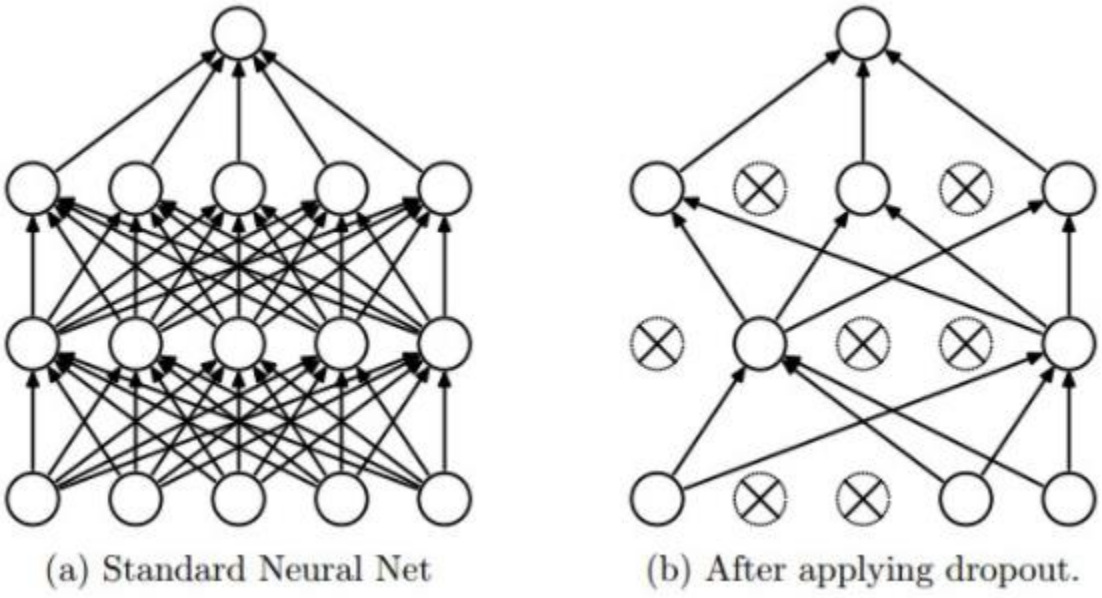

Figure 2.7 Dropout implementation (retrieved from [17])

2.1.6 3D Convolution and Regression

Most of the existing convolutional neural networks take 2D images as inputs. However, given the nature of MRI brain images, this project designed a convolutional structure based on 3D convolution proposed in [19]. In two-dimensional convolutional neural networks, the convolutional operations are applied to the two-dimensional feature maps. In this project, however, remaining the three-dimensional structure will allow more information to flow in the convolutional neural networks. To this end, the 3D convolution is achieved by stacking multiple feature cubes together.

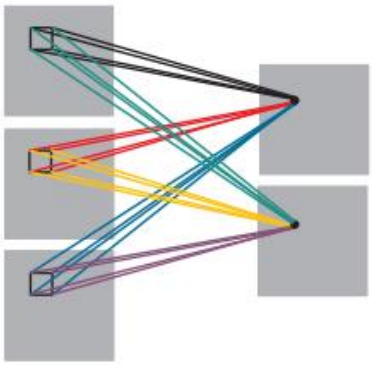

Figure 2.8 3D Convolution (retrieved from [19])

The typical deep learning classification problem uses cost function such as SVM in the last layer of the neural network. Since the output here is a scalar, the estimated age, mean square error is used as a cost function in the last layer. This regression in deep learning can be understood as a classification problem with only one group. The output is a numpy [x] instead of typical [x y] in two-group classification problems.

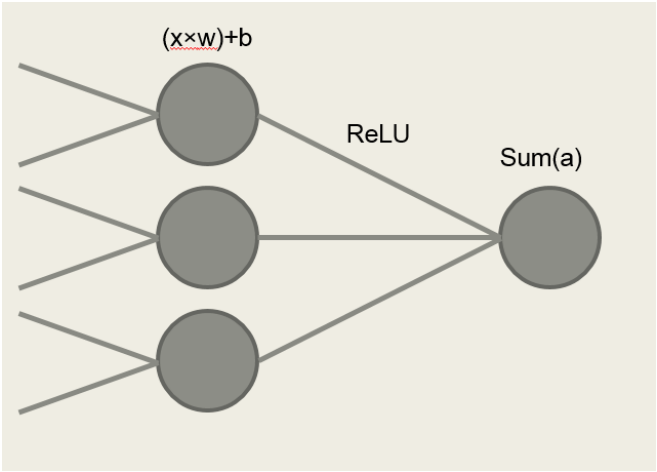

Figure 2.9 Regression in Deep Learning

2.1.7 CNN Structures

The final structure selected is shown below. It is a few repetitions of the combination of a 3×3×3 convolutional layer and a 2×2×2 max-pooling layer. After each convolutional layer, a ReLU function is implemented and a dropout layer with a dropout rate of 0.5-0.9 ensues. The final output is a scalar calculated using mean square error. Two types of sample size were selected: 65×65×55 or 130×130×110.

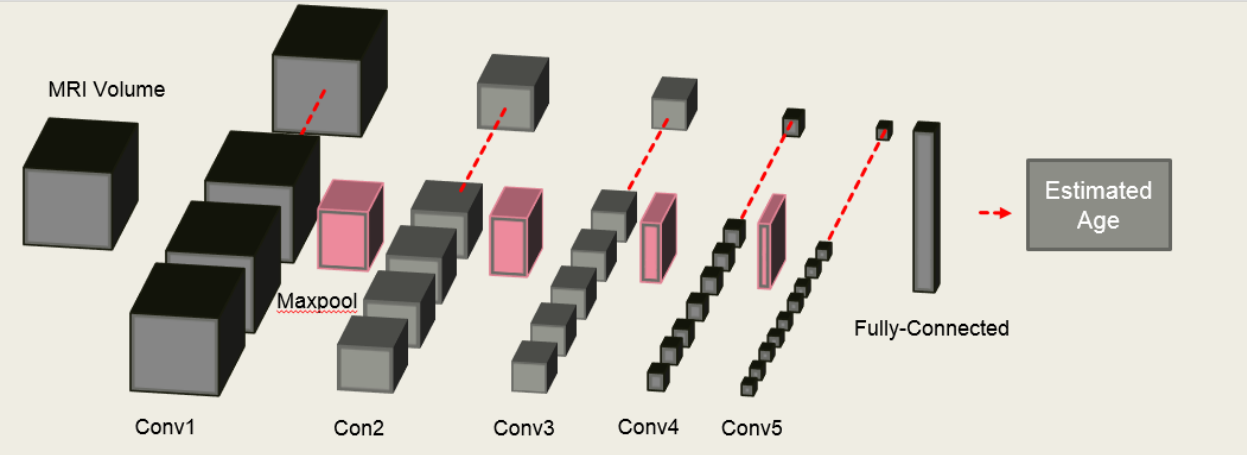

Figure 2.10 Convolutional Neural Network Structure Diagram

Layer	Parameters	Activation	Output
Input			65×65×55×1
Conv1	3×3×3	ReLu	65×65×55×8
Maxpool	2×2×2		33×33×28×8
Conv2	3×3×3	ReLu	33×33×28×16
Maxpool	2×2×2		17×17×14×16
Conv3	3×3×3	ReLu	17×17×14×32
Maxpool	2×2×2		9×9×7×32
Conv4	3×3×3	ReLu	9×9×7×64
Maxpool	2×2×2		5×5×4×64
Conv5	3×3×3	ReLu	5×5×4×128

Maxpool	2×2×2		3×3×2×128
Fully-connected			2304
Regression			1

Table 2.11 Parameters for input size 65×65×55

Layer	Parameters	Activation	Output
Input			130×130×110×1
Conv1	3×3×3	ReLu	130×130×110×8
Maxpool	2×2×2		65×65×55×8
Conv2	3×3×3	ReLu	65×65×55×16
Maxpool	2×2×2		33×33×28×16
Conv3	3×3×3	ReLu	33×33×28×32
Maxpool	2×2×2		17×17×14×32
Conv4	3×3×3	ReLu	17×17×14×64
Maxpool	2×2×2		9×9×7×64
Conv5	3×3×3	ReLu	5×5×4×128
Maxpool	2×2×2		5×5×4×128
Fully-connected			12800
Regression			1

Table 2.12 Parameters for input size 130×130×110

2.2 Data Acquisition

All MRI brain imaging data is collected from four repositories: Autism Brain Imaging Data Exchange (ABIDE), Autism Brain Imaging Data Exchange Second Phase (ABIDEII), Open Access Series of Imaging Studies (OASIS), and IXI. All data was approved by the sharing scheme of international neuroimaging data-sharing initiative (INDI) or OASIS site.

The raw data is in ANALYZE 7.5 format or NIFTI format. The mprage images of the subjects were selected and transformed into numpy array format for further processing. All images fed into the model are T1-weighted. As the assumption of the model is that healthy individuals' chronological ages are close to their brain ages, all brain images from individuals with autism and Alzheimer's disease were excluded. These images, however, may be helpful in further analyzing the model's efficiency in detecting brain-related diseases. The data information is shown below.

Name	**Size (N)**	**Age RANGE**	**Age Average**
OASIS	316	18-96	52.4
ABIDE	574	6-56	17.0
IXI	590	20-86	49.4
ABIDEII	593	6-64	14.9
Total	2072	6-96	31.0

Table 2.13 Raw data information

2.3 Data Preprocessing

Since the raw data is bulky, collected from different repositories, and obtained in different conditions, minimal preprocessing is needed for the data to qualify as the input of the neural network. All MRI images are preprocessed by following five steps. First, all images are resampled according to the pixel dimension information read from the image header files. This ensures that all images will have the same size and resolution, thus increasing between-scanner reliability. Second, the images are cropped or padded to the desired shape (130×130×110 or 65×65×55). Third, the indexes of the images are shuffled and recorded. By doing this, the test data selected will be randomly selected from all repository, thus increasing the between-

repository reliability. Fourth, the mean image of all the training data is computed and is subtracted from all training and test data. It is worth noting that the test data does not contribute to the mean image. This is because the training data, and only training data, needs to have zero-mean for better training performance. Finally, the data, including the corresponding chronological age information, is combined into batches with a size of 16. Note that the images have already been shuffled, thus the data is not influenced by the order of the repositories.

Function Name	Argument(s) and Return	Description	Notes
def loadData def saveDataset	Input: sub_ID (the ID specific to a single repository) Output: training_data (the numpy array), training_pixdim(the pixel dimension to be used in resampling) Input: dataset (numpy array), name (file name)	Load raw data (.nifti & Analyze) and save it in .npy format.	Syntax appropriate for all repositories. Use of try and except block may lead to error if a new repository is used.
def print_data_shape def show_slices	Input: dataset (numpy array) Input: dataset (numpy array)	Printing out data shape for a sanity check Visualize the 3D image from numpy array.	

def resample def crop_center	Input: image (the original numpy array), pixdim (the pixel dimension read from the image header), new_spacing (the desired spacing) Output: image (the edited numpy array)	With the build-in np.pad function from the numpy library, the images are resampled, cropped and padded sequentially to qualify as inputs.	The resample function needs a picture pixel dimension input which is read from the image header file. The reason why some of the data needs to be cropped as well as padded is that the dimensions are different for different axis.
	Input: img (the original numpy array), cropx, cropy, cropz (the individual cropping along three dimensions) Output: image (the edited numpy array)		
def shuffle		Shuffle the image indexes and save the new index set to a .csv file.	The new indexes are to be added to the main .csv file with the column name 'shuffledID'.
def calculate_mean		Calculate the mean image and save the numpy array.	The mean image should only include training data.
def combined_preprocess	Input: start (the shuffled ID of the first image to be combined), end (the shuffled ID of the last image to be combined)	Subtract the mean image from the images and combined them into batches.	
def conv3d	Input: x (the layer input), W(the layer weights) Output: the layer output	Define the 3D convolutional layer in each block.	With stride 1 along all dimensions.

def maxpool3d	Input: x (the layer input), W(the layer weights) Output: the layer output	Define the max pooling layer in each block.	With the size of 2×2×2.
def convolutional_neural_network	Input: x (the MRI volume input)	Define the CNN structure	
def train_neural_network		Train the network according to user-defined hyperparameters and save the model.	
def rotate_oasis		Rotate all images from the repository OASIS to the same orientation with images from ABIDE and ABIDEII	The raw images from OASIS have a different orientation with images from ABIDE and ABIDEII. However, the original orientation will be used in rotation test. Therefore, in the following section, images with the original orientation will be denoted as 'after rotation'.
def shape_oasis		Reshape the OASIS images due to mismatch of shape after rotation	

def show_slices	Input: slices (the particular slices to be shown)	Visualize the image for a sanity check	The input should be the slice to be shown (for example img [50,:,:]) instead of the whole image.

Table 2.14 A summary of user-defined functions

2.4 Software and Hardware

All codes were written in Python, a high-level programming language that has a machine learning framework called tensorflow. Due to the high requirement of computational ability of deep learning, the training part of the project was run on google cloud platform. The following versions of the software were used: tensorflow-gpu-1.2.0, CUDA 8.0, cuDNN 5.1.

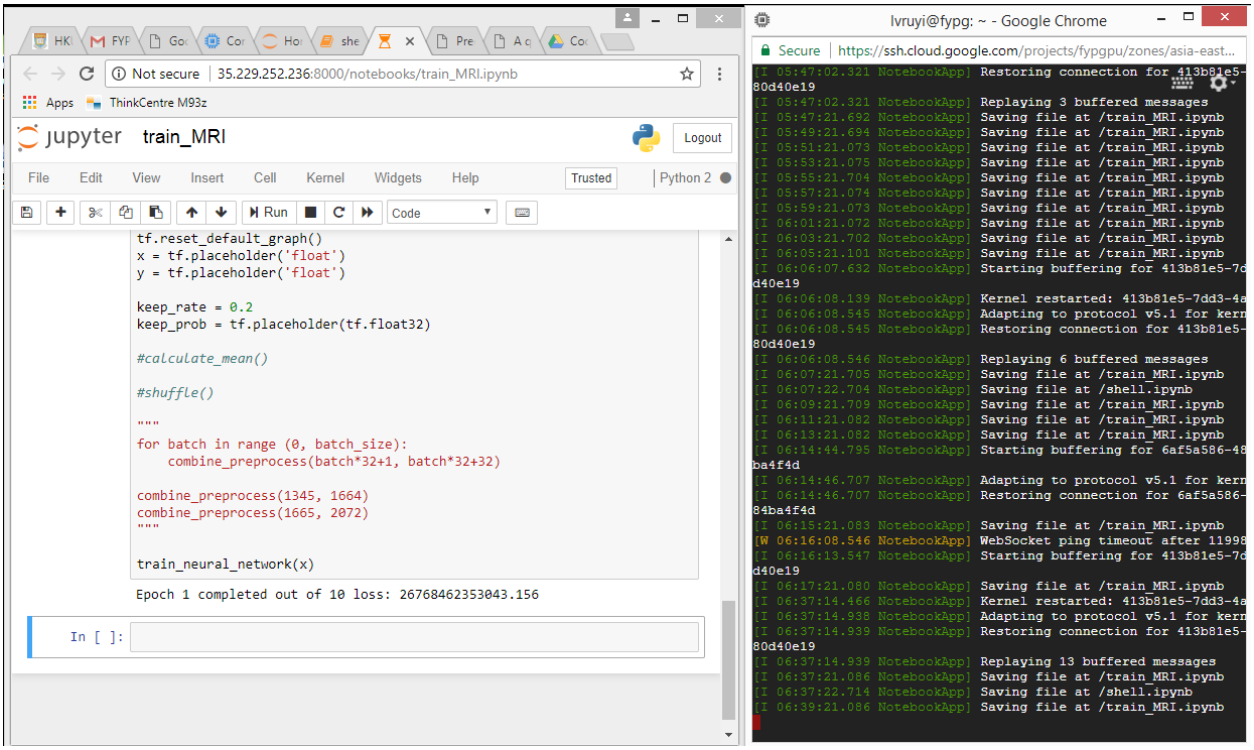

Figure 2.15 Google Cloud Platform and iPython Interface

The hardware is extremely important for the speed of training. Since one of the main objectives of this project is to develop a high training speed model, the hardware was carefully chosen. 8 vCPUs were used with 52 G memory. The GPU was NVIDIA Tesla P100 in the operating system Ubuntu 16.04 LTS.

2.5 Evaluation of Performance

The root mean square error and the Pearson correlation coefficient were used to evaluate the performance of the predicting model. If the chronological ages are denoted as Y_{chro} and the estimated brain ages are denoted as Y_{brain} , then the Pearson correlation coefficient is computed as $r = \frac{\mathrm{cov}(\mathrm{Ybrain}-\mathrm{Ychro})}{\sigma_{\mathrm{Ybrain}}-\sigma_{\mathrm{Ychro}}}$, where cov is the covariance. The root mean square error is calculated as RMSE = $\sqrt{\frac{\sum_{t=1}^{n}(\mathrm{ybrain}-\mathrm{ychro})^2}{n}}$, where n is the sample size.

3. Results

3.1 Data Visualization

When adopting bio-imaging in the field of deep learning, one of the gravest obstacles is the variability of the images introduced by different scanner settings. To preliminarily exclude the possibility that the images in different repositories are too different to be generalized by the neural network, the mean image, as well as the representatives from different repositories, are visualized.

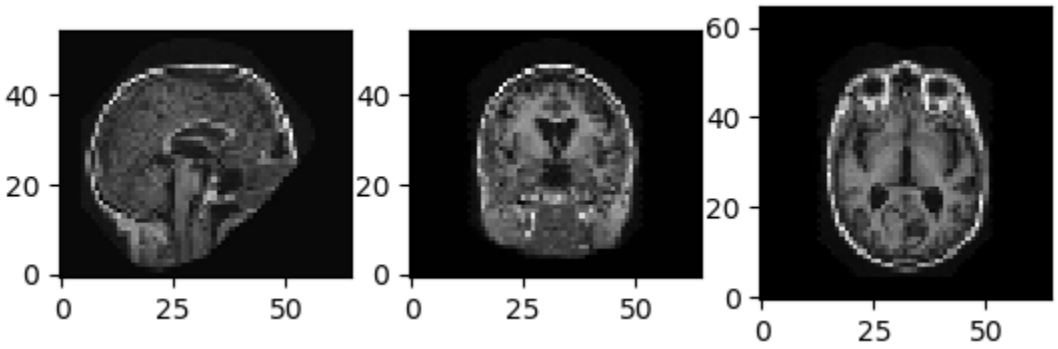

Figure 3.1 OASIS before rotation 65×65×55

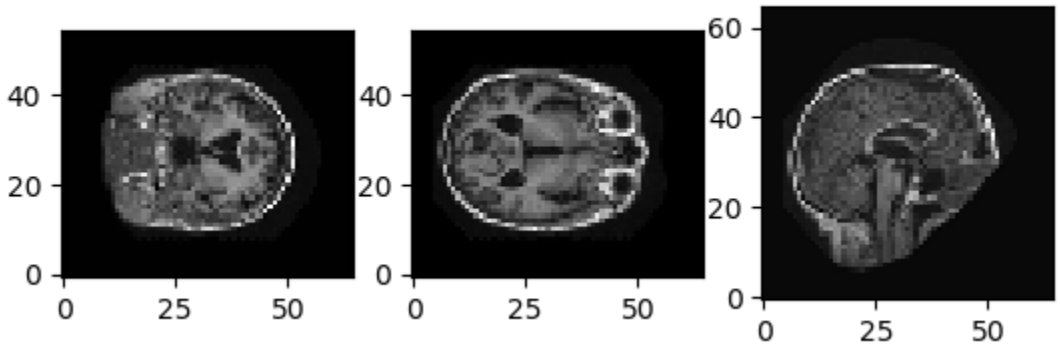

Figure 3.2 OASIS after rotation 65×65×55

In order to test the model's ability to extract features from brains in different orientations and to look for clues about how brain ages are estimated, all images from one of the repositories, OASIS, were rotated. All images were rotated along z axis clockwise for 90 degrees and then rotated along x axis counterclockwise for 90 degrees. Images from other repositories stayed the same. The dataset with the rotation and the dataset without the rotation were trained separately. Further details will be explained in later sections.

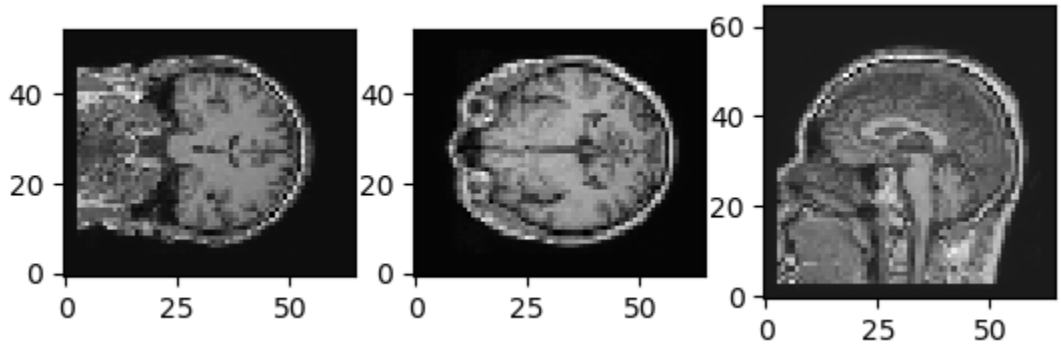

Figure 3.3 IXI 65×65×55

In order to test the model's robustness in terms of taking in data that had not been preprocessed, images from a repository, IXI, were included. The non-brain tissues remained in these images. Again, the dataset with the modification and the dataset without the modification were trained separately. It should be noted that this test was conducted independently from the rotation test. In other words, the rotated images and the images with non-brain tissue do not exist at the same time in any of the trained model.

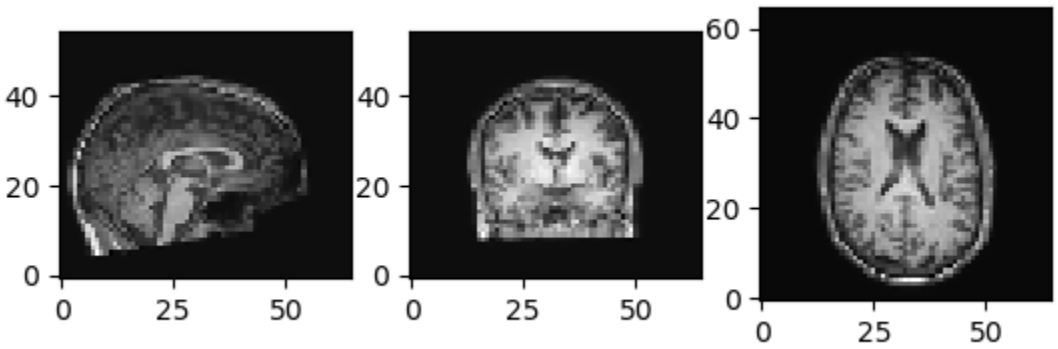

Figure 3.4 ABIDEII 65×65×55

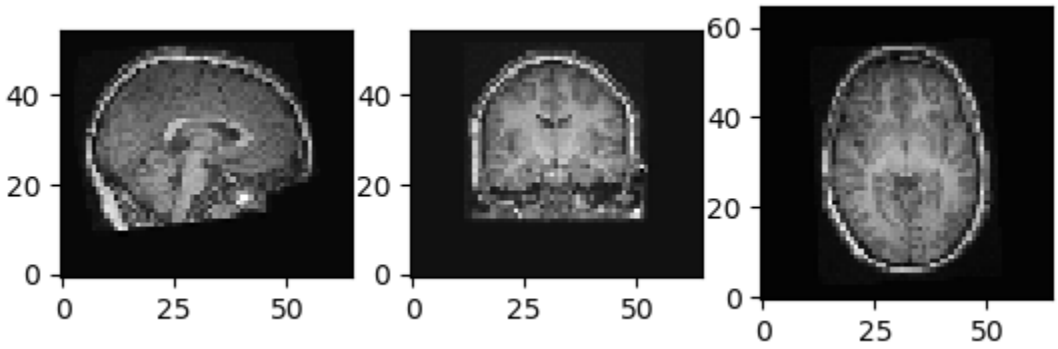

Figure 3.5 ABIDE 65×65×55

In a convolutional neural network, images are usually minimally-preprocessed. Here, the mean image of all the training data was computed and subtracted from all the images, including both training and test data. Notice only the mean image without the rotation data and the data with non-brain tissues is shown here.

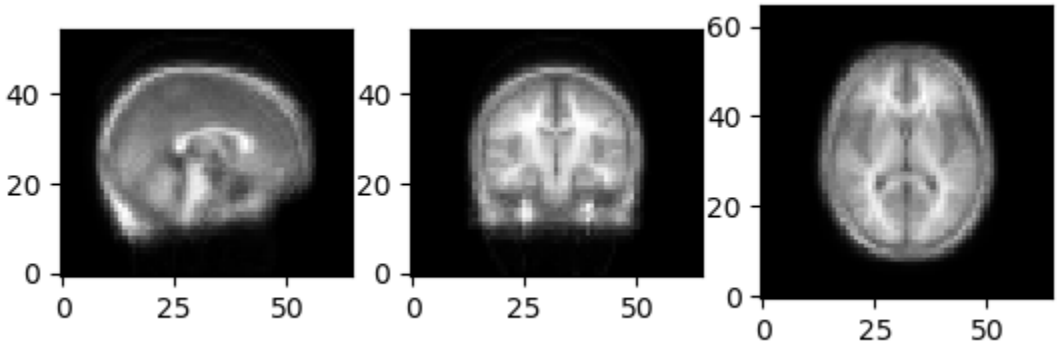

Figure 3.6 Mean Image 65×65×55

In order to investigate the effect of sample size, two image qualities were used: 65×65×55 and 130×130×110. The visualizations of the larger images are shown below. The rotation and the inclusion of non-brain tissue were the same for 130×130×110 images.

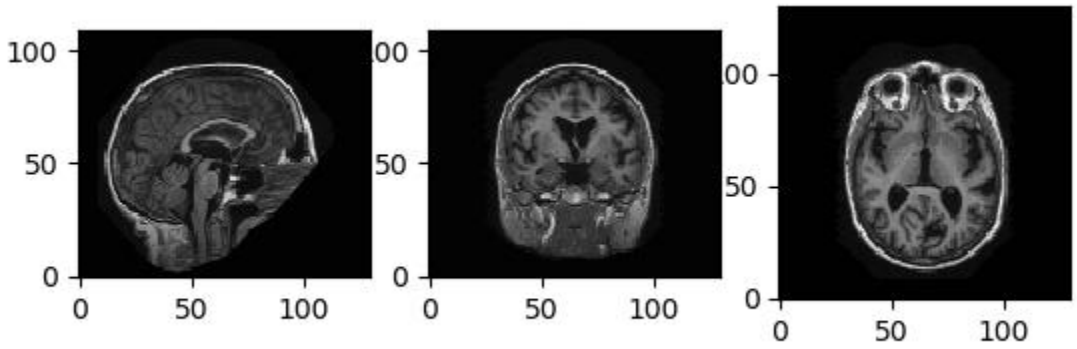

Figure 3.7 OASIS before rotation 130×130×110

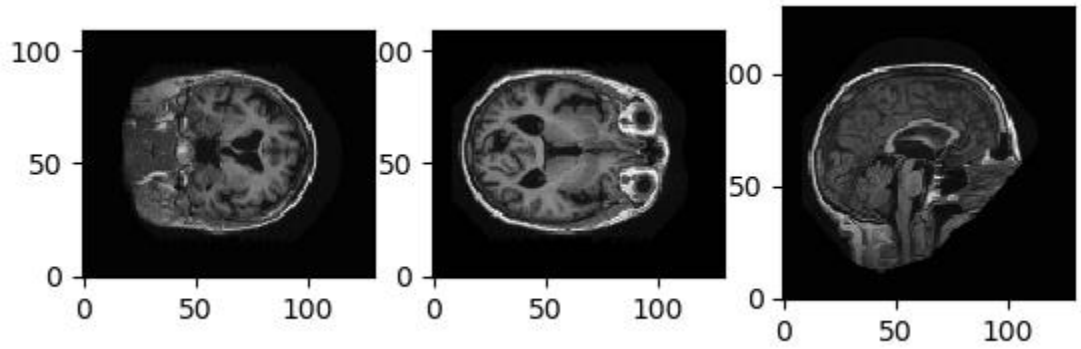

Figure 3.8 OASIS after rotation 130×130×110

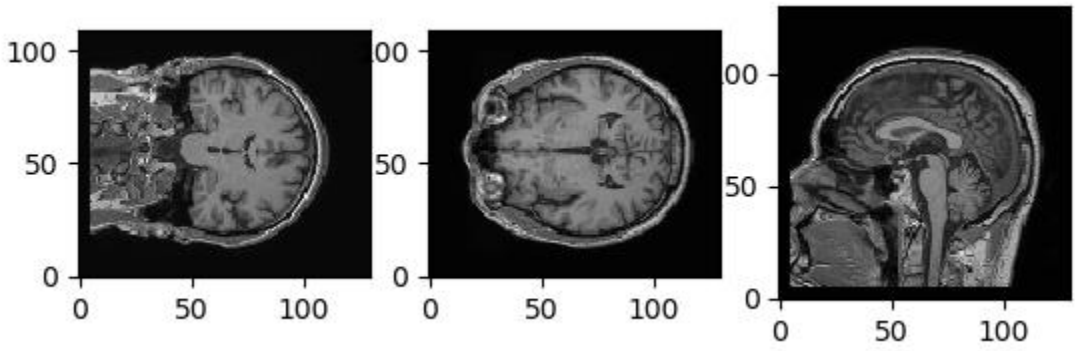

Figure 3.9 IXI 130×130×110

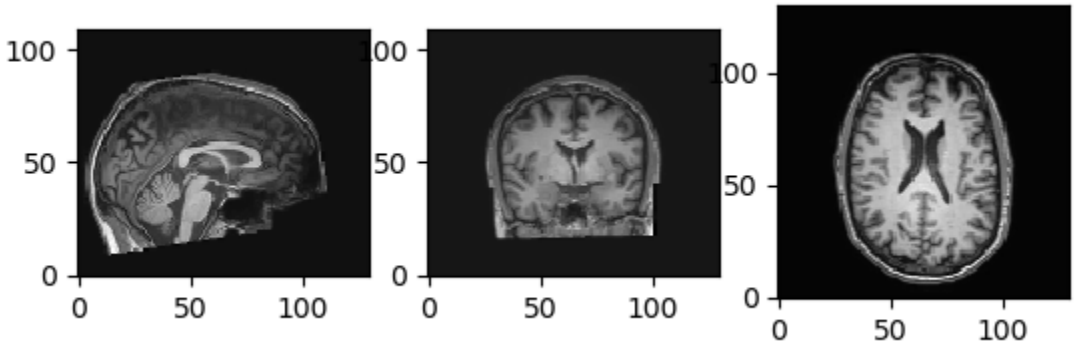

Figure 3.10 ABIDEII 130×130×110

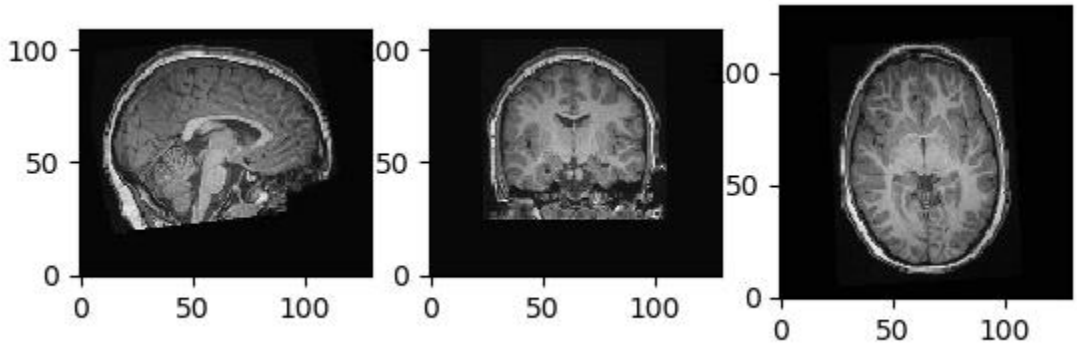

Figure 3.11 ABIDE 130×130×110

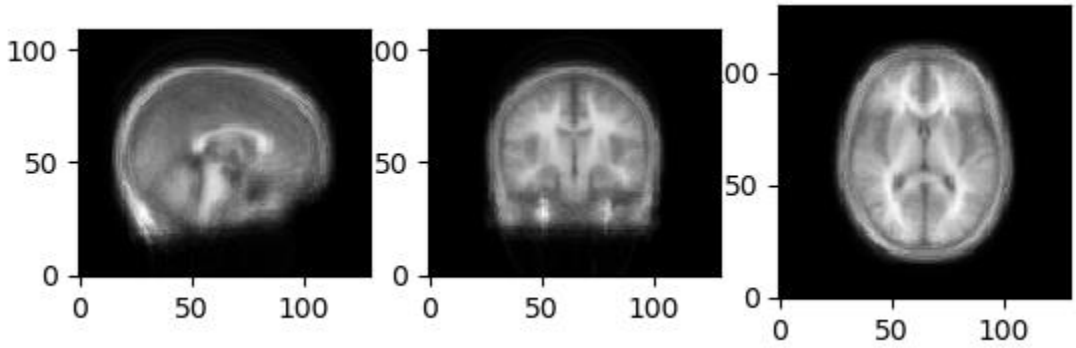

Figure 3.12 Mean Image 130×130×110

3.2 Best Performance Model

For the purpose of testing the robustness of the model as well as investigating the mechanisms behind the brain-estimated age, four databases were developed and were trained separately. All preprocessing (mean image) and shuffling were individualized for different databases.

Database	1	2	3	4
Data size	65×65×55	65×65×55	65×65×55	130×130×110
Source Repositories	Not rotated OASIS; ABIDE; ABIDE II.	Rotated OASIS; ABIDE; ABIDE II	Rotated OASIS; IXI; ABIDE; ABIDE II.	Rotated OASIS; IXI; ABIDE; ABIDE II

Table 3.13 A summary of databases

During training, the training data cost function value, the test data cost function value, and the Pearson correlation coefficient of the test data are printed in real-time. The model weights are saved with a five-epoch interval and ten estimated ages are printed out for a sanity check.

```
Epoch 406 completed out of 20000 cost: 29.16933423814007
avgvalicost 94.47368050805886
(2, 272)
pearson: [[1.         0.88840121]
 [0.88840121 1.        ]]
Epoch 407 completed out of 20000 cost: 32.11355309350048
avgvalicost 102.59554573116108
(2, 272)
pearson: [[1.         0.89223496]
 [0.89223496 1.        ]]
Epoch 408 completed out of 20000 cost: 28.073025566497016
avgvalicost 100.71503017989657
(2, 272)
pearson: [[1.         0.87949728]
 [0.87949728 1.        ]]
Epoch 409 completed out of 20000 cost: 29.151552851588846
avgvalicost 92.26097698451706
(2, 272)
pearson: [[1.         0.89024352]
 [0.89024352 1.        ]]
Epoch 410 completed out of 20000 cost: 32.63009313967867
avgvalicost 88.84402981818337
(2, 272)
pearson: [[1.         0.89481227]
 [0.89481227 1.        ]]
label value: [25] ;    estimated value: [15.997336]
label value: [11] ;    estimated value: [12.059135]
label value: [20] ;    estimated value: [27.951054]
label value: [14] ;    estimated value: [26.221176]
label value: [12] ;    estimated value: [12.286159]
label value: [22] ;    estimated value: [56.450542]
label value: [9] ;    estimated value: [17.422153]
label value: [80] ;    estimated value: [53.22381]
label value: [22] ;    estimated value: [43.6266]
label value: [14] ;    estimated value: [14.615014]
```

Figure 3.14 A sample of training log

The model with the best performance was achieved using dataset 2 with an RMSE of 9.66 years and Pearson Correlation Coefficient of 0.90.

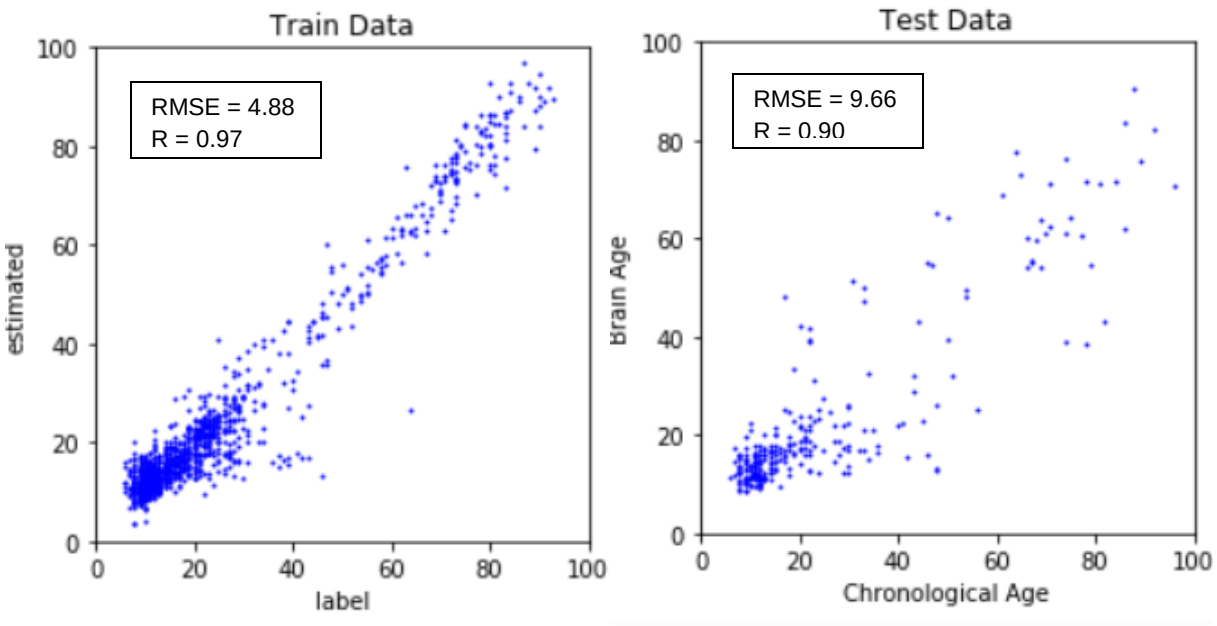

Figure 3.15 Performance of the model trained by dataset 2

3.3 Image Rotation

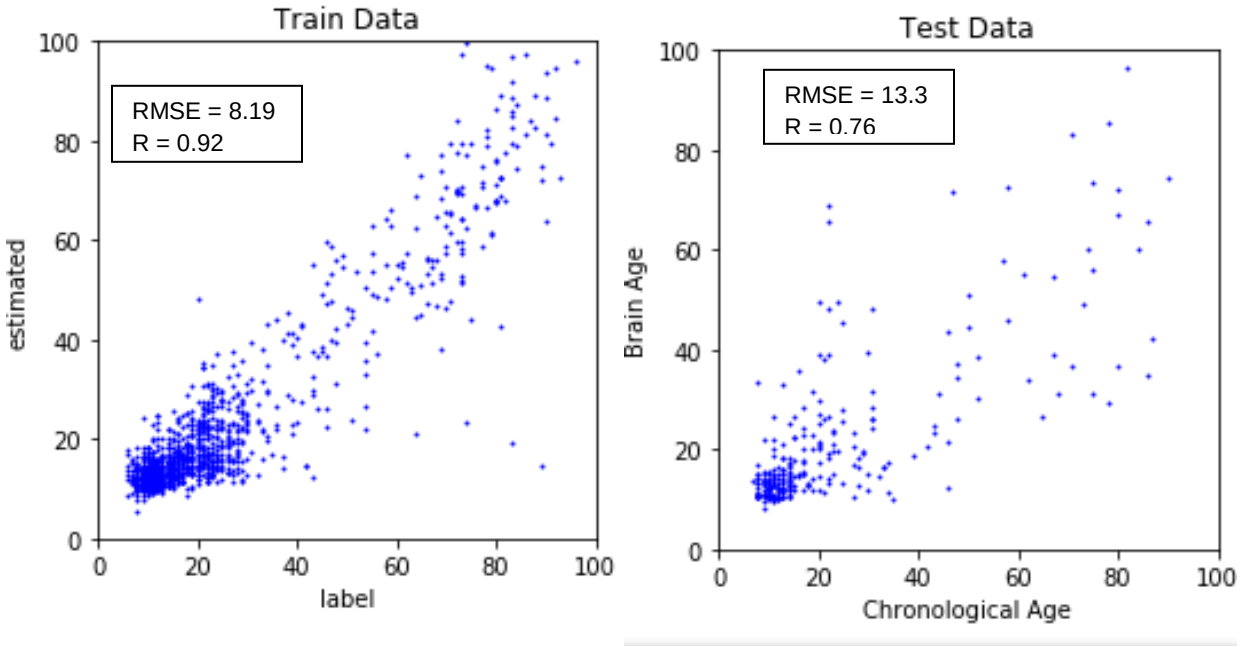

Figure 3.16 Performance of the model trained by dataset 1

The best performance model of dataset 2 took around 6.9 hours to train, experiencing 410 epochs in total. All images from OASIS were rotated along z-axis clockwise for 90 degrees and then rotated along x-axis counterclockwise for 90 degrees. Sound performance, even better than the one without rotating, is achieved for the dataset. It seems that the model has the capability to take in data with different orientations.

3.4 Non-brain tissue

To investigate the model's potential to estimate brain ages in clinical settings, which usually have different scanner conditions and preprocessing procedures, the models trained by dataset 2 and dataset 3 are compared. The only difference between the two datasets is the inclusion of non-brain tissue in part of the images in dataset 3. The performance of model trained by dataset 2 has already been shown in the last section.

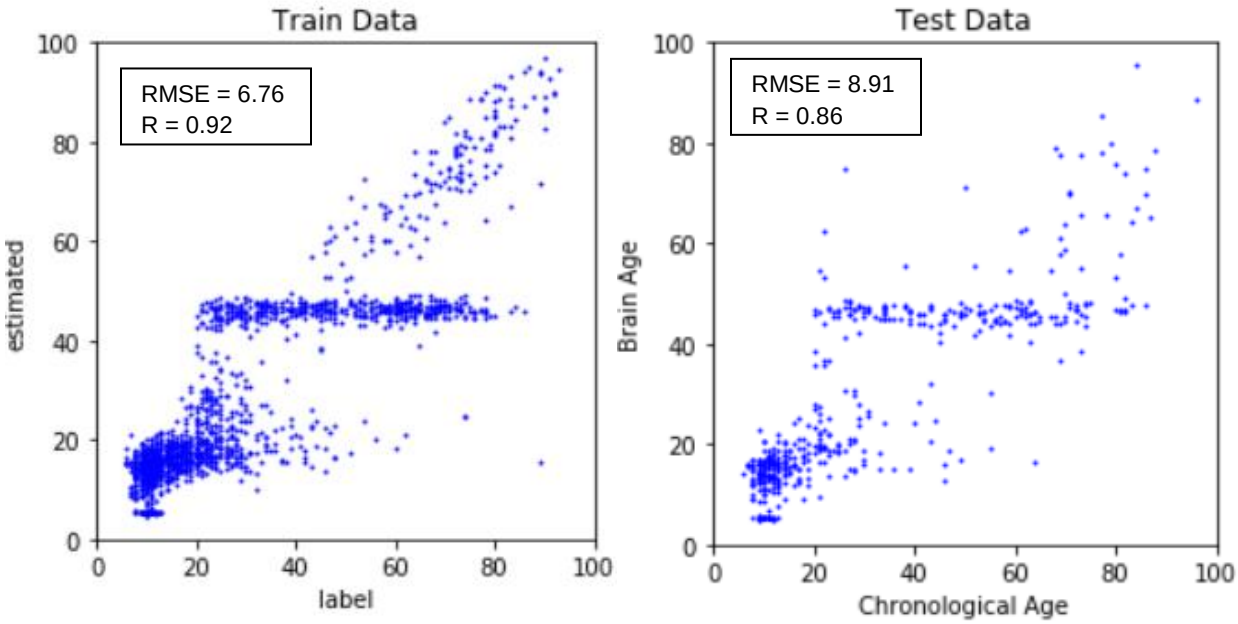

Figure 3.17 Performance of the model trained by dataset 3

The model took 2.8 hours to train. A key observation here is that a 'wall' around the value of 48 was formed in the model trained by dataset 3. After consultation with the original data, it is noticed that the images within this 'wall' are all from IXI, the repository that did not exclude the

non-brain tissue. Therefore, a reasonable inference here is that the neural network specified did not have the capability to exclude the non-brain tissues. The data with non-brain tissue was simply pushed towards the average age during training.

3.5 Dropout Interference

Dropout is a technique to avoid overfitting. Although the mechanism of dropout is relatively simple, its source of power is still a current field of research. The training data of dataset 2 is used here as an example to investigate the specific role of dropout in estimating brain-ages. During training, the dropout layer with a probability of 0.8 is added after each convolutional layer and the fully-connected layer. The neural network's performance on the training data is better than its performance on test data, which is fairly normal in deep learning. Therefore, although there is a gap between training performance (RMSE = 4.88 years) and test performance (RMSE = 9.66 years), mass overfitting has been successfully avoided by using dropout.

When performing classification tasks, such as cat and dog classification, it is generally suggested to use dropout during training but to exclude dropout when testing. The removal of dropout simply means that all neurons will be effective. However, in terms of the brain-age estimation model in this project, the results are quite different with and without dropout applied. When dropout is removed, the correlation coefficient is still similar, but all brain ages become approximately 0.7 of the original estimations.

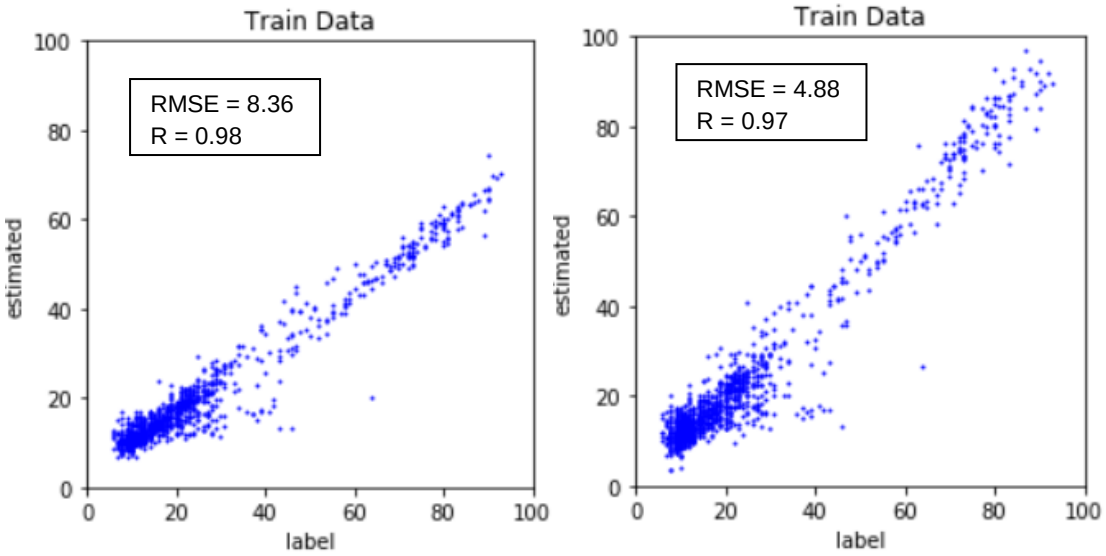

Figure 3.18 Same model without and with dropout when evaluating

3.6 Sample Size

During a deep learning project, there is usually a compromise between efficiency limited by computing power and accuracy limited by the image quality. The models trained by dataset 3 and dataset 4 are investigated here. The model trained by dataset 3 took 2.8 hours while the model trained by dataset 4 took 11.25 hours. The performance of the previous model is obviously better. Although the model trained by dataset 4 might show better performance with longer training time, the sound performance using small data size has shown that the small image already contains all the information needed to estimate brain ages. This point has been supported by [20], using only a few slices to train the model.

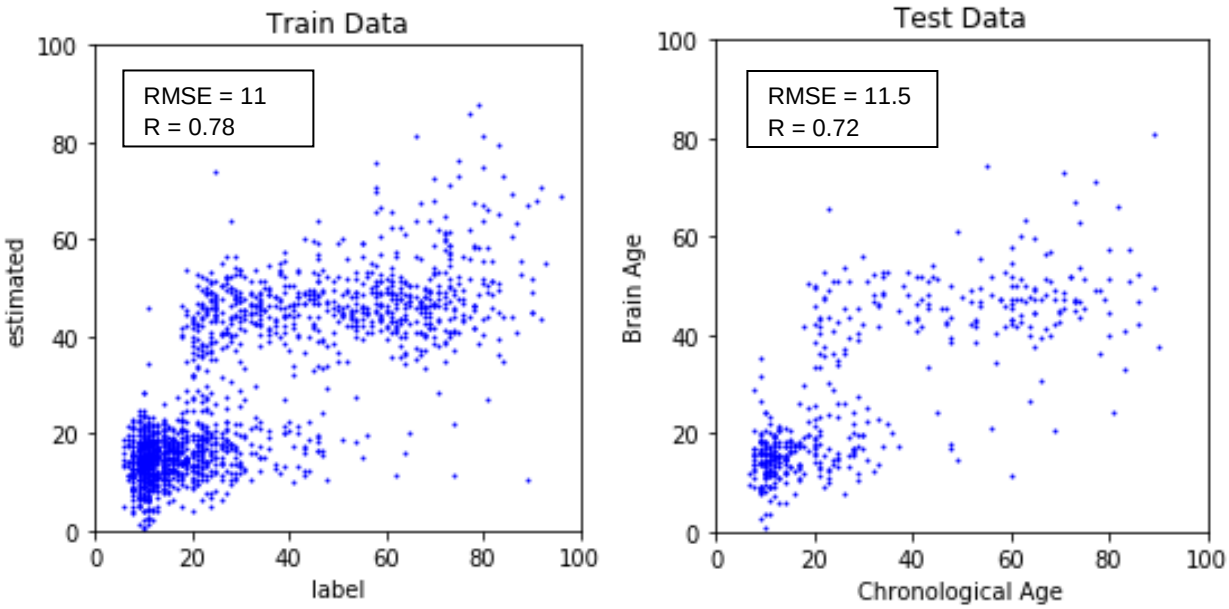

Figure 3.19 Performance of the model trained by dataset 4

3.7 Overfitting Alleviation and Learning Rate

Dropout rate and learning rate are the two most important tunable hyper-parameters in this project. When using a new dataset, optimal values were narrowed down by random search. The method is more efficient than the traditional grid search method [21]. Generally, the learning rate only has a large effect on the first several epochs. This is due to the incorporation of recent gradients and second momentum in the Adam update rule. A suitable dropout rate would avoid a

large degree of overfitting. However, a small possibility value usually means a lower training speed. The optimal values found for each dataset are shown below.

Database	1	2	3	4
Optimal Dropout Rate	0.9	0.9	0.8	0.8
Optimal Learning Rate	0.00005	0.00003	0.0001	0.00008

Table 3.20 A summary of optimal hyper-parameters

3.8 Post Statistical Test

To investigate the effect of sex on brain ages, a two-sample t-test is conducted. The value 'age distance' is defined here as the difference between chronological age and brain age. All age distances are divided into two groups, those of male and those of female. The null hypothesis is that there is no significant difference between the two sets of age distances.

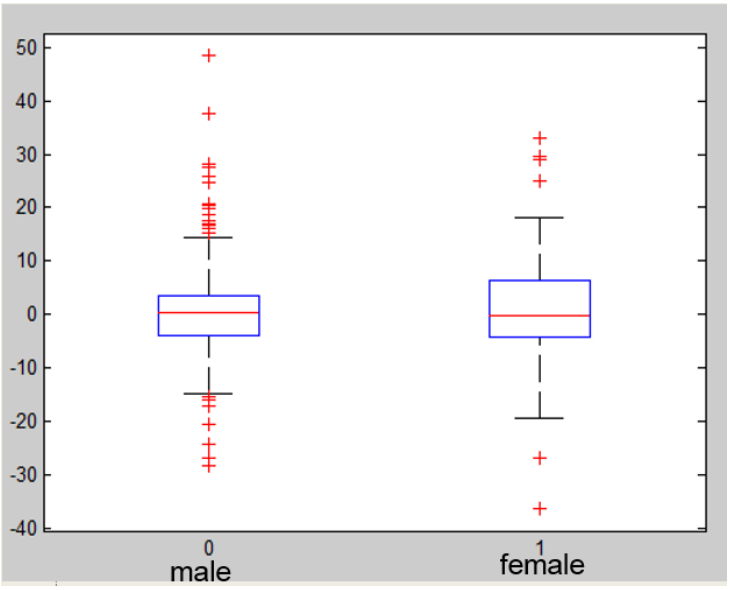

Figure 3.21 Error Bar Plot for the t-test

The p-value of this t-test is 0.9128, which means there is no reason to reject the null hypotheses. Therefore, the sex factor does not contribute significantly to the estimation of brain ages.

4. Discussion

4.1 Rotation and Non-brain Tissue: Robustness of the Model

With the combination of three-dimensional neural networks and linear regression models, chronological ages of healthy individuals were successfully predicted from T1-weighted MRI brain images. The performance of the model depends on the selection of raw dataset, the values of hyper-parameters, and the quality of images. This project is a demonstration of a high training-speed brain age estimation system. The models with the best performance took around 8 hours to train, compared to 83 hours by the previous study [5]. The decrease in training time is favorable for improvement in other aspects, for instance, the between-scanner reliability, optimal hyper-parameters, and CNN structures.

One major objective of neuroimaging research is the application in clinical settings, where time efficiency is a significant factor to be considered. In order to produce accountable data for real-time decision making, a model needs to have the capability to take in data with minimal pre-processing. MRI image pre-processing methods can take more than one day. These methods include bias-field correction, brain segmentation, removal of artifact and motion correction. The effectiveness of these methods is still a current area of research [23] [24]. By using raw images with only mean subtracted, this project demonstrated that this source of discrepancy can be removed. Two limiting factors in deep learning are the computational intensiveness and the superb data quality required. By dismissing the need for long training hours and heavily pre-processed images, this project makes the brain-age estimation model more accessible and portable.

One of the obstacles that clinical neuroimaging applications are facing now is the various settings during acquisition and pre-processing. In order to improve the between-scanner reliability, all images used in this project have been adjusted according to the original pixel dimensions. The test-retest reliability of every model was evaluated by RMSE and Pearson correlation coefficient. However, if the model is to be used in clinical settings, one other thing to be evaluated is the robustness of the model to withstand novel images. The steadiness of the

system in this respect was evaluated by dataset 2 and dataset 3. Dataset 2 is the same with dataset 1 except that images from one of the repositories were rotated. Dataset 3 is the same with dataset 2 except that a repository, whose non-brain tissue has not been removed, is included.

The model trained by dataset 2 had sound performance. Two possibilities exist: either the model has one or multiple nodes to figure out the orientation of image or the model relies on the 'texture' of the brain images instead of the structural information. Unfortunately, as the hidden layers in a deep learning network are usually treated as a 'black box' and the mechanism can only be probed via external information [25], it is unclear which scenario is the actual case here. Either way, the model is adept at investigating brain images with different orientations. Obtaining external information needs feature visualization or another deep learning network for understanding the current neural network, which will be discussed in the future direction part.

It is obvious that the model trained by dataset 3 had a limited performance, both on training data and test data. Furthermore, the distribution of the estimated brain-ages, specifically the 'wall' formed around the average value, advises that the model does not have the capability to process images with non-brain tissue remained. This is also a sound caveat on the unification of preprocessing method. If a similar model is to be used in clinical settings, neuroimaging preprocessing has to be cautiously monitored.

4.2 Dropout: Clues for Brain Age Estimation

Since the invention of deep learning, great effort has been put into understanding the substantial functions of the hidden layers. Specifically for this project, we are interested in what factors the neural network consider, what area is the most important for the decision, and ultimately, how the neural network estimates the brain ages. Due to the absence of theoretical support, however, these questions can only be investigated obliquely using external information. Dropout layers in the model trained by dataset 2 were removed in the testing phase. The outliners were analyzed for common traits.

The model trained by dataset 2 had a good performance. After training, the dropout layers were first kept and the performance was recorded to be RMSE=8.36 and r=0.98 for the training data. After removing the six dropout layers, the performance was recorded as RMSE=4.88 and r=0.97. In traditional deep learning classification problems, dropout layers merely act as an overfitting circumvention tool. However, the model with dropout layers removed had a dropped RMSE in this project. More interestingly, the linear relationship between estimated chronological age and actual chronological remains. The intuitive explanation is that the model has multiple indexes that evaluate the brain ages and the estimate brain age is the sum of these indexes. Therefore, when dropout is removed, around 0.7 of these indexes are removed as well. The edited sum is 0.7 of the original sum. Nonetheless, indirect probing of the problem may not be accurate. Further support is needed from new theories and visualization of the layers.

4.3 Limitations

One limitation of this project is the precision of the chronological age estimates. With the RMSE achieved, the models are not accurate enough to be used in clinical settings. One attempt that may lead to further RMSE decrease is the exploration of new data augmentation methods and more appropriate sample size. The significant role of data quality in deep learning requires more complicated data manipulation. In addition to this, a repository with larger sample size and a unified standard would definitely help to improve the precision.

It should be noted the attempt in this project to demystify brain-ages using dropout is indirect and preliminary. To understand the brain ages in terms of anatomical specificity, further investigations into the feature visualization and extraction are needed. However, the effort to unmask the feature importance in deep neural networks is still underway. Further research is needed to provide theoretical support for the understanding of brain ages. In addition to this, it should be remembered the structural development of brain may not be linear. For instance, the maturation under 18 years old may be dominated by one mechanism and the aging of the brain after that may be dominated by another. Therefore, a brain age estimation model that covers the whole lifespan may not be specific enough to delve into a particular mechanism.

4.4 Future Directions

4.4.1 Feature Visualization and Inversion

In convolutional neural networks, the filters in the convolutional layers perform the convolutional operations. They offer a measure for the similarity between the input and the feature. An intuitive approach towards understanding how the hidden layers work is to visualize the learned weights of these filter as images [26]. By comparing these images, we can get some sense of what these filters are looking for. Specifically for the estimation of brain ages, if most of the filter images are focused on a certain region of the brain, then we know that this region is particularity significant for predicting chronological age based on brain images. However, in reality, visualizing the filters is only a valid approach for the filters in the first layer. The filters in this layer are usually looking for edges, angle, and structural information. The filter images in other layers are usually hard for humans to comprehend.

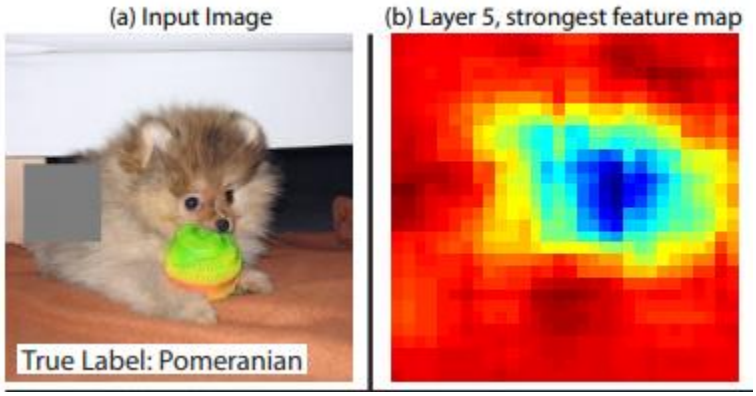

Figure 4.1 Filter Visualization (Retrieved from [27])

Feature inversion is a technique developed specifically for the understanding of convolutional neural networks. Two steps are involved: modeling a representation and calculating the inexact inverse [28]. The technique is different from visualization of filters in that the inversion is an image instead of a feature map. This may be particularly helpful in the estimation of brain ages as the region in brains that is still valid for each layer will be shown. For example, at a certain depth in the convolutional layer, maybe only two structures in the brain are still important. This would give us a large amount of information in deciding the importance of brain regions.

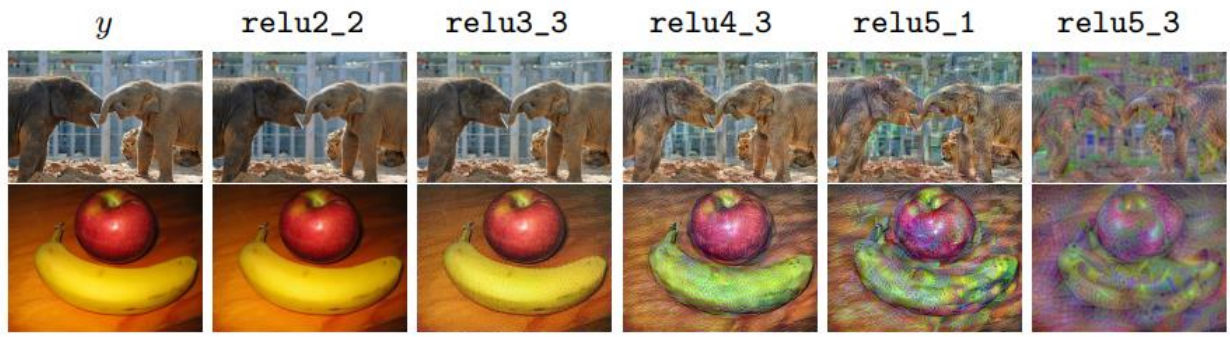

Figure 4.2 Inversion Images in Different Convolutional Layers (Retrieved from [29])

4.4.2 Disease Detection

During data acquisition, images from individuals with brain diseases, such as autism and Alzheimer's disease were excluded. The primary assumption of this project is that the brain age of a healthy individual is close to his or her chronological age, to the extent that individual differences can be neglected. If the brains with known diseases are included, the RMSE would be the only place that the discrepancy will be shown. The integration of various factors would render the separation of disease interference impossible. Furthermore, the extent of diseases, such as different phases of Alzheimer's disease, will be hard to entertain.

After the development of an accurate brain-age estimation system, however, data from individuals with known brain disease can be used in several ways. Firstly, these images can be the input of the already trained deep learning model. If the estimated brain ages for these individuals have a statistically significant deviation from their chronological ages, the model would be an ideal tool for early detection of brain diseases. Secondly, the degree of the disease can be correlated with the degree of deviation in order to deepen our understanding of the disease. Finally, if the deviation is not significant, these images can be used as training data in a new network. The increase in sample size probably will improve the performance of the estimation model.

5. Conclusion

Chronological age of healthy individuals can be predicted from T1-weighted MRI brain images using deep learning with relatively high training efficiency. The resulting brain age estimated by the model using novel data is a valid biomarker. The model is adept at assessing brain images with different orientations but performs poorly when non-brain tissues are not removed. With deep learning visualization and understanding, brain age has the potential to facilitate early detection of brain diseases and to probe the maturation and aging process of brains.

References

[1] Wilson, Robert S., et al. "Cognitive decline in prodromal Alzheimer disease and mild cognitive impairment." *Archives of neurology* 68.3 (2011): 351-356.

[2] Desikan, R. S., et al. "MRI measures of temporoparietal regions show differential rates of atrophy during prodromal AD." *Neurology* 71.11 (2008): 819-825.

[3] Cole, James H., et al. "Increased brain-predicted aging in treated HIV disease." *Neurology* 88.14 (2017): 1349-1357.

[4] Franke, Katja, et al. "Advanced BrainAGE in older adults with type 2 diabetes mellitus." *Frontiers in aging neuroscience* 5 (2013): 90.

[5] Cole, James H., et al. "Predicting brain age with deep learning from raw imaging data results in a reliable and heritable biomarker." *NeuroImage* 163 (2017): 115-124.

[6] Luders, Eileen, Nicolas Cherbuin, and Christian Gaser. "Estimating brain age using high-resolution pattern recognition: younger brains in long-term meditation practitioners." *Neuroimage* 134 (2016): 508-513.

[7] Liu, Xiaoxiao, et al. "Low-rank atlas image analyses in the presence of pathologies." *IEEE transactions on medical imaging* 34.12 (2015): 2583-2591.

[8] Franke, Katja, and Christian Gaser. "Longitudinal Changes in Individual BrainAGE in Healthy Aging, Mild Cognitive Impairment, and Alzheimer's Disease 1Data used in preparation of this article were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (adni. loni. ucla. edu). As such, the investigators within the ADNI contributed to the design and implementation of ADNI and/or provided data but did not participate in analysis or writing of this report. A complete listing of ADNI investigators can be found at: adni ...." *GeroPsych* (2012).

[9] Tipping, Michael E. "Sparse Bayesian learning and the relevance vector machine." *Journal of machine learning research* 1.Jun (2001): 211-244.

[10] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436.

[11] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[12] Lawrence, Steve, et al. "Face recognition: A convolutional neural-network approach." *IEEE transactions on neural networks* 8.1 (1997): 98-113.

[13] Stanford, CS231N: Convolutional Neural Networks for Visual Recognition, spring 2018, Stanford.

[14] Zhou, Weixun, et al. "Learning low dimensional convolutional neural networks for high-resolution remote sensing image retrieval." *Remote Sensing* 9.5 (2017): 489.

[15] Hughes, John R. "Post-tetanic potentiation." *Physiological reviews* 38.1 (1958): 91-113.

[16] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*(2014).

[17] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

[18] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.

[19] Ji, Shuiwang, et al. "3D convolutional neural networks for human action recognition." *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013): 221-231.

[20] Huang, Tzu-Wei, et al. "Age estimation from brain MRI images using deep learning." *Biomedical Imaging (ISBI 2017), 2017 IEEE 14th International Symposium on*. IEEE, 2017.

[21] Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." *Journal of Machine Learning Research* 13.Feb (2012): 281-305.

[22] Chollet, Francois. *Deep learning with Python*. Manning Publications Co., 2017.

[23] Ribeiro, Andre Santos, David J. Nutt, and John McGonigle. "Which Metrics Should Be Used in Non-linear Registration Evaluation?." *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, Cham, 2015.

[24] Valverde, Sergi, et al. "Comparison of 10 brain tissue segmentation methods using revisited IBSR annotations." *Journal of Magnetic Resonance Imaging* 41.1 (2015): 93-101.

[25] Shwartz-Ziv, Ravid, and Naftali Tishby. "Opening the black box of deep neural networks via information." *arXiv preprint arXiv:1703.00810* (2017).

[26] Yosinski, Jason, et al. "Understanding neural networks through deep visualization." *arXiv preprint arXiv:1506.06579*(2015).

[27] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer, Cham, 2014.

[28] Mahendran, Aravindh, and Andrea Vedaldi. "Understanding deep image representations by inverting them." (2015): 5188-5196.

[29] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." *European Conference on Computer Vision*. Springer, Cham, 2016.

Appendix

MRI_train

In [1]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import tensorflow as tf
import numpy as np
import math
import random
import pandas as pd
import csv
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

In [2]:
```
def conv3d(x, W):
    return tf.nn.conv3d(x, W, strides=[1,1,1,1,1], padding='SAME')
```

In [3]:
```
def maxpool3d(x):
    #                      size of window         movement of window
    return tf.nn.max_pool3d(x, ksize=[1,2,2,2,1], strides=[1,2,2,2,1], padding='SAME')
```

In [4]:
```
def convolutional_neural_network(x):
    initializer = tf.contrib.layers.xavier_initializer()
    weights = {'W_conv1':tf.Variable(initializer([3,3,3,1,8]), name="W_conv1"),
               'W_conv2':tf.Variable(initializer([3,3,3,8,16]), name="W_conv2"),
               'W_conv3':tf.Variable(initializer([3,3,3,16,32]), name="W_conv3"),
               'W_conv4':tf.Variable(initializer([3,3,3,32,64]), name="W_conv4"),
               'W_conv5':tf.Variable(initializer([3,3,3,64,128]), name="W_conv5"),
               'W_fc':tf.Variable(initializer([2304,1024]), name="W_fc"),
               'W_out':tf.Variable(initializer([1024, n_classes]), name="W_out")}

    biases = {'b_conv1':tf.Variable(initializer([8]), name="b_conv1"),
              'b_conv2':tf.Variable(initializer([16]), name="b_conv2"),
             'b_conv3':tf.Variable(initializer([32]), name="b_conv3"),
             'b_conv4':tf.Variable(initializer([64]), name="b_conv4"),
             'b_conv5':tf.Variable(initializer([128]), name="b_conv5"),
              'b_fc':tf.Variable(initializer([1024]), name="b_fc"),
              'b_out':tf.Variable(initializer([n_classes]), name="b_out")}

    x = tf.reshape(x, shape=[-1, IMG_SIZE_PX, IMG_SIZE_PX, SLICE_COUNT, 1])

    conv1 = tf.nn.relu(conv3d(x, weights['W_conv1']) + biases['b_conv1'])
    conv1 = tf.nn.dropout(conv1, dropout_rate)
    conv1 = maxpool3d(conv1)

    conv2 = tf.nn.relu(conv3d(conv1, weights['W_conv2']) + biases['b_conv2'])
    conv2 = tf.nn.dropout(conv2, dropout_rate)
    conv2 = maxpool3d(conv2)

    conv3 = tf.nn.relu(conv3d(conv2, weights['W_conv3']) + biases['b_conv3'])
    conv3 = tf.nn.dropout(conv3, dropout_rate)
    conv3 = maxpool3d(conv3)

    conv4 = tf.nn.relu(conv3d(conv3, weights['W_conv4']) + biases['b_conv4'])
    conv4 = tf.nn.dropout(conv4, dropout_rate)
    conv4 = maxpool3d(conv4)

    conv5 = tf.nn.relu(conv3d(conv4, weights['W_conv5']) + biases['b_conv5'])
    conv5 = tf.nn.dropout(conv5, dropout_rate)
    conv5 = maxpool3d(conv5)


    fc = tf.reshape(conv5,[-1, 2304])
    #fc = tf.nn.relu(tf.matmul(fc, weights['W_fc'])+biases['b_fc'])
    fc = tf.matmul(fc, weights['W_fc'])+biases['b_fc']
    fc = tf.nn.dropout(fc, dropout_rate)

    output = tf.add(tf.matmul(fc, weights['W_out']),biases['b_out'],name="output")

    return output[0]
```

In [5]:
```
def train_neural_network(x):


    prediction = convolutional_neural_network(x)
    cost = tf.reduce_mean(tf.square(prediction-y),name="cost")
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
    #accuracy = tf.contrib.metrics.streaming_pearson_correlation(prediction, tf.cast(y,tf.float32))[0]
    saver = tf.train.Saver(save_relative_paths=True)


    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(tf.local_variables_initializer())

        for epoch in range(hm_epochs):
            avg_cost = 0.

            index = [ind for ind in range(1,train_batch+1)]
            original = index[:]
            random.shuffle(index)

            for original_index in range (1, train_batch+1):
                current_batch = index[original_index-1]
                batch_data = np.load(folder + 'batch({},{})-{}-{}-{}.npy'.format(current_batch*batch_size+1,
                                    current_batch*batch_size+batch_size,IMG_SIZE_PX,IMG_SIZE_PX,SLICE_COUNT))

                for data in batch_data:
                    X = data[0]
                    Y = data[1]
                    _, c, p = sess.run([optimizer, cost, prediction], feed_dict={x: X, y: Y})
                    avg_cost += c / (train_batch * batch_size)

            print('Epoch', epoch+1, 'completed out of',hm_epochs,'cost:',avg_cost)

            avg_valicost = 0.
            pred_matrix = np.array([]).reshape(0,1)
            label_matrix = np.array([]).reshape(0,1)
            for current_validation in range (train_batch, train_batch+validation_batch):
                validation_data = np.load(folder + 'batch({},{})-{}-{}-{}.npy'.format(current_validation*batch_size+1,
                                        current_validation*batch_size+batch_size,IMG_SIZE_PX,IMG_SIZE_PX,SLICE_COUNT))
                for data in validation_data:
                    c, predict_value = sess.run([cost, prediction], feed_dict={x:data[0], y:data[1]})
                    predict_value = predict_value.reshape(1,1)
                    avg_valicost += c / (validation_batch * batch_size)
                    label_value = data[1].reshape(1,1)
                    pred_matrix=np.concatenate((pred_matrix, predict_value), axis=0)
                    label_matrix=np.concatenate((label_matrix, label_value), axis=0)

            print("avgvalicost", avg_valicost)
            print(np.concatenate((pred_matrix.T, label_matrix.T),axis=0).shape)
            evaluated_accuracy = np.corrcoef(np.concatenate((pred_matrix.T, label_matrix.T), axis=0))
            print('pearson:',evaluated_accuracy)

            if ((epoch+1)%epoch_save==0):
                filename = '/home/lvruyi/regression_small_epoch_' + str(epoch+1)
                saver.save(sess, filename)
                test_data = np.load(folder + 'batch(1473,1482)-65-65-55.npy')
                #test_data = np.load(folder + 'batch(2065,2072)-65-65-55.npy')
                for tdata in test_data:
                    label_value = tdata[1]
                    predicted_value = sess.run(prediction, feed_dict={x: tdata[0], y:tdata[1]})
                    print ("label value:", label_value, ";    estimated value:", predicted_value)
```

In [ ]:
```
IMG_SIZE_PX = 65
SLICE_COUNT = 55
n_classes = 1
#train_batch = 15
train_batch = 75
batch_size = 16
#validation_batch = 4
validation_batch = 17
labels_file = pd.read_csv('FYP_Phenotypic2noxrotate.csv')

tf.reset_default_graph()
x = tf.placeholder('float',name="x")
y = tf.placeholder('float',name="y")

learning_rate = 0.00007
dropout_rate = 0.8
hm_epochs = 20000
epoch_save= 5
#"combined2\\"
folder = '/home/lvruyi/combined2noxrotate/'

#calculate_mean()`

#shuffle()

"""
for batch in range (0, train_batch+validation_batch-1):
    combine_preprocess(batch*batch_size+1, batch*batch_size+batch_size)

combine_preprocess(2065, 2072)
"""

train_neural_network(x)
```

MRI_restore

In [1]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import tensorflow as tf
import numpy as np
import math
import random
import matplotlib.pyplot as plt
import pandas as pd
import csv
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

In [2]:
```
def conv3d(x, W):
    return tf.nn.conv3d(x, W, strides=[1,1,1,1,1], padding='SAME')
```

In [3]:
```
def maxpool3d(x):
    #                        size of window         movement of window
    return tf.nn.max_pool3d(x, ksize=[1,2,2,2,1], strides=[1,2,2,2,1], padding='SAME')
```

In [4]:
```
def train_neural_network(x):

    with tf.Session() as sess:
        meta_path = '/home/lvruyi/noxrot_small_epoch_520/regression_small_epoch_520.meta'
        checkpoint_path = '/home/lvruyi/noxrot_small_epoch_520'
        saver1 = tf.train.import_meta_graph(meta_path)
        #saver1= tf.train.Saver(tf.global_variables())
        #saver1= tf.train.Saver(tf.local_variables())
        saver1.restore(sess, tf.train.latest_checkpoint(checkpoint_path))
        #tf.reset_default_graph()

        graph = tf.get_default_graph()

        x = graph.get_tensor_by_name("x:0")
        y = graph.get_tensor_by_name("y:0")
        cost = graph.get_tensor_by_name("cost:0")


        prediction = graph.get_tensor_by_name("output:0")


        """
        #for removing dropout
        weights = {'W_conv1':graph.get_tensor_by_name("W_conv1:0"),
               'W_conv2':graph.get_tensor_by_name("W_conv2:0"),
               'W_conv3':graph.get_tensor_by_name("W_conv3:0"),
               'W_conv4':graph.get_tensor_by_name("W_conv4:0"),
               'W_conv5':graph.get_tensor_by_name("W_conv5:0"),
               'W_fc':graph.get_tensor_by_name("W_fc:0"),
               'W_out':graph.get_tensor_by_name("W_out:0")}

        biases = {'b_conv1':graph.get_tensor_by_name("b_conv1:0"),
               'b_conv2':graph.get_tensor_by_name("b_conv2:0"),
              'b_conv3':graph.get_tensor_by_name("b_conv3:0"),
              'b_conv4':graph.get_tensor_by_name("b_conv4:0"),
              'b_conv5':graph.get_tensor_by_name("b_conv5:0"),
               'b_fc':graph.get_tensor_by_name("b_fc:0"),
               'b_out':graph.get_tensor_by_name("b_out:0")}


        conv1 = tf.nn.relu(conv3d(tf.reshape(x, shape=[-1, IMG_SIZE_PX, IMG_SIZE_PX, SLICE_COUNT, 1]), weights['W_conv1']) + bias
        conv1 = maxpool3d(conv1)

        conv2 = tf.nn.relu(conv3d(conv1, weights['W_conv2']) + biases['b_conv2'])
        conv2 = maxpool3d(conv2)

        conv3 = tf.nn.relu(conv3d(conv2, weights['W_conv3']) + biases['b_conv3'])
        conv3 = maxpool3d(conv3)

        conv4 = tf.nn.relu(conv3d(conv3, weights['W_conv4']) + biases['b_conv4'])
        conv4 = maxpool3d(conv4)

        conv5 = tf.nn.relu(conv3d(conv4, weights['W_conv5']) + biases['b_conv5'])
        conv5 = maxpool3d(conv5)


        fc = tf.reshape(conv5,[-1, 2304])
        fc = tf.matmul(fc, weights['W_fc'])+biases['b_fc']

        prediction = tf.add(tf.matmul(fc, weights['W_out']),biases['b_out'],name="output")
        cost = tf.reduce_mean(tf.square(prediction-y),name="cost")
        """
```

```
Y = []
X = []
ii = 1
for current_batch in range (0, train_batch):
        batch_data = np.load(folder + 'batch({},{})-{}-{}-{}.npy'.format(current_batch*batch_size+1,current_batch*batch_s
        data_i = current_batch*batch_size+1
        for data in batch_data:
            predict_value = sess.run(prediction, feed_dict={x:[data[0]], y:[data[1]]})[0][0]
            label_value = data[1][0]
            selected = labels_file[labels_file.shuffledID == data_i]
            for index, row in selected.iterrows():
                cohort = row['COHORT']
            if predict_value>0 and predict_value<100:
                Y.append(predict_value)
                X.append(label_value)
                print (ii, "-----label value:", label_value, ";    estimated value:", predict_value)
                ii = ii +1
            data_i = data_i +1


plt.title('Train Data')
plt.xlabel('label')
plt.ylabel('estimated')
plt.xlim(0, 100)
plt.ylim(0, 100)
for index in range(len(X)):
    plt.scatter(X[index], Y[index], c = 'blue', s=1)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()

avg_cost = 0.
pred_matrix = np.array([]).reshape(0,1)
label_matrix = np.array([]).reshape(0,1)
for current_batch in range (0, train_batch):
    batch_data = np.load(folder + 'batch({},{})-{}-{}-{}.npy'.format(current_batch*batch_size+1,current_batch*batch_size+
    data_i = current_batch*batch_size+1
    for data in batch_data:
            c, predict_value = sess.run([cost, prediction], feed_dict={x:data[0], y:data[1]})
            predict_value=predict_value.reshape(1,1)
            label_value = data[1].reshape(1,1)
            selected = labels_file[labels_file.shuffledID == data_i]
            for index, row in selected.iterrows():
                cohort = row['COHORT']
            if predict_value>0 and predict_value<100 and cohort!='IXI':
                avg_cost += c / (train_batch * batch_size)
                pred_matrix=np.concatenate((pred_matrix, predict_value), axis=0)
                label_matrix=np.concatenate((label_matrix, label_value), axis=0)
                print ("label value:", label_value, ";    estimated value:", predict_value, "avgcost", avg_cost)
            data_i = data_i +1

    pred_matrix=np.concatenate((pred_matrix, predict_value), axis=0)
    label_matrix=np.concatenate((label_matrix, label_value), axis=0)

print(np.concatenate((pred_matrix.T, label_matrix.T),axis=0).shape)
evaluated_accuracy = np.corrcoef(np.concatenate((pred_matrix.T, label_matrix.T), axis=0))
print('pearson:',evaluated_accuracy)
```

```
Y = []
X = []
for current_validation in range (train_batch, train_batch+validation_batch):
        validation_data = np.load(folder + 'batch({},{})-{}-{}-{}.npy'.format(current_validation*batch_size+1,
                          current_validation*batch_size+batch_size,IMG_SIZE_PX,IMG_SIZE_PX,SLICE_COUNT))
        data_i = current_validation*batch_size+1
        for data in validation_data:
            predict_value = sess.run(prediction, feed_dict={x:data[0], y:data[1]})[0][0]
            label_value = data[1][0]
            selected = labels_file[labels_file.shuffledID == data_i]
            for index, row in selected.iterrows():
                cohort = row['COHORT']
            if predict_value>0 and predict_value<100:
                Y.append(predict_value)
                X.append(label_value)
                print ("label value:", label_value, ";    estimated value:", predict_value)
            data_i = data_i +1

plt.title('Test Data')
plt.xlabel('Chronological Age')
plt.ylabel('Brain Age')
plt.xlim(0, 100)
plt.ylim(0, 100)
for ii in range(len(X)):
    plt.scatter(X[ii], Y[ii], c = 'blue',s=1)
plt.gca().set_aspect('equal', adjustable='box')
#plt.draw()
```

```
        avg_cost = 0.
        pred_matrix = np.array([]).reshape(0,1)
        label_matrix = np.array([]).reshape(0,1)
        for current_validation in range (train_batch, train_batch+validation_batch):
            validation_data = np.load(folder + 'batch({},{})-{}-{}-{}.npy'.format(current_validation*batch_size+1,
                                              current_validation*batch_size+batch_size,IMG_SIZE_PX,IMG_SIZE_PX,SLICE_COUNT))
            data_i = current_validation*batch_size+1
            for data in validation_data:
                    c, predict_value = sess.run([cost, prediction], feed_dict={x:data[0], y:data[1]})
                    predict_value=predict_value.reshape(1,1)
                    label_value = data[1].reshape(1,1)
                    selected = labels_file[labels_file.shuffledID == data_i]
                    for index, row in selected.iterrows():
                        cohort = row['COHORT']
                    if predict_value>0 and predict_value<100 and cohort!='IXI':
                        avg_cost += c / (validation_batch * batch_size)
                        pred_matrix=np.concatenate((pred_matrix, predict_value), axis=0)
                        label_matrix=np.concatenate((label_matrix, label_value), axis=0)
                        print ("label value:", label_value, ";    estimated value:", predict_value, "avgcost", avg_cost)
                    data_i = data_i +1

            pred_matrix=np.concatenate((pred_matrix, predict_value), axis=0)
            label_matrix=np.concatenate((label_matrix, label_value), axis=0)

        print(np.concatenate((pred_matrix.T, label_matrix.T),axis=0).shape)
        evaluated_accuracy = np.corrcoef(np.concatenate((pred_matrix.T, label_matrix.T), axis=0))
        print('pearson:',evaluated_accuracy)
        """
        """
```

In [5]:

```
IMG_SIZE_PX = 65
SLICE_COUNT = 55
n_classes = 1
#train_batch = 15
train_batch = 75
#train_batch = 100
batch_size = 16
#validation_batch = 4
validation_batch = 17
#validation_batch = 29
labels_file = pd.read_csv('FYP_Phenotypic2noxrotate.csv')

x = tf.placeholder('float')
y = tf.placeholder('float')

learning_rate = 0.000001
dropout_rate = 1
epoch_save= 20
#"combined2\\"
folder = '/home/lvruyi/combined2noxrotate/'
hm_epochs = 20000

#calculate_mean()`

#shuffle()

"""
for batch in range (0, train_batch+validation_batch-1):
    combine_preprocess(batch*batch_size+1, batch*batch_size+batch_size)

combine_preprocess(2065, 2072)
"""

train_neural_network(x)
```

def show_slices

```
def show_slices(slices):
        """ Function to display row of image slices """
        fig, axes = plt.subplots(1, len(slices))
        for i, slice in enumerate(slices):
            axes[i].imshow(slice.T, cmap="gray", origin="lower")
```

def rot90

```
def rot90(m, k=1, axis=2):
    """Rotate an array by 90 degrees in the counter-clockwise
      direction around the given axis"""
    m = np.swapaxes(m, 2, axis)
    m = np.rot90(m, k)
    m = np.swapaxes(m, 2, axis)
    return m
```

Example of visualization

```
first = np.load('data2\\85#(65, 65, 55).npy')
second = np.load('shuffled2\\45#(65, 65, 55).npy')

print(first.shape)
show_slices([
              first[int(first.shape[0]/2), :, :],
                 first[:, int(first.shape[1]/2), :],
              first[:, :, int(first.shape[2]/2)]])
plt.show()
```

```
show_slices([second[int(second.shape[0]/2), :, :],
                 second[:, int(second.shape[1]/2), :],
                 second[:, :, int(second.shape[2]/2)]])
plt.show()
```

def loadData

```
def loadData(sub_ID):

    try:
        training_img = nib.load(INPUT_FOLDER + "\\" + str(sub_ID) + "\\PROCESSED\\
                MPRAGE\\SUBJ_111\\" + str(sub_ID) + "_mpr_n4_anon_sbj_111.img")
    except:
        training_img = nib.load(INPUT_FOLDER + "\\" + str(sub_ID) + "\\PROCESSED\\
                MPRAGE\\SUBJ_111\\" + str(sub_ID) + "_mpr_n3_anon_sbj_111.img")

    training_pixdim = training_img.header['pixdim'][1:4]

    training_data = training_img.get_data()

    return training_data, training_pixdim
```

def saveDataset

```
def saveDataset(dataset, name):
    np.save("data\\" + str(name), dataset)
```

def resample

```
def resample(image, pixdim, new_spacing=[1,1,1]):
    # Determine current pixel spacing
    spacing = pixdim

    resize_factor = spacing / new_spacing
    new_real_shape = image.shape * resize_factor
    new_shape = np.round(new_real_shape)
    real_resize_factor = new_shape / image.shape
    new_spacing = spacing / real_resize_factor

    image = scipy.ndimage.interpolation.zoom(image, real_resize_factor,
                                             mode='nearest')

    return image
```

Example of preliminary preprocessing

```
INPUT_FOLDER = 'disc1'
COHORT = 'OASIS'
SITE_ID = ' '

labels_file = pd.read_csv('FYP_Phenotypic.csv')
print(labels_file[labels_file.COHORT == COHORT])

selected_rows = labels_file[labels_file.COHORT == COHORT]


for index, row in selected_rows.iterrows():
    print ("Saving data id: " , row['ID'])
    osingle_data, single_dim = loadData(row['SUB_ID'])

    single_data = osingle_data.reshape(256,256,160)

    single_resampled = resample(single_data, single_dim, [2,2,2])

    DESIRED_SHAPE = (130, 130, 110)
    X_before = int((DESIRED_SHAPE[0]-single_resampled.shape[0])/2)
    Y_before = int((DESIRED_SHAPE[1]-single_resampled.shape[1])/2)
    Z_before = int((DESIRED_SHAPE[2]-single_resampled.shape[2])/2)

    npad = ((X_before, DESIRED_SHAPE[0]-single_resampled.shape[0]-X_before),
            (Y_before, DESIRED_SHAPE[1]-single_resampled.shape[1]-Y_before),
            (Z_before, DESIRED_SHAPE[2]-single_resampled.shape[2]-Z_before))
    single_padded = np.pad(single_resampled, pad_width=npad, mode='constant',
                           constant_values=0)

    saveDataset(single_padded, str(row['ID']) + "#" + str(single_padded.shape))

    print(str(osingle_data.shape) + "  --->   " + str(single_data.shape) +
          "  --->   " + str(single_resampled.shape) + "  --->   "
          + str(single_padded.shape))
```

def rotate_oasis

```
def rotate_oasis():
    for x in range (1, 317):
        img_data = np.load("data2\\" + str(x) + "#(" + str(IMG_SIZE_PX)+
                           ", " + str(IMG_SIZE_PX) + ", " +
                           str(SLICE_COUNT)+ ").npy")
        img_data = rot90(img_data, 3, 0)
        img_data = rot90(img_data, 1, 2)
        print("img" + str(x) + " rotated: ")
        np.save("data2\\" + str(x) + "#(" + str(IMG_SIZE_PX)+ ", " +
                str(IMG_SIZE_PX) + ", " +str(SLICE_COUNT)+ ").npy", img_data)
```

def shape_oasis

```
def shape_oasis():
    for x in range (1, 317):
        img_data = np.load("data2\\" + str(x) + "#(" + str(IMG_SIZE_PX)+
                           ", " + str(IMG_SIZE_PX) + ", " +
                           str(SLICE_COUNT)+ ").npy")
        npad = ((5, 5), (0, 0), (0, 0))
        img_data = np.pad(img_data, pad_width=npad,
                          mode='constant', constant_values=0)

        startz = 65//2-(55//2)
        img_data = img_data[0:65,0:65, startz:startz+55]
        print("img" + str(x) + " shaped: ")
        np.save("data2\\" + str(x) + "#(" + str(IMG_SIZE_PX)+ ", " +
                str(IMG_SIZE_PX) + ", " +str(SLICE_COUNT)+ ").npy", img_data)
```

def calculate_mean

```
def calculate_mean():

    only_img = []

    for x in range (1, 1201):
        img_data = np.load("shuffled2\\" + str(x) + "#(" +
                           str(IMG_SIZE_PX)+ ", " + str(IMG_SIZE_PX)
                           + ", " +str(SLICE_COUNT)+ ").npy")

        print("img" + str(x) + " appended: ")
        only_img.append(img_data)

    mean_img = np.mean(only_img, dtype=np.int, axis=0)
    np.save('mean_img2.npy', mean_img)
```

def shuffle

```
def shuffle():
    index = [i for j in (range(1,317), range(907, 2073)) for i in j]
    #index = [x for x in range(1,2073)]
    original = index[:]

    for n,i in enumerate(index):
        if i>906:
            index[n]=index[n]-590
    temp = index[:]
    random.shuffle(index)

    print(original)
    print("-------------------------")
    print(temp)
    print("-------------------------")
    print(index)
    csvfile = "index_reference2.csv"


    with open(csvfile, "w", newline='') as output:

        writer = csv.writer(output)
        for val in index:
            writer.writerows([[val]])

    for n,i in enumerate(original):
        img_data = np.load("data2\\" + str(i) + "#(" +
                           str(IMG_SIZE_PX)+ ", " + str(IMG_SIZE_PX)
                           + ", " +str(SLICE_COUNT)+ ").npy")
        new_index = index[n]
        np.save("shuffled2\\" + str(new_index) + "#(" +
                str(IMG_SIZE_PX)+ ", " + str(IMG_SIZE_PX)
                + ", " +str(SLICE_COUNT)+ ").npy", img_data)
        print ("saved " + str(i))
```

def combined_preprocess

```
def combine_preprocess (start, end):

    combined = []
    mean_image = np.load('mean_img2.npy')

    for x in range (start, end+1):
        img_data = np.load("shuffled2\\" + str(x) + "#("
                           + str(IMG_SIZE_PX)+ ", " + str(IMG_SIZE_PX)
                           + ", " +str(SLICE_COUNT)+ ").npy")
        selected = labels_file[labels_file.shuffledID == x]

        for index, row in selected.iterrows():
            print (str(row['ID']) + " 's age is " + str(row['AGE_AT_SCAN']))
            age = row['AGE_AT_SCAN']

        label = np.array([age])

        img_data -= mean_image

        combined.append([img_data,label])

    np.save('combined2\\' + 'batch({},{})-{}-{}-{}.npy'.format
            (start,end,IMG_SIZE_PX,IMG_SIZE_PX,SLICE_COUNT), combined)
```

Example of preprocess, shuffle and combine data

```
rotate_oasis ()

shape_oasis()

shuffle()

calculate_mean()


for batch in range (0, train_batch+validation_batch-1):
    combine_preprocess(batch*batch_size+1, batch*batch_size+batch_size)

combine_preprocess(1473, 1482)
```

FYP Report

ORIGINALITY REPORT

2%	2%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	James H. Cole, Rudra P.K. Poudel, Dimosthenis Tsagkrasoulis, Matthan W.A. Caan, Claire Steves, Tim D. Spector, Giovanni Montana. "Predicting brain age with deep learning from raw imaging data results in a reliable and heritable biomarker", NeuroImage, 2017 Publication	1%
2	www.ncbi.nlm.nih.gov Internet Source	1%
3	arxiv.org Internet Source	1%

Exclude quotes	Off	Exclude matches	< 1%
Exclude bibliography	Off		