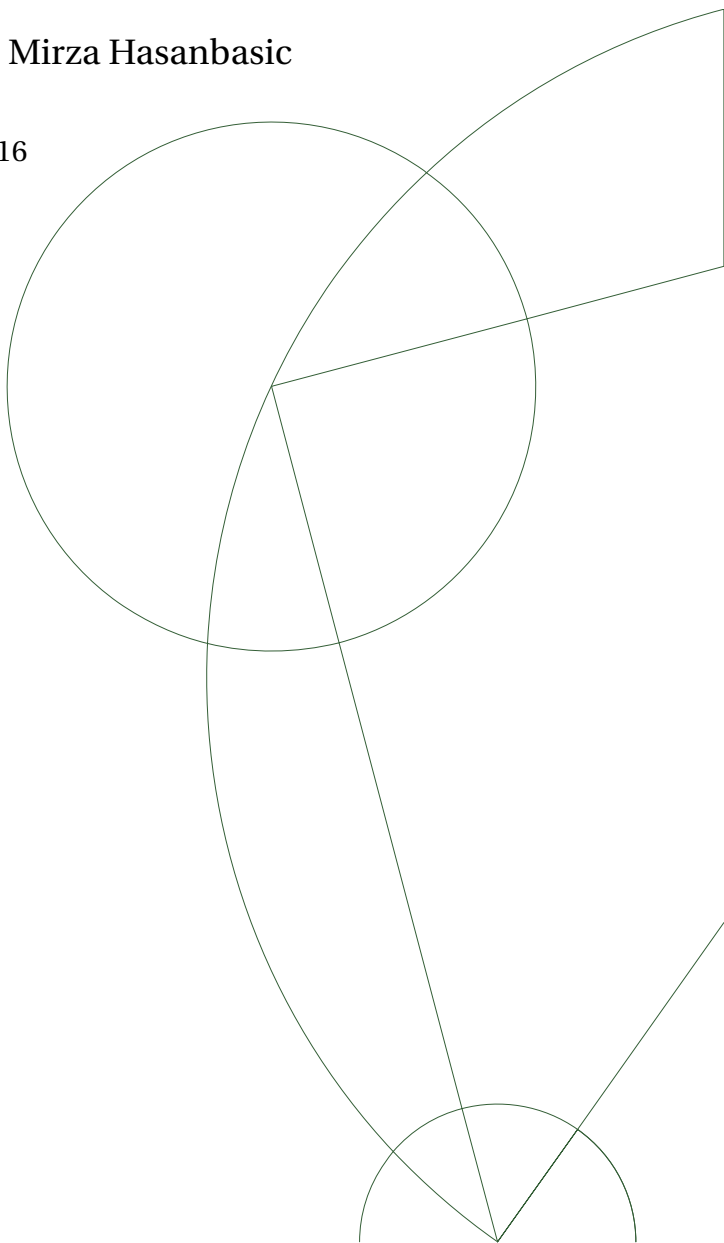
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

MR Image texture analysis applied to the diagnosis of Alzheimer's Disease

Mathias Bjørn Jørgensen & Mirza Hasanbasic

10th June 2016



Contents

List of Figures

List of Tables

List of Corrections

Mathias Bjørn Jørgensen & Mirza Hasanbasic

Abstract

This report will examine MRI scans of brains, using image texture analysis and machine learning

Chapter 1

Introduction

Alzherimer's Disease (AD) is the most common cause of dementia among people and is a growing problem in the aging populations. It has a big impact on health services and society as life expectancy increases. In 2010 the total global costs of dementia was estimated to be about 1% of the worldwide gross domestic product[1]. AD is the cause in about 60%-70% of all cases of dementia[1] and about 70% of the risk is believed to be genetic [5]. Currently there are no way to cure dementia or to alter the progressive course. But however, much can be done to support and improve the lives if AD is diagnosed in the early stage of progression [1]

In this report we will examine MRI data of the hippocampus using image texture analysis and apply machine learning in order to diagnose AD in patients. Our dataset contain 100 patients 50 control and 50 with AD.

We will be using two different image texture analysis method, one which will me in 2D[7][4] and the other one will be in 3D[9], from which we calculate the data to the gray level co-occurrence matrix (GLCM).

1.1 Problem Definition

Is it possible to classify MRI data of the hippocampus into groups of healthy controls vs Alzheimer's patient, using a predefined set of image texture metrices, with an accuracy greater than 80%?

Is there a difference in diagnosing AD successfully by calculating the co-occurence matrix in 3D compared to 2D.

[1] With the terms as of direct medical costs, direct social costs and costs of informal care

Chapter 2

Data

2.1 ADNI data

The data in this study was provided already downloaded and preprocessed. It had been previously obtained from the ADNI database (adni.loni.usc.edu). The ADNI was launched in 2003 by the National Institute on Aging, the National Institute of Biomedical Imaging and Bioengineering, the Food and Drug Administration, private pharmaceutical companies, and nonprofit organizations, as a $60 million, 5-year, public–private partnership. The primary goal of ADNI has been to test whether serial MRI, positron emission tomography (PET), biological markers, and clinical and neuro-psychological assessments can be combined to measure the progression of MCI and early AD. Determination of sensitive and specific markers of very early AD progression is intended to aid researchers and clinicians to develop new treatments and monitor their effectiveness, as well as to lessen the time and cost of clinical trials. ADNI is the result of efforts of many co-investigators from a broad range of academic institutions and private corporations, and subjects have been recruited from over 50 sites across the U.S. and Canada. For up-to-date information, see http://www.adni-info.org/ We were provided with a random subset (50 controls, 50 AD) of the "complete annual year 2 visits" 1.5T dataset from the collection of standardized datasets released by ADNI http://www.adni.loni.usc.edu/methods/mri-analysis/adni-standardized-data/ [11]. The complete dataset (504 subjects) comprised one associated 1.5T T1-weighted MRI image out of the two possible from the back-to-back scanning protocol in ADNI [8] at baseline, 12-month follow-up, and 24-month follow-up.

2.2 Preprocessing

The data were provided for use already preprocessed. This preprocessing, and subsequent hippocampal segmentation was performed with the freely available FreeSurfer software package (version 5.1.0) [6] using the cross-sectional pipeline with default parameters. The original MRI image resolution of [0.94, 1.35] x [0.94, 1.35] x 1.2mm was conformed to a 1.0 x 1.0 x 1.0 mm resolution, and all MRIs were bias field corrected. The bias field correction in FreeSurfer utilizes the nonparametric nonuniform intensity normalization algorithm [10], often

referred to as N3. The input data for this study was therefore the corrected T1-weighted MRI image volume, and corresponding separate binary masks of left and right hippocampi.

Chapter 3

Method

3.1 Image texture analysis methods

Image texture is a feature that can help to segment images into regions of interest and to classify those regions. Textures gives us some information about the arrangement of the intensities in an image. Texture analysis is a technique for evaluating the position and intensity of signal features[4]. Statistical texture analysis methods evaluate the interrelationship of voxels, based on mathematical parameters computed from the distribution and intensities of voxels in the image.

3.1.1 Co-occurrence matrix

The co-occurence matrix (COM) is second-order statistics methods, which is based on information about colours in pair of pixels. The matrix is defined over the image with distribution values at a given offset. Mathematically we have a COM matrix **C** which is defined over an $n \times m$ image **I**, with $\Delta x, \Delta y$ being the parameterized offset, is calculated by

$$C_{\Delta x,\Delta y}(i,j) = \Sigma_{p=1}^{n}\Sigma_{q=1}^{m} \begin{cases} 1, & \text{if } I(p,q) = i \text{ and } I(p+\Delta x, q+\Delta y) = j \\ 0, & \text{otherwise} \end{cases}$$

FiXme Note: find reference

FiXme Note: find reference

The element (5,4) in the COM can be translated to meaning how many times there exist an element in the image with GI **FiXme Note: level or intensity?** 5 and another element offset $\Delta x, \Delta y$ from the originial with greyscale intensity[1] (GI) 4, i.e. if the offset is (0,1)**FiXme Note: rigtigt offset?** and the first element is (x,y)(4,3) with GI 5 it would mean that element (x,y)(5,3) would have GI 4. If COM(4,4) is ten, it translates into there being ten instances with element (x,y) = 5 and (x+Δx,y+Δy) = 4. COMs calculated on GIs are often called gray-level co-occurence matrices (GLCM).

FiXme Note: level or intensity?

FiXme Note: rigtigt offset?

[1] The pixel value is a single number that represents the brightness of the pixel. Typically one is taken to be black and 256 is taken to be white and values in between make up the different shades of grey

A single image have multiple GLCMs as different offsets creates different relations. Consider a 3 × 3 matrix looking at element (2,2) we can then create eight different offsets, (1,0),(1,1), (0,1)),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1) **FiXme Note: ændre til** $COM_{(0,1)}$... **osv**, however they are not unique. **FiXme Note: Vise for transpose GLCM**

FiXme Note: ændre til $COM_{(0,1)}$... osv
FiXme Note: Vise for transpose GLCM

Focusing on the two offsets (0,1), (0,-1) in element (2,2) and (1,2) with GI 1 and 2 respectfully increases the entry $GLCMs_{1,0}(1,2)$ and $GLCMs_(-1,0)$ (1,1) with one, showing that $GLCMs^{\mathbf{T}}_{(0,1)}$ = $GLCMs_{(0,-1)}$. There exist the same relation between (1,1)(-1,-1), (0,1)(0,-1), and (-1,1)(1,-1). This leaves four different offsets for analysis (0,1),(-1,1), (-1,0)),(-1,-1) in general (0,d),(-d,d),(-d,0),(-d,-d) where d is the distance which are commonly named angles 0°, 45°, 90° and 135° as seen in figure 3.1.

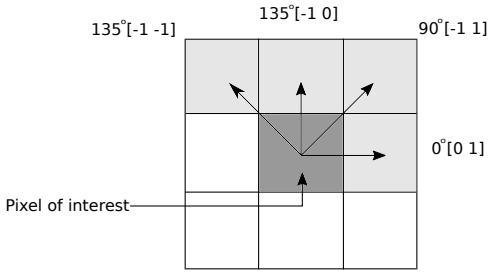


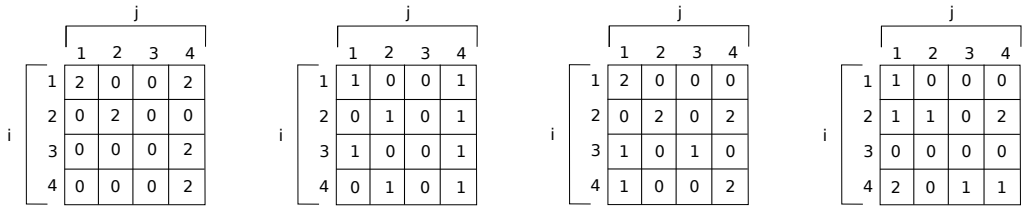
Figure 3.1: Example of the offsets for the 2D

and let figure 3.2 illustrate this concept with a 4 × 4 image I with four different COM's for $I : C_{(0,1)}, C_{(-1,1)}, C_{(-1,0)}$ and $C_{(-1,-1)}$

I

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

Figure 3.2: Image I that is 4-by-4



(a) $C_{(0,1)}$

i \ j	1	2	3	4
1	2	0	0	2
2	0	2	0	0
3	0	0	0	2
4	0	0	0	2

(b) $C_{(-1,1)}$

i \ j	1	2	3	4
1	1	0	0	1
2	0	1	0	1
3	1	0	0	1
4	0	1	0	1

(c) $C_{(-1,1)}$

i \ j	1	2	3	4
1	2	0	0	0
2	0	2	0	2
3	1	0	1	0
4	1	0	0	2

(d) $C_{(-1,-1)}$

i \ j	1	2	3	4
1	1	0	0	0
2	1	1	0	2
3	0	0	0	0
4	2	0	1	1

Figure 3.3: Four different COM's for a gray-tone image. Shows how the GLCM are calculated of the 4-by-4 image I 3.2.

The co-occurrence matrix is quadratic with the number of rows and columns equal to the amount of GI, for example if we have 256 GI we get a 256 × 256 GLCM.

Extending this method to three-dimensions it is necessary to look on how the offsets are defined because the size of the GLCM is defined by the amount of GIs and not by the images it is derived from. Considering a 3 × 3 × 3 matrix we have a possible of 26 offsets. In two-dimensions it is possible to eliminate half of the offsets because of the relation $\text{GLCM}^{\mathbf{T}}_{d,d} = \text{GLCM}_{-d,-d}$, and it is the same case in three-dimensions with the relation being $\text{GLCM}^{\mathbf{T}}_{d,d,d} = \text{GLCM}_{-d,-d,-d}$. This leaves 13 offsets which are illustrated below.

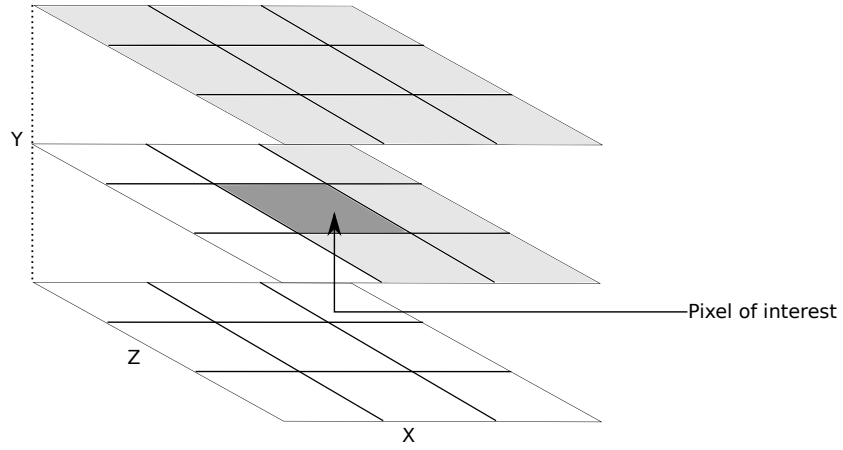


Figure 3.4: Example of the offsets for the 3D

To create a two-dimensional GLCM on three-dimensional image we create slices through the image. Given a $n \times m \times l$, n = 20, m = 20 l = 20, image we can create a slice for each n. Slice_1 is equal to the matrix M($n = 1 \times m \times l$), Slice_2 = M($n = 2 \times m \times l$), and so on, giving a total of 20 $m \times l$ images instead. It is possible for each slice to calcualte the $\text{GLCM}_{d,d}$ for each slice, and we define $\text{GLCM}_{d,d} = \Sigma^2_{n=1}0\ \text{GLCM}_{d,d}(\text{Slice}_n)$

These slices are done in all three directions of the image for the four difference offsets resulting in 12 GLCMs. There is an overlap between six of the GLCMs. Slicing through n axis with offset (0,d) is equal to slicing through the m axis with the same offset. Slicing through the m axis with offset (-d,d) is equal to slicing through the l axis with offset (-d,0). Slicing through the l axis with offset (-d,d) is equal to slicing through the n axis with offset (d,0), which is the transposed offset of (-d,0). Removing the duplicates leaves nine GLCMs which we calculate at distances d = (1,2,...,10), giving a total of 90 GLCMs for each mri scan.

The three-dimensional versions is also calculated for distances d = (1,2,...,10) resulting in 130 GLCMs for each mri scan.

3.1.2 Texture features from co-occurrence matrix

To compare the differences between the GLCMS 13 different features are computed. They are the same as used in [7] except for one difference. The difference is how the correlation is calculated as $\sum_{i,j} \frac{(i-\mu_i)(j-\mu_j)glcm(i,j)}{(\sigma_i \cdot \sigma_j)}$, where μ_i, μ_j, σ_i, σ_j are the means and standard deviations of C_i and C_j respectively. This correlation is called the Pearson product moment correlation coefficient and it determines how correlated a pixel is to its neighbour, over the

whole image[2][3]. We calculate two versions of these features, one where we normalized the COMs and one where we do not.

3.2 Machine learning

Machine learning is a method used to create complex models and algorithms that lend themselves to prediction, when the models are exposed to new data that should be able to teach themselves to grow and change. There are several different categories of machine learning, one of them being supervised learning. In supervised laerning given a large sample of input and output pairings, the goal is to find the complex function that maps that relation between input and output. With a sufficient large dataset it would be possible to train a supervised algorithm to predict output values for some new input values that it have not seen before.

3.2.1 10-fold cross-validation

Given a model with unknown paramers and a training set to which the model can be fit then the fitting process optimizes the models parameters to fit the training data. Validating this model against independent data (test data) from the same data pool as the training, it will generally turn out that the model does not fit the test data as well as the training data. It is known as overfitting and is a problem when the size of the training data set is small. Cross-validation is used to counteract overfitting.

Dividing the entire data set into 10 groups at random, one subsample is saved for testing and the reamining nine is used to fit the model. The procedures is done so all subsamples get to be used to validate exactly one time and the validation result can then be averaged to produce a single estimation. This solves the problem of overfitting, as the validation data set is never used in to fit the model. **FiXme Note: how does this remove overfitting?**

FiXme Note: how does this remove overfitting?

3.2.2 Feature selection

Feature selection is the process of selection af subset of relevant features for use in model contruction. The main goal of feature selection is to choose a subset of the entire set of input features so the subset can predict the output with an accuracy comparable to using the entire set to predict the output, but with a large reduction in the computational cost. When a dataset contains many feautres that are either redundant og irrelevant it is possible to remove them without incurring much loss of information. Features may be redundant due to the pressence of another feature with which it is strongly correlated, while they may be very informative for the model their information have already been provided by a different feature. Only choosing a subset of the possible features have the added advantage of decreasing the complexity of the model.

Sequential forward feature selection(SFS) is a greedy algorrithm to choose features. It starts off with finding the best possible single feature to describe the model. Given the feature set of that single feature, the next step is to find which other feature would improve the predictiveness of the model and then add that to the set of selected featuers. It continues to grow

the set of selected features, until the goal have been reached. The goal can either be a specific amount of features, a specific accuracy for the model or it can stop if all choices of a new feature would decrease the accuracy. A problem with SFS is due to its greedy nature, there is no guarantee that the first feature selected is part of the optimal solution. Given three features X_1, X_2 and X_3 where X_1 is the best single feature, X_2 is second best and X_3 the worst, it does not necessitate that pairing X_1, X_2 is better than X_2, X_3, nor does it secure any other relation, meaning that the SFS does not guarantee the optimal solution. Which leads to one of the drawbacks of this feature selection, if a feature is selected it is not possible to exclude it later even if it would increase the evaluation score to do so.

```
1  The algorithm for SFS with 10-fold cross validation
2  Step 1:
3  Set selected features Y = \emptyset
4  X = entire featuerset
5  Seperate dataset into 10 folds of equal size with 5 of each class, {K1,K2,...,K10}.
6
7  Step 2:
8  For feature i= 1 to No. of features
9    for fold j=1 to No. of folds
10     Train a k-nn model on {Y,Xi} using fold K1,2,...,10\ Kj as training
11     Calcualte missclassification error of model on Kj
12 Average the error for all 10 folds.
13
14 Step 3:
15 F = feature with lowest error
16 X = X \ F
17 Y = Y ∪ F
18
19 Step 4:
20 Continue step 2-3 until the missclassification error worsen or 15 features have been
       ↪ selected.
```

The algorithm is run on 10 different k values for the k-nn model, k = 1,2,...,10.

3.2.2.1 Naive

3.2.3 K Nearest Neighbors algorithm

The K Nearest Neighbor (KNN) is a methoed that is used for classification and regression. It should be noted, that KNN is a non parametric lazy learning algorithm, which means that it does note make any assumptions on the underlying data distribution. In other words, it means that the training phase is fast and KNN keeps all the training data. It should be noted that KNN makes decision based on the entire training data set and in the KNN an object is classified by majority vote of its neighbors, with the object being assigned to the class most common amongst its *KNN*'s. 3.5
Since the training phase is minimal, then it should be noted that the testing phase is very costly for KNN in both memory and time and often it can be a worst case for time needed, since all points might take point in the decision making.

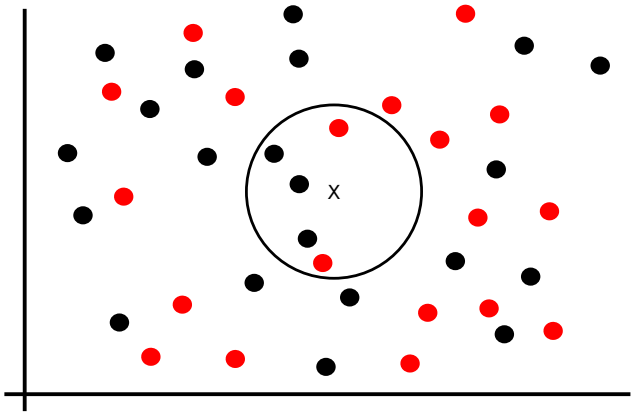


Figure 3.5: The KNN grows a spherical region until it encloses k training samples, and labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point $\mathbf{x}$ would be labelled the category of the black points

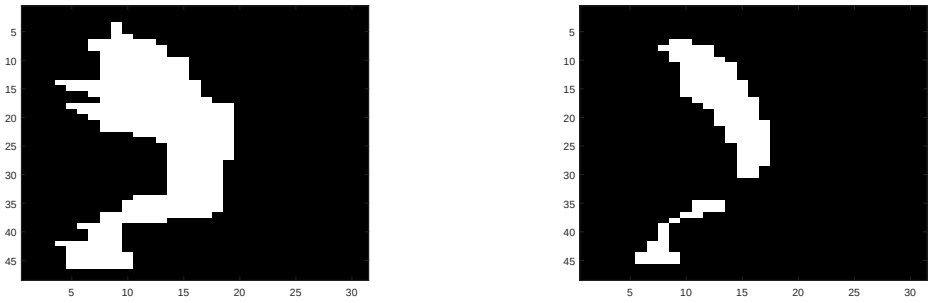
Because the KNN classifer predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Image that we have a large scale variabels, they will have a much larger effect on the distance between observations and hence the KNN classifier, than variables on a small scale. Often a good way to handle this problem is to standardize the data.

The disadvantage of KNN is that choosing a k may be tricky, so we are left to test the algorithm on multiple k's and often it needs a large number of samples for better accuracy.

Typically will the KNN classifier be based on a distance, commonly it is based on the Euclidean distance between a test sample and the specified training samples. Let x_i be an input sample with p features $(x_{i1}, x_{i2}, ..., x_{ip})$, and n be the total number of input samples $i = 1, 2, ..., n$ and p the total number of features $j = 1, 2, ..., p$. The Euclidean distance between sample x_i and x_l, $l = 1, 2, ..., n$ is defined as $d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + ... + (x_{ip} - x_{lp})^2}$

3.3 Erode

Each patients hippocampus has been segmented in the MRI scan. The problems we can run into are that the background will blur with the segmentation i.e. the hippocampus. Performing a erosion can solve this problem and we can focus on the hippocampus, with the maximum number of details. As seen in figure 3.6a the erosion has not been performed yet, but we might have some problems with data surrounding the hippocampus is blurring out the edges of the hippocampus. To solve this, we create a mask to separate the hippocampus from the background data and end up with what is left in figure 3.6b



(a) Note eroded

(b) After erosion has been performed

Figure 3.6: Hippocampus at slice 10 on the X-axis as bitwise

The erosion we have used is called a city-block metric and can be seen in figure 3.7.

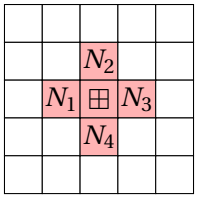
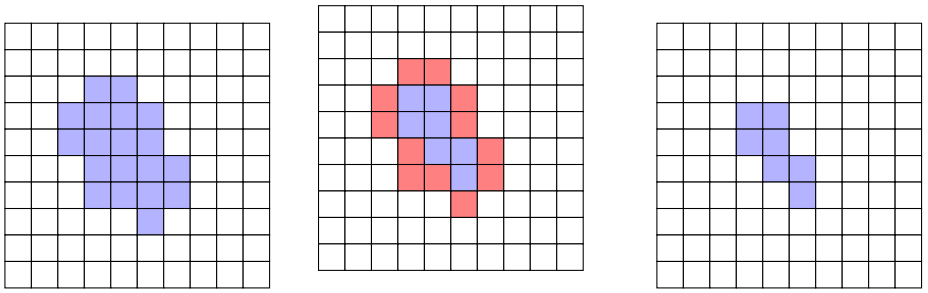


Figure 3.7: Text

To give an example of how the erosion works, it will be illustrated and we will use the city-block metric for this purpose. So we will use figure 3.7 and erode the image in figure 3.8.



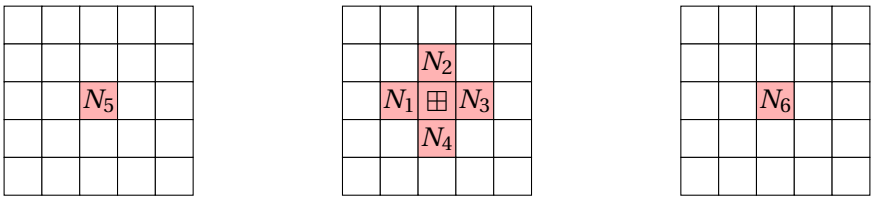
(a) Original image

(b) Performing erode on the image

(c) The image after erosion

Figure 3.8: Example of a city-block erosion with before and after

As seen in figure 3.8 the noise (background) have been removed. This is an example in 2D. Now we wish to extend the erosion city-block to 3D. As seen in figure 3.7 it have 4 neighbours and when we extend this to 3D we will end up with 6 neighbours instead as seen in figure 3.9 and the concept is still the same as in 2D



(a) From above (b) From the front (c) From below

Figure 3.9: 2D example of the 3D city-block

Chapter 4

Implementation

4.1 Preparation of Data

The MRI files are 256 × 256 × 256 matrices but we are only interested in the small part which overlaps with the masks from segmentation, i.e. where the elements in segmentation is either 1 (left hippocampus) or 2 (right hippocampus). We have created the function HippoMatrix, which takes three variables, which file to load, wether or not erosion should be performed, and if the left or right hippocampus is desired. First we assign a value based on if we are looking for the right hippocampus, as they are associeated with 1 and 2 respectfully. If erosion is desired we create the city-block for erosion by taking advantage of distances calculations, distance = $\sqrt{x^2 + y^2 + z^2}$. All parts of the city-block have distance 1 from the origin point. With the city-block defined we can use MATLABs build in function imerode to perform the erosion. **FiXme Fatal: insert snip of code.**

FiXme Fatal: insert snip of code.

Looping through the entire segmentation matrix we identify all the datapoints where segmentation is one for the left hippocampus or two if we are trying to identify the right hippocampus. For each instance in segmentation we save the coordinate (i,j,k) and the mri(i,j,k) value in an array as v(1) = (i_1,j_1,k_1, MRI(i_1,j_1,k_1)).**FiXme Fatal: snip of main loop, explain left is also used incase of right**

FiXme Fatal: snip of main loop, explain left is also used incase of right

On the basis of this we can create a three-dimensional matrix which contains all the datapoints, hippoBox = max(i) - min(i) + 1 × max(j) - min(j) +1 × maj(k) - min(k) + 1. Then we simply loop through our array with the relevant data and input them into hippoBox, all other elements inside the matrix are set to NaN. **FiXme Fatal: snip of last loop**

FiXme Fatal: snip of last loop

The return value from the function is the matrix hippoBox containing only the relevant data.

4.2 Calculating GLCMs

To calculate the GLCM's in two-dimensions we have taken advantage of MATLABs built-in function graycomatrix. It calculates as described in methods. It is then a matter of giving the proper offsets, and the right number of GIs. We can then loop through the hippoBox slices and sum up the GLCMs.**FiXme Fatal: snip af glcm2dallangels hvor vi udregner**

FiXme Fatal: snip af glcm2dallangels hvor vi udregner

We ultimately save all 90 glcm's in a cell.**FiXme Fatal: Fodnote der beksriver hvad en cell er?**

FiXme Fatal: Fodnote der beksriver hvad en cell er?

To implement the three-dimensional GLCM's we have created our own function.

4.3 Calculating(Computing) the GLCM Features

In the implementation of the GLCM Feature derivation we are taking two inputs. The first input variable is the GLCM matrix and the second is wether we wish to normalize the data.

What we are doing first is to make sure that all variables are implemented. Firstly we find the size of the GLCM which will be the greylevels. Hereafter we can initiate the C_x, C_y, C_{x+y} and C_{x-y} since we know the size of the GLCM.

For the pixel values in the GLCM we are using MATLAB's `ind2sub` function, that is a command that determines the equivalent subscript values corresponding to a single index into an array.**FiXme Note: Lav et eksempel (diagram) af hvordan cxminusy og cxplusy ser ud**. We are using these variables in the GLCM Features as seen in Appendix A.

FiXme Note: Lav et eksempel (diagram) af hvordan cxminusy og cxplusy ser ud

To calculate the C_{x+y} and C_{x-y} we have two for loops as seen in Appendix A.3 and A.4 where N of course is the greylevels.

To find the mean and standard deviation for C_x and C_y we just use the functions that MATLAB have.

The GLCM features, as seen in Appendix A, utilizes MATLABs use of vectorization. This is rewarding in the vectorized code appears more like the mathematical expressions and makes the code easier to understand and is shorter. There is often a performance gain in using vectorized code than the corresponding code containing loops.

It should be obvious for the reader to tell that the code looks alot like the mathematical expression like in Appendix A.

```
1  HXY1 = -nansum(glcm(tmpsub)'.*log(cX(I).*cY(J)));
2  HXY2 = -nansum(cX(I).*cY(J).*log(cX(I).*cY(J)));
3  HX   = -nansum(cX.*log(cX));
4  HY   = -nansum(cY.*log(cY));
5  HXY  = -nansum(glcm(:).*log(glcm(:)));
6
7  stats.angularSecondMoment                = sum(glcm(:).^2);
8  stats.contrast                           = sum(abs(I-J).^2.*glcm(tmpsub));
9  stats.correlation                        = (sum(I.*J.*glcm(tmpsub)) - muX*muY) ./ (stdX
       ↪ *stdY);
10 stats.variance                           = sum(((I - mean(glcm(:))).^2).*glcm(tmpsub));
11 stats.inverseDifferenceMoment            = sum(glcm(tmpsub)./(1 + (I-J).^2));
12 stats.sumAverage                         = sum(bsxfun(@times,(2:2*nGrayLevels)',cXplusY
       ↪ ));
13 stats.sumVariance                        = sum(((2:2*nGrayLevels) - stats.sumAverage)
       ↪ '.^2.*cXplusY((2:2*nGrayLevels)-1,1));
14 stats.sumEntropy                         = nansum(cXplusY.*log(cXplusY));
15 stats.entropy                            = HXY;
16 stats.differenceVariance                 = var(cXminusY);
17 stats.differenceEntropy                  = nansum(cXminusY.*log(cXminusY));
18 stats.informationMeasuresOfCorrelation1  = (HXY - HXY1)./(max(HX,HY));
```

Chapter 5

Result

Chapter 6

Discussion

Chapter 7

Conclusion

Appendices

Appendix A

Co occurrence matrix derivation features

$$C_x(i) = \sum_{j=1}^{N} C(i,j) \tag{A.1}$$

$$C_y(i) = \sum_{i=1}^{N} C(i,j) \tag{A.2}$$

$$C_{x+y}(k) = \sum_{\substack{i=1 \\ i+j=k}}^{N} \sum_{j=1}^{N}, \quad \text{k = 2, 3, ... , 2N} \tag{A.3}$$

$$C_{x+y}(k) = \sum_{\substack{i=1 \\ |i-j|=k}}^{N} \sum_{j=1}^{N}, \quad \text{k=0,1,...,N-1} \tag{A.4}$$

$$f_1 = \sum_{i=1}^{N} \sum_{j=1}^{N} \{C(i,j)\}^2 \tag{A.5}$$

$$f_2 = \sum_{n=0}^{N-1} n^2 \left\{C_{x+y}(k)\right\} \tag{A.6}$$

$$f_3 = \frac{\sum_{i=1}^{N} \sum_{j=1}^{n} i\,j\,C(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{A.7}$$

$$f_4 = \sum_{i=1}^{N} \sum_{j=1}^{N} (i-\mu)^2 C(i,j) \tag{A.8}$$

$$f_5 = \sum_{i=1}^{N} \sum_{j=1}^{n} \frac{1}{1+(i-j)^2} C(i,j) \tag{A.9}$$

$$f_6 = \sum_{i=2}^{2N} i C_{x+y}(i) \tag{A.10}$$

$$f_7 = \sum_{i=2}^{2N} (i-f_6)^2 C_{x+y}(i) \tag{A.11}$$

$$f_8 = \sum_{i=2}^{2N} C_{x+y}(i) \log(C_{x+y}(i)) \tag{A.12}$$

$$f_9 = -\sum_{i=1}^{N} \sum_{j=1}^{N} C(i,j) \log(C(i,j)) \tag{A.13}$$

$$f_{10} = \text{variance of } C_{x-y} \tag{A.14}$$

$$f_{11} = -\sum_{i=0}^{N-1} C_{x-y}(i) \log(C_{x-y}(i)) \tag{A.15}$$

Bibliography

[1] Dementia. http://www.who.int/mediacentre/factsheets/fs362/en/. Accessed: 2016 April.

[2] Mathworks graycopros. http://se.mathworks.com/help/images/ref/graycoprops.html. Accessed: 2016 May.

[3] Pearson product. https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient. Accessed: 2016 May.

[4] G Castellano, L Bonilha, LM Li, and F Cendes. Texture analysis of medical images. *Clinical radiology*, 59(12):1061–1069, 2004.

[5] Anne Corbett Carol Brayne Dag Aarsland Emma Jones Clive Ballard, Serge Gauthier. Alzheimer's disease. *Lancet*, 377:1019–1031, March 2011.

[6] Bruce Fischl, David H Salat, Evelina Busa, Marilyn Albert, Megan Dieterich, Christian Haselgrove, Andre Van Der Kouwe, Ron Killiany, David Kennedy, Shuna Klaveness, et al. Whole brain segmentation: automated labeling of neuroanatomical structures in the human brain. *Neuron*, 33(3):341–355, 2002.

[7] Peter A Freeborough and Nick C Fox. Mr image texture analysis applied to the diagnosis and tracking of alzheimer's disease. *Medical Imaging, IEEE Transactions on*, 17(3):475–478, 1998.

[8] Clifford R Jack, Matt A Bernstein, Nick C Fox, Paul Thompson, Gene Alexander, Danielle Harvey, Bret Borowski, Paula J Britson, Jennifer L Whitwell, Chadwick Ward, et al. The alzheimer's disease neuroimaging initiative (adni): Mri methods. *Journal of Magnetic Resonance Imaging*, 27(4):685–691, 2008.

[9] Rouzbeh Maani, Yee Hong Yang, and Sanjay Kalra. Voxel-based texture analysis of the brain. *PloS one*, 10(3):e0117759, 2015.

[10] John G Sled, Alex P Zijdenbos, and Alan C Evans. A nonparametric method for automatic correction of intensity nonuniformity in mri data. *Medical Imaging, IEEE Transactions on*, 17(1):87–97, 1998.

[11] Bradley T Wyman, Danielle J Harvey, Karen Crawford, Matt A Bernstein, Owen Carmichael, Patricia E Cole, Paul K Crane, Charles DeCarli, Nick C Fox, Jeffrey L Gunter, et al. Standardization of analysis sets for reporting results from adni mri data. *Alzheimer's & Dementia*, 9(3):332–337, 2013.