Erik Mason (erikmason04@gmail.com)

Started January 9th, 2018

**Create\_Database\_Table: Documentation and Notes**

General Notes

* The script must be run from inside the DAMM folder:
  + C:\Alzheimer's Project Code Base - Github Master Updated\Polarization-Analysis
* **The Polarization Analysis GUI must be open as well**. This is necessary because the script takes data from the GUI metadata files, which are saved as a special object that requires access to the class definition files to open. The only way I could find to read the metadata files properly was to have the GUI open in the same MATLAB session.
* The function does not have inputs or outputs, so it will run like a script (by pressing ‘Run’ in the Editor tab at the top of the MATLAB interface, or by pressing F5 while looking at the function code).
* The script will read in the previous database table, which should be in the filepath listed above. It will move this old database table into a folder called “Old Database Tables”, and save a new one that includes the date of generation in the filename.

User Inputs

In the script, after the initial comments, there is a section for user inputs:

* database\_path: all the potential options are there. Just umcomment (click on the line and press CNTRL + T) the database you wish to create a table for, and make sure all the others are commented (click on the line and press CNTRL + R)
  + Currently (Jan 9th 2018) the code has been focused for the human database since its creation. At some point the dog database worked as well, and the other databases haven’t been tested.
* label\_empty: set to 1 if you wish to create a table where empty *categorical type* values are labelled with the string “Empty”, instead of an empty categorical value in MATLAB. In this way, missing data will show up as a separate field called “Empty” instead of being ignored and thus excluded from any analysis. Default is 0.
* recalculate\_all: set to 1 if you wish to recalculate all of the polarization metric stats for the entire database. Useful if there’s a change in the way the metric stats are calculated, and everything needs to be calculated again. If it is set to 0, the code will still calculate the metric stats for new deposits that don’t have them calculated already. The rest of the metric stats will be copied in from the previous database table. Default is 0.

**Important Notes on Sub-Functions, and Assumptions in the Code**

1.) create\_pathnames\_structure.m

Creates a nested structure data type in MATLAB, that contains filepaths for all of the metadata files and images that exist in the database. The saved variable is called ‘dbpaths’, for database paths. It is structured in the same way as the database folders themselves – organized into subject, eye, quarters, locations, and different image types. Only complete filepaths are saved in the structure, so that an imread( ) or load( ) can be called directly on any element.

The variable dbpaths is then passed on to the rest of the scripts, to be looped through in order to read all of the metadata and images.

Assumptions/Things to Note

* If a path is written into dbpaths, then the corresponding file or image exists. This means that if you expect 16 intensity images, but only 14 are in the folder, then there will only be 14 image paths saved in the corresponding spot in dbpaths. This means that the 6060 intensity image may not always be saved in index 16, if an image in the middle is missing. However, this has not been a problem yet (except for unregistered microscope images, in which case that location cannot be registered or analysed anyway), and so far if any images are present then they all are.
* If a folder is empty, or metadata/images are missing, then the corresponding spot(s) in the structure is filled with an empty value. An empty folder corresponds to an empty location in the structure, with no other structure fields inside. For example if an eye folder exists, but has nothing in it, then there will be an empty value in place of an eye structure. It will not have location folder structures that have empty metadata and image paths.
* Within a location folder, there are assumptions in how the code chooses the correct sessions:
  + The code always looks for the folder “CS 001 (Microscope)” for the microscope and fluorescent images, and nothing else.
  + For registered images, it looks for the most recent registration session – for example, between “PS 001 (Legacy Registration)” and “PS 003 (Legacy Registration)”, it will choose the latter. This assumes that the most recently created (uploaded) registration session is the correct one to use.
  + For polarization analysis folders, it is a bit more complicated:
    - If there is only one registration session (“PS 001 (Legacy Registration)” and no others), then the code will choose the polarization analysis folder that has the lowest number – for example, between “PS 004 (Polarization Analysis - v1)”, “PS 005 (Polarization Analysis - v1)”, and “PS 006 (Polarization Analysis - v1)”, it will choose the first. It is rare that there is more than one polarization analysis folder per registration session, but it is due to the use of subsections earlier in the data collection. Basically, certain cropped areas were processed and saved in separate folders than the full image. The polarization metrics for the full image are always saved in the first Polarization Analysis folder.
    - If there is more than one registration session, then the code uses the most recent polarization analysis folder – for “PS 00x (Polarization Analysis - v1)“, the code will choose the folder with the highest x. This is based off of the assumption that the most current registration session is the correct one (as stated above), and also that subsections are not used anymore. Then, the most recent polarization analysis folder will be the full images for the most recent registration session. To-date (Jan 9th 2018) only one subject has locations with more than one registration session (VA 14-91, or subject 8). In the future when/if subjects are re-registered, subsections will not be used.

2.) create\_metadata\_table.m

Loops through the database folders (using dbpaths) to extract information from the database metadata files, and saves it into a table variable called ‘dbt’ for database table. Examples of the data saved are: patient age, diagnoses, who performed the dissection, fluoro and crossed signal, deposit location, any notes entered into the GUI, etc. One row is given to each location that is imaged, and the subject/eye/quarter/location indices are included for reference.

Further calculations are performed on the collected metadata, before the table is passed into the next script. The calculations performed are:

* labelling the first new subject, new eye, or new quarter in the table, so they can be easily extracted to create a table with only one row per subject (pre-saved as dbt\_s), one row per eye (dbt\_e), or one row per quarter (dbt\_q) instead of one row per location. This allows for analysis (for example tabulation) on metadata that exists only once per subject/eye/quarter, without counting each value once for each location. For example the quarter type for each quarter, eye source for each eye, or patient gender of each subject.
* finding the number of locations in each quarter; locations and quarters in each eye; and locations, quarters and eyes in each subject
* reordering the table columns for better structuring
* converting data types to categorical.

Any metadata that is in the database table is added in this script.

Assumptions/Things to Note

* Three data types are used in the table:
  + double (aka numerical or continuous). This is used for metadata that would typically be classified as continuous, like patient age and deposit location. It is also used for other values added to the table like subject/eye/quarter/location indices, number of locations in each eye, etc. Not all metadata that is double type in the metadata files will be double type in the table, for example InitialFixativePercent. Metadata like that, which was found to only have a discrete set of values, is converted to categorical type. These choices are made on a case-by-case basis; the data type of a column in dbt can be checked by clicking the drop-down arrow on the right side of the column name (when you hover over it), when dbt is open in the Variables tab in MATLAB.
  + logical (aka Boolean). This is used only for values that are calculated after collecting metadata. In particular the logical-type columns are IsNewSubject, IsNewEye, IsNewQuarter, IsRegistered, IsSegmented, and IsProcessed. All of these values are explicitly used to index the table, so logical type is the best choice. Any metadata of logical type is converted to categorical type, as they are also used for analysis. This choice was made so that any analysis variables are either categorical or double type.
  + categorical. This is used for any metadata that has a discrete set of values, whether they are ordered or not. Examples are patient age, diagnoses, quarter type, who dissected the retina, etc. Values that are logical in the metadata, like SessionRejected, or CrossedSignature, are converted to categorical in the table (with labels ‘0’ and ‘1’, as strings). Also, notes are converted to categorical as well, just for consistency and so that a string type doesn’t have to exist in the table. Also, in the future there may be standard notes that are used, like ‘wrong layer’, which will need to be analysed like categorical values. MATLAB has a datatype for categorical variables, which allows for easy tabulation using the tabulate( ) function, inclusion in box plots using the boxplot( ) function, and use in functions like gscatter( ).
* If metadata is missing, it is given a MATLAB standard empty value corresponding to its data type. These are NaN for double type and <undefined> for categorical type. If label\_empty is set to 1, empty categorical variables will be filled with the word “Empty” instead of an <undefined> category. Double variables will stay NaN. In most analysis or plotting functions, MATLAB does a good job of automatically excluding data points with these standard missing values.
* Subject, eye, quarter and location indices are defined by their respective position in the database folders, NOT what the name of the folder is. For example, if an eye folder has Quarter 2, Quarter 3, and Quarter 4 folders, the corresponding quarter indices will be 1, 2, and 3. Similarly, the locations will be given indices from 1 to the number of location folders that are there, regardless of what the locations are named in the folder names. This is because the indices are generated from detecting these folders and are used to iterate through them.
* Subject, eye, quarter and location metadata files are assumed to exist, and an error will be thrown if one is missing. The ‘session’ metadata files (microscope, registration, and polarization) are checked for existence. To date (Jan 11th, 2018) this has not been a problem, as it is very rare to expect missing metadata files. The fact that any are checked for existence was done because of one missing metadata file in one subject.
* Before being put into dbt, all metadata is passed through the function filter\_data, which requires the classification of the data as continuous (double type) or nominal (categorical). These classifications were chosen manually, based off of the observed behaviour of the metadata. For example, the metadata field ‘InitialFixativePercent’ is double type in the metadata files, and is technically continuous. However once collected and tabulated, it was noticed that 96.77 % of the values are 10 %, and the remainder are 4 %. So ‘InitialFixativePercent’ is served better as a categorical variable, with options 10 or 4.
* Some metadata elements are collected in a try block. This is because these elements are saved as a structure inside the metadata, and the required data itself is a field of that structure. For example, the patient gender is saved in the metadata file as a structure called ‘gender’, with the string for ‘Male’ or ‘Female’ being saved as gender.displayString. Normally if a metadata field doesn’t exist, and it’s assigned to a new variable, the new variable will take an empty value []. However if the code attempts to access gender.displayString, and the gender structure doesn’t exist, it will throw an error. So, a try block is used, where the catch fills an empty data value into dbt.
* The code is written to only save two diagnoses from the metadata. No more than two have been seen to date (Jan 11th 2018), indicating that it’s unlikely to see more. If there are more, the code will save the first two and give a message.
* The ‘AD\_Positive’ column of dbt is not taken from the metadata, but created based on the diagnoses. If there is an explicit diagnosis of ‘Alzheimer's Disease Negative’ for either of the two diagnoses, then AD\_Positive = 0. If there is an explicit ‘Alzheimer's Disease Postive’ diagnosis, and no other diagnoses, then AD\_Positive = 1. If there is another diagnosis along with the positive Alzheimer’s disease diagnosis, then AD\_Positive = 2. If none of these conditions are met, then AD\_Positive = -1 (so there are two empty diagnosis fields, or two non-AD diagnoses).
* The NumberOfEyesInSubject column of dbt is calculated as the number of eye folders in a subject folder that are not empty. Explicitly, the eye folder must have at least one quarter folder inside it. This distinction is made because there was an eye folder with nothing in it (except a metadata file), which was counted as an imaged eye and threw off deposit counts (when adjusting for the number of eyes imaged). The same is true for the NumberOfQuartersInEye; there must be at least one location folder in a quarter folder for it to be counted. NumberOfLocationsInQuarter is different – it is simply the number of location folders inside each quarter folder. The location folders themselves can still be empty (unexpected, because when the folder is generated it is also filled with images) and they will still be counted.
* After saving subject, eye, and quarter metadata into dbt for a given location, all of that metadata is copied into the next row in the table. This is because of the structure of dbt, where each imaged location is given its own row. This structure was chosen so that all metadata and location information is in easily accessible columns that can be plotted directly, instead of being tied up in a nested structure variable like dbpaths. This means that things like the patient gender and age, for each subject, are copied into each row of the table for every location in that subject. When the next location is in a new subject, this is fine because the new metadata is saved over it in the next loop of the code (through all of the subjects). Any empty metadata is explicitly written into dbt as an empty value, so there shouldn’t be any metadata from the previous location remaining where it shouldn’t be. Once all locations are looped through, there is an extra row that’s been copied from the last location; the extra row is deleted.
* The RadialDistFromFovea column is calculated as the Euclidian sum of the x and y location coordinates (), which are defined from -1 to 1 with the point (0, 0) at the center.
* The IsRegistered column is a Boolean variable that is set to 1 if the registration metadata file exists, and 0 if it does not. This assumes that if the metadata file exists, then the registered images exist as well. This is not necessarily true. It just means that at some time, someone uploaded registered images for that location through the GUI. The images may have been deleted after (it shouldn’t happen though). Similarly, IsProcessed is 1 if the polarization metadata file exists (in the folder named something like ‘PS 002 (Polarization Analysis - v1)’), and 0 if it doesn’t. The specific registration and polarization folders that the code checks (if there are more than one) is outlined in the Assumptions/Things to Note section of the notes for create\_pathnames\_structure.m.
* IsNewSubject, IsNewEye, and IsNewQuarter are 1 if the subject, eye, and quarter index are different than the one that follows (or, a lower index changes if the higher one does. So, if one row is subject 4 eye 1, and the next row is subject 5 eye 1, IsNewEye will be 1 because the subject index changed). So they select the first location in each subject, eye, and quarter respectively. To note, the metadata specific to the location (Deposit, IsRegistered, XCoord, YCoord, etc) will remain in the tables containing one row per subject (dbt\_s), eye (dbt\_e), and quarter (dbt\_q) – this is simply the metadata for the first location in the subject, eye, or quarter. It should not be relevant, and will likely be excluded in further edits to the code.
* The number of crossed signal, fluoro signal, and deposits in each quarter are counted as the number of locations (each folder with its own metadata is considered a separate location, regardless of if there are multiple images taken of the same deposit. Hopefully this will be fixed later) that have ‘crossedSignature’, ‘fluoroSignature’, and ‘deposit’ = 1 in the metadata files respectively, and also have ‘rejected’ = 0. This means the counts in dbt do not count rejected deposits. The ‘deposit’ Boolean is taken from the metadata file in the location folder, and ‘crossedSignature’, ‘fluoroSignature’, and ‘rejected’ are taken from the microscope metadata file, in the folder ‘CS 001 (Microscope)’. NumberOfLocationsInQuarter is simply the number of location folders in the quarter. NumberOf\_\_InEye columns are the sum of the corresponding NumberOf\_\_InQuarter values for each quarter in the eye. For example, NumberOfCrossedInEye is the sum of NumberOfCrossedInQuarter for each of the quarters in the given eye. The same is true one level up – NumberOf\_\_InSubject values the sum of the corresponding NumberOf\_\_InEye, for each eye in the subject (most have only one anyway). This is true for NumberOfQuartersInSubject as well, which couldn’t be calculated directly from counting folders.
* The columns of dbt are reordered so that they are not stuck in the order that they were calculated or taken from the metadata. Of note:
  + The subject, eye, quarter, and location indices are moved to the first four columns of the table, so they can be seen together and the location can be found easily in the database.
  + The columns IsNewSubject, IsNewEye, IsNewQuarter, IsRegistered, and IsProcessed are the first columns before the rest of the subject, eye and quarter, registration, and processing metadata respectively. This is for better organization of the columns and clarity of what level each metadata field is on. Similarly, the notes in the metadata are placed at the end of the respective level of metadata – for example, the quarter notes are after all quarter metadata, and before the location metadata. IsNewLocation does not exist, because each new row is a new location, so the location data starts after QuarterNotes.
* When converting the columns of dbt into categorical variables, the code checks only if the column is a double or logical type. Those two are untouched, and any other column type is converted into categorical. Because of filter\_data, the only other type should be a string. In order to convert to categorical type, the column must be the same data type throughout (which should be a requirement of tables in general, but I don’t know for sure) and any missing values should be the standard missing value for that type (ex NaN for double, and ‘’ for strings).

3.) filter\_data

A short sub-function for create\_metadata\_table.m, made to avoid repeating the same code many times. It’s used to clean up data (particularly empty values) before entering it into the metadata table. As inputs it takes the particular metadata entry, the label\_empty Boolean, and a string ‘data\_type’. The string can be two values, and the function returns different things depending on what it is:

* data\_type = ‘continuous’: used for metatada with type double. If the given metadata is empty, filter\_data returns NaN instead.
* data\_type = ‘nominal’: used for metadata that is categorical, and non-ordered. I should have used the name ‘categorical’, because the data being nominal or ordinal isn’t important (at least here. All metadata categoricals are nominal. For disease pathology metrics, the ordinal type of categorical is used. See add\_disease\_stage notes). If the given metadata is non-empty, filter\_data returns it as a string (converting it from its original data type). It is converted to a categorical type in MATLAB later in the function create\_metadata\_table.m. If the given metadata is empty, filter\_data returns either an empty string (‘’, which is important because it will be converted into an <undefined> categorical variable, as opposed to an empty data value [] which throws an error when you attempt to convert it to categorical) or the string “Empty” if label\_empty = 1.

4.) add\_disease\_stage

A function that reads from an excel file with pathology information for BC eyes, converts the pathology levels to categorical variables, and adds it into the database table.

Assumptions/Things to Note

* Currently the script reads from the excel sheet: ‘Y:\shared\Tissue\Human From Ian Mackenzie\AAIC 2017 Stats and Figures\OFFICIAL AD eyes final rating with AD likelihood (WITH AB FROM RETINAS)- Ian Mackenzie.xlsx'. In the future, there are plans to create a common excel sheet (this is one of many, in different locations, saved with similar names) in a better location. It is not known if this sheet will remain updated, or for how long.
* The code matches the subjects in the excel sheet (listed by case number, ex ‘VA 11-47’) with the SubjectId column in the database table. There are some typos in the SubjectId’s that prevent the code from finding them as a correct match to the column in the excel sheet – for example an extra space in VA 13-06, or an extra dash in VA-14-91. These are addressed manually in the code to match their corresponding row in the excel sheet, so caution must be taken not to miss one. As I typed this, I found another (VA-15-38 with the extra dash).
* Various data cleaning is performed on the excel sheet data, so they are properly converted to categorical data later in the script:
  + Most Braak stage (tau) entries are double type, and some were ‘NaN’ (as a string). All are converted to strings.
  + For CAA (CR) and CAA (Abeta), the entries like ‘moderate (was negative)’ are changed to ‘moderate’.
  + For CAA (Abeta+CR), empty cells in the excel sheet were replaced with empty strings.
  + For all others, some entries had an extra space after them, so they were removed with strtrim( ).
* The disease pathology levels are entered into dbt just before the column ‘NumberOfEyesInSubject’
* In order to loop the disease stages into dbt, the table is split into two parts before and after NumberOfEyesInSubject. New columns are created for each disease stage metric, by entering an empty string (‘’) for the first row – which is the first location of the first subject in the database. This subject happens to be an eye with no pathology information, so this is okay. But it assumes the first location will not have pathology information. This was done to avoid troubles with data types that the MATLAB table assumes for new columns, when you start looping in variables one by one. There is probably a better way to code this.
* When looping the disease stage information into the database table, the named columns (ex ‘Braak\_stage\_tau’) are hard-coded to the corresponding column index in the excel sheet (ex column 5). The code does not read the column titles in the excel sheet to determine the metric. This means that if the excel sheet changes the column order, or a new pathology metric is added, they may be incorrectly added to the table. This should be easily noticed if it does happen, but caution should be taken when changing this code.
* The code converts all the disease stage metrics to ordinal categorical. The orderings of each are explicitly set, as they should be ordered (ex ‘mild’ then ‘moderate’ then ‘severe’). The only thing to note is that ‘negative’ is put before ‘sparse’ in some, and ‘not done’ is put before ‘negative’ in one metric. The ordering tells MATLAB which order to list them in boxplots, for example, instead of using alphabetical sorting if the categorical variables are nominal.

5.) merge\_old\_dbt\_data

This function takes the old database table (old\_dbt) and the one that’s being created (dbt), and merges the segmentation information from old\_dbt into dbt. If recalculate\_all is 0, then it will also merge all of the polarization metric stats. If recalculate\_all is 1, then it will only merge the segmentation information, and the new polarization metric stats will be added to dbt in calculate\_polarization\_stats.

Assumptions/Things to Note

* The necessity of this function is based on the idea that the segmentation information will be added to dbt when the deposits are manually segmented. Otherwise, it does not need to be recalculated and can just be copied into the new table. See the notes on segmentation code under future plans at the end of this documentation for more information on these interactions.
* This code assumes ‘IsSegmented’ is the first column containing segmentation information, and ‘SegmentationNotes’ is the last. The column after ‘SegmentationNotes’ is then assumed to be the start of the polarization metric stats. For this reason, caution should be had when changing or adding columns to the segmentation information in the database table.
* Three data types (categorical, double, and logical) are the only types that are checked for in segmentation columns, so if a new column is added with a different type (which it shouldn’t be, as dbt only has these three types by design) then this function needs to be modified to include it.
* Before looping any old information into the new dbt, all the segmentation (always) and polarization (if recalculate\_all = 1) columns are copied from old\_dbt. This is done by copying the appropriate columns from old\_dbt, making them the same height as the new dbt (either by removing rows from the bottom, or copying the bottom row over and over until the heights match), appending these columns onto the new dbt, then replacing all of the rows with empty values. The empty values are added based on the column type (which is why it’s important that a new column with a different type isn’t added): double type columns get filled with NaNs, logical type with 0s, and categorical type with <undefined>.
* <undefined> categorical values couldn’t be looped into the new table from old\_dbt, so any categorical columns (ex ‘SegmentedBy’, ‘SegmentationNotes’) were treated differently than double or logical columns. Each element in the categorical columns was looped over individually, and passed through isundefined( ). If the element was undefined, it was not added to the new dbt. If it had a value, then it was copied from old\_dbt into the new dbt.

6.) calculate\_polarization\_stats

(explain deposit, background, full image columns)

(explain absmean vs mean)

Assumptions/Things to Note

* You need to run Create\_Database\_Table with recalculate\_all = 1, if any new columns are added to the polarization metric stats, or columns are changed. If not, then the new columns will not be merged into the new dbt, and only the locations that are new will have the new metrics calculated for them.
* This code assumes that ‘Background\_Diattenuation\_45\_AbsMean' is the first polarization metric stat column
* In order for this code to calculate polarization stats for a location, the following conditions must be met:
  + The location must be segmented (‘IsSegmented’ = 1, and there is a segmentation mask saved in the database for the deposit)
  + The location must be processed (‘IsProcessed’ = 1, so polarization metric images exist)
  + Recalculate\_all = 1 OR all of the polarization metric stat columns for that location (from ‘Background\_Diattenuation\_45\_AbsMean' to the end) are NaN. This is the criteria used to find new deposits that haven’t had the stats calculated for them yet, as the default values from merge\_old\_dbt\_data are NaNs.
* When looking for a segmentation mask (currently, Jan 12th 2018), the code looks for a folder called ‘Segmentation’ in the location folder, and a bmp image inside that folder that’s name starts with ‘Manually’. This is done to differentiate between the masks created from manual segmentation, and the previous masks that were generated with an automatic threshold. In the future, I plan to make a sub-folder inside ‘Segmentation’ called ‘Euclidian Distance Method’, and potentially change the name of the mask that’s created from manual segmentation.
* If the segmentation mask is empty (all zeros), then the deposit metric stats remain NaN, and the background metric stats are calculated using the entire image.
* The code assumes all polarization images are present if any exist – for example, if there are any diattenuation images, the code will loop from 1 to 5 and expect all five diattenuation images to exist. It also assumes each image is in alphabetical order, and in the correct spot. For example, the order it expects from 1 – 5 is: 45, circ, full, horz, then lin.
* All polarization metric images have 3 pixels cropped from the outside, to avoid registration cropping errors (as of Jan 12th 2018, a fix for this has been made, and it will be added to the automatic registration code in the microscope room soon).
* To clean the data before calculating the stats, several steps are applied:
  + NaNs are exluded
  + Infinities (positive or negative) are excluded
  + Data outside of the expected range of the metric (ex -1 to 1 for circular diattenuation) are excluded
* (explain choice of number of bins for median)
* (different behaviour for MM elements?)
* If a new column is created in dbt, when the code calculates the stat for the first location, it will auto-fill the rest of the column with 0 of double type. These zeros are replaced with NaNs after all of the stats are calculated. The code simply searches all of the metric stats for any values equal to exactly 0, and replaces them with NaN. I assume this is okay, as the chance of an actual calculation coming out to exactly 0 (to the bit), with so much bit depth, is very very low.

7.) circ\_mean, circ\_std, circ\_r

(used for calculate\_polarization\_stats, to perform circular statistics calculations. Just taken off the internet)

**Future Plans**

Segmentation Code and GUI

As of now (Jan 12th 2018) the database segmentation code hasn’t been made into a stable state. The plan is to have it read the current database table, and change any segmentation information that the user decides to change (either segmenting deposits that haven’t been segmented yet, or re-doing old segmentation) and save that information directly into the current database table mat file. It will act separately to Create\_Database\_Table, which updates the paths and metadata, and optionally recalculates all of the polarization metric stats. The segmentation code should have the capability to calculate polarization metric stats as well – when a new deposit is segmented, or an old one is re-segmented, it would be nice for the code to recalculate the metric stats using the newly generated mask, instead of the user having to run Create\_Database\_Table again. This would also prevent situations where the polarization metric stats are calculated from segmentation that doesn’t match the segmentation information in the table.

I’ve made a GUI that allows for manual changing of the segmentation threshold for the Euclidian distance method. I plan to add different segmentation methods to it, and make it something that’s used in the standard processing of images. In this case, I would want the segmentation GUI code to call Create\_Database\_Table on opening, so that the current database table it uses is up to date (namely the paths and metadata). Then, any segmentation the user does is saved directly into dbt. The polarization metric stats could be calculated in the background as the user continues to segment, or potentially after the user is done with segmentation and closes the GUI. In this way, Create\_Database\_Table would be directly linked to the segmentation GUI. It would only need to be used on its own to recalculate the polarization metric stats, if there’s a new metric to calculate or an equation to change.

CHANGES AS I WRITE THIS (starting Jan 9th, 2018), TO CHECK FOR CORRECT BEHAVIOUR LATER:

* Telling create\_database\_paths which registration and polarization folders to choose as the correct ones.
  + EDIT (should be fine, at a glance).
* Removing the try catch around loading the polarization metadata file in create\_metadata\_table, because there’s already a check to see if it’s empty (there may have been another reason for this, like a corrupted file that exists but couldn’t be loaded for some reason).
  + EDIT (didn’t fail).
* Adding VA15-38 to the manually corrected matches between the excel sheet and SubjectId metadata in add\_disease\_stage.
  + EDIT (worked but it’s subject 30, the test one that doesn’t exist).
* Changing Braak\_stage\_tau and A\_beta\_thal to ordered categorical in add\_disease\_stage (check if the conversion happened properly, and if the ordering is correct by default).
  + EDIT (should be fine, at a glance).
* For merge\_old\_dbt\_data, change the first polarization column to 'Background\_Diattenuation\_45\_AbsMean'
  + EDIT (should be fine)
* When merging, treat the categorical columns separately because you can’t loop in <undefined> values into a new table.
  + EDIT (works)