Capstone 202222: Xhand Image Processing

Lucas Carr
Department of Computer Science
crrluc003

Gadi Friedman
Department of Computer Science
frdgad001

Gabriel Marcus
Department of Computer Science
mrcgab004

Contents

1. Introduction

Machine learning is a branch of Artificial Intelligence, which enables computs to use large amounts of data to teach a model to predict outcomes, provided certain inputs. The uses of machine learning are almost ubiquitous; however, its application in the medical sector is incredibly exciting, and likely to have massively beneficial outcomes.

A challenge associated with machine learning is that having a good model requires a large amount of training data - furthermore, this training data needs to meet certain criteria in order to be suitable for the model. Data which is too noisy might be beneficial in its ability to make an existing model more robust; however, if all the training data were too noisy, it is unlikely that anything useful would be produced. In light of this, a considerable amount of effort goes into pre-processing of the training data.

The primary functino of the 'XHAND Image Pre-Processing' software is to take x-ray images of hands, and process the images in order to make them more suitable for machine learning.

---

2. Requirements Analysis

2.1. Identifying Stakeholders

Prior to discussion of the requirements, it was necessary to identify all the stakeholders of the project; this auxiliary process facilitates a more thorough breakdown and analysis of the various dimensions of the project.

The stakeholders of this project were:

- the client *(Professor Patrick Marais)*
- the supervisor *(Sabrina Mackay)*
- the development team *(Gabriel Marcus, Lucas Carr, Gadi Friedman)*
- potential future users *(currently unknown)*

With each of these stakeholders, different requirements became apparent.

2.2. Functional Requirements

The functional requirements of the project were split into two categories: the core, and the periphery requirements.

2.2.1. Core Functional Requirements

- the user should be able to interact with the software through a graphical user interface
- the user should be able to upload a batch of x-ray images they would like to have processed
- the user should be able to write the processed x-ray images to disk
- the user should be able to preview the x-ray images before they are written to disk
- the software should rotate the x-ray images such that the middle finger runs along the y-axis

2.2.2. Periphery Functional Requirements

- given an x-ray image, the user should be able to find a range of similar x-ray images
- given an x-ray image, the user should be able to find a range of similar x-ray images
- the user should be able to perform K-Means clustering on a batch of processed images.

2.3. Non-Functional Requirements

The non-functional requirements of this project were also split into two categories: technical, and exterior requirements. Technical non-functional requirements were concerned with how the software runs, and facets related to the user experience. Exterior requirements were unrelated to software, and rather requirements on the project as a whole - for example, deadlines.

2.3.1. Technical Requirements

- the graphical user interface should be intuitive and easy to navigate
- the software should be reasonably efficient
- the software should provide understandable feedback in the even of errors
- the software should be simple to install on a computer

2.3.2. Periphery Functional Requirements

- the software development should use Gitlab as a repository management system
- the software development should be completed and tested by the 19th of September

2.4. Use-Case Narratives

The actor would like to process a batch of x-ray images

Scenario:

The actor has a folder of images of hand x-rays. They would like to use these images as training data for a machine learning model; however, the images contain clutter, and are aligned in various different ways - which makes them less suitable for use as training data. They would like to use the software to process these images, making them more suitable for use as training data.

Preconditions:

- The software has been opened and runs on the actor's computer
- The actor has a directory on their computer they have permission to read from
- The actor has a directory on their computer they have permission to write to

Main Flow:

1. The actor uses the software's interface to select the directory which contains images of x-rays of hands.
2. The directory contains .png formatted images.
3. The path to the directory is recorded, displayed to the agent, and the contained images are read into memory.
4. The actor uses the software's interface to select the directory where they would like to store the processed images.
5. The path to the specified directory is recorded and displayed to the agent.
6. The actor uses the software interface to activate the image processing component of the software.
7. The images are processed and written to the directory specified by the agent.

Alternate Flow:

1 A. The agent decides they do not want to process images.

2. The agent exits the software.

2 A. The directory contains files which are not of .png type.

3. The actor is presented with a notification that the directory they specified did not contain correctly formatted files.
4. The actor corrects the formatting error, and repeats step 1 with the appropriate data and the flow continues as stipulated in main flow.

2 B. The directory is empty.

3. The actor is presented with a notification that the directory they specified did not contain correctly formatted files.

4. The actor corrects the formatting error, and repeats step 1 with the appropriate data and the flow continues as stipulated in main flow.

4 A. The actor does not specify a directory where they would like the images to be stored.

5. The interface displays the path to the default output directory.
6. The actor uses the software interface to activate the image processing component of the software.
7. The images are processed and written to the default directory.

The actor compares the alignment accuracy of a processed image to its original version

Scenario:

The actor has performed all base-level tasks and the images have been processed and stored in an accessible directory. The actor would now like to visualise the changes made to a specific image, comparing it to the original, unprocessed image.

Preconditions:

- The software has been opened and runs on the actor's computer
- The actor has selected a directory containing .png images they would like to process
- The images have been processed and stored in either the directory specified by the actor, or in the default directory

Main Flow:

1. The actor uses the interface to navigate to a chosen image.
2. The actor uses a button on the interface to display the comparison of the processed and unprocessed image.
3. The image is displayed on the interface, with an overlay of the original, unprocessed image.
4. The actor saves the composition of the two images to a file.

Alternate Flow:

2 A. The selects a different image to view with its original image overlaid.

2 B. Since processing the images, the actor has deleted some of the stored files, or removed the storage device.

3. The software notifies the actor that there has been an error, and that files required to perform the specified task cannot be found.
4. The actor rectifies the error (by either reconnecting the storage device, or redoing the image processing), and returns to step 1.

4 A. The actor does not save the composition of the two images to a file, and instead performs another task, or exits the program.

3. Design

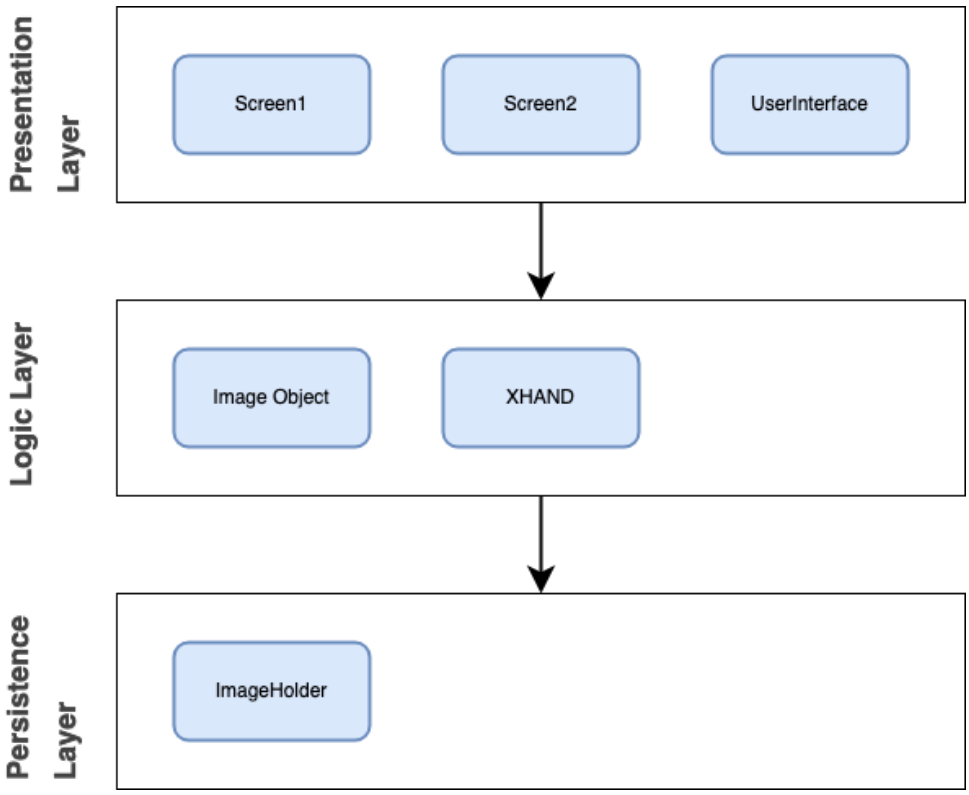
3.1. Justification of Software

As mentioned in the introduction, having good quality training data is a necessary aspect of machine learning; furthermore, given the volume of training data required, processing images manually is unlikely to be a possible route to attaining this data. Consequently, the primary purpose of the XHAND image processing software is to enable a user to upload a large amount of images (for the current use case, these should be x-rays of a person's left hand). These images are then put through a series of processing steps - found below, and to be explained in further detail in Section 6 - and finally written to storage.

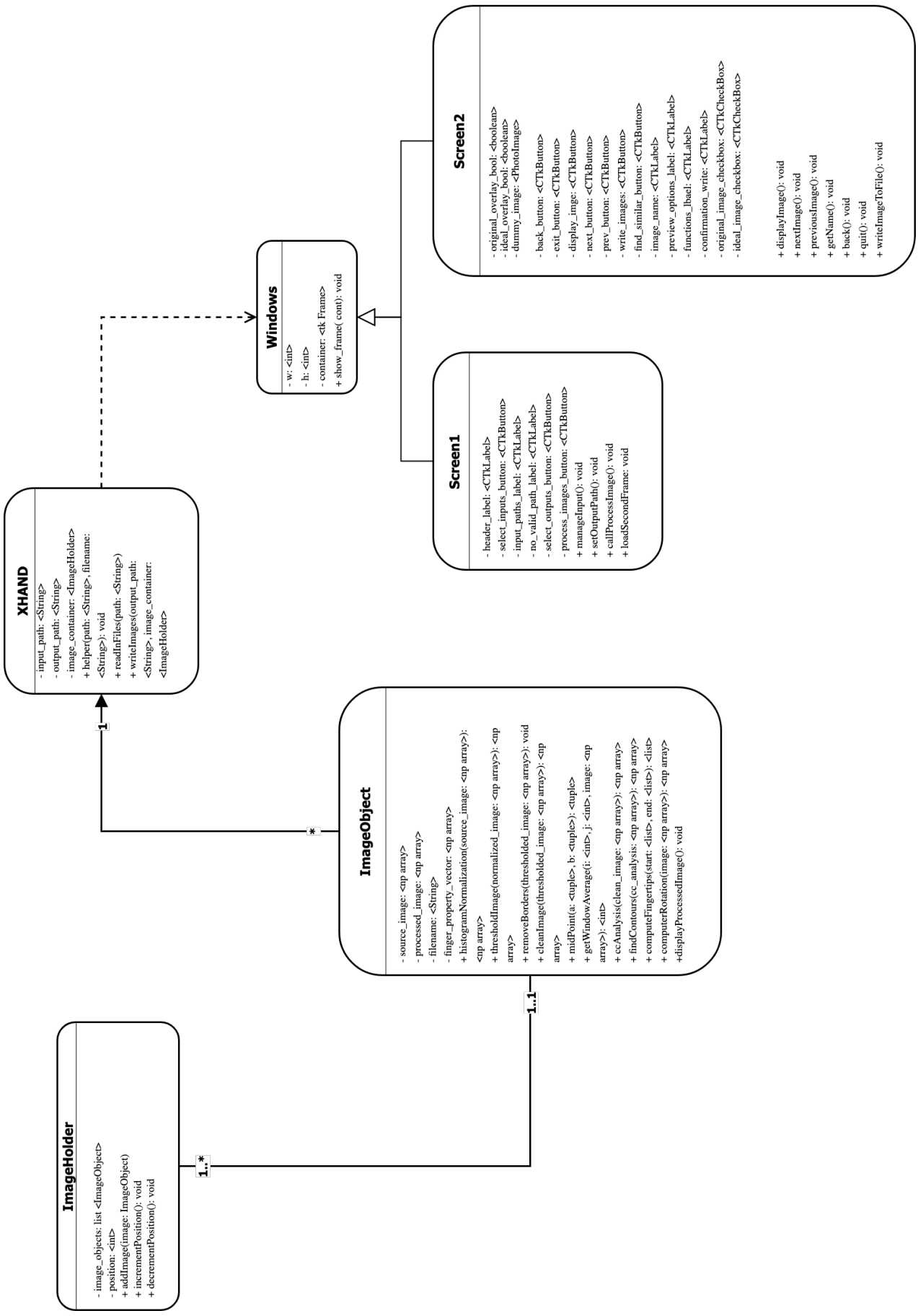
1. Histogram Normalization
2. Binary Thresholding
3. Border Removal
4. Adaptive Thresholding
5. Connected-Component Analysis
6. Contouring and Fingertip Extraction
7. Masking
8. Rotation
9. Clustering
10. Metric Comparison

A more detailed discussion of these image processing steps can be found in section 3.4, where there is an explanation of the approaches and solutions used for each step of image processing.

3.2. Layered Architecture Diagram



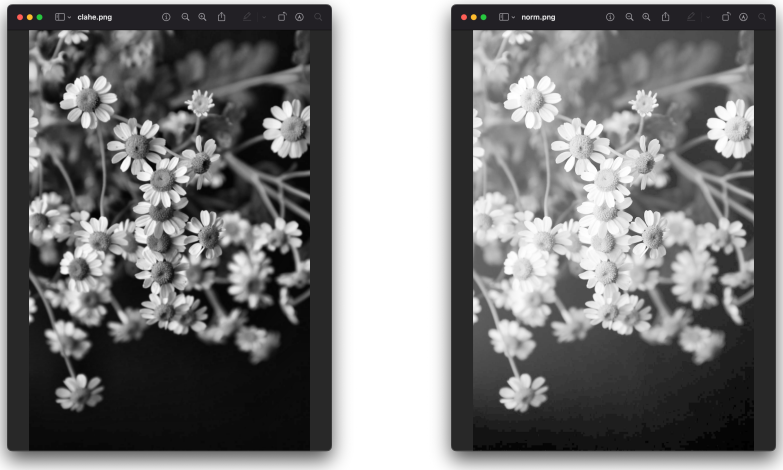
3.3. Class Diagram



3.4. The Processing Pipeline

To begin with, data from each image would be read into a numpy array - at risk of repetition, the justification being that numpy arrays are far more efficient than conventional python lists. For each image, a corresponding ImageObject was constructed. The image data would then undergo histogram normalization.

The approach to histogram normalization was to make use of openCVs Contrast Limiting Adaptive Histogram Normalization (CLAHE). CLAHE is a type of *adaptive histogram normalization*; traditional histogram normalization struggles to produce useful contrast benefits when the distribution of pixel values in the image is large: when certain parts of the image are considerably lighter or darker than the rest of the image, the effects of the process are poor. An alternative is to make use of adaptive histogram normalization methods, such as CLAHE. This process performs the histogram normalization based of a window of neighbouring pixels, rather than the entire image. A particularly desirable aspect of CLAHE is that it provides the ability to impose a limit on contrast, which prevents regions of mostly divergent (noisy) data to be amplified by the use of adaptive thresholding.



(a) CLAHE (b) Normal

Figure 1: Contrast between using CLAHE and openCVs 'equalizeHist' function

Once the histogram normalization process had been applied, the image could then be thresholded. OpenCV provides the ability to perform either binary thresholding, or adaptive thresholding. Binary thresholding works by evaluating each pixel against a global value, or threshold. Pixels with a value greater than the threshold become 255, otherwise their value is set to 0. Adaptive thresholding works similarly to CLAHE, where pixels are evaluated with regard to their neighours; a threshold value is determined by evaluating neighbouring pixels (for example, by taking the mean of neighbouring pixel values.)

For the current use case, a decision was made to use binary thresholding. For the most part, the images which were to be processed did not require adaptive thresholding in order to produce good results; moreover, adapative thresholding results in a longer processing time.

A downside of this choice was that certain images were poorly suited to thresholding, and would result in a image similar to *Figure 2: a*. This type of image posed a problem for the next step of the image processing pipeline: connected component analysis. Due to the thresholding being unable to produce a clear distinct 'component', a single hand object would not be able to be identified. To combat this, a novel adaptive thresholding algorithm was written which would scan the lower quarter of the image and determine the average pixel value of neighbouring pixels. If this average was greater than some threshold, the pixel value would be set to 255, otherwise 0.

Threshold Correction Algorithm

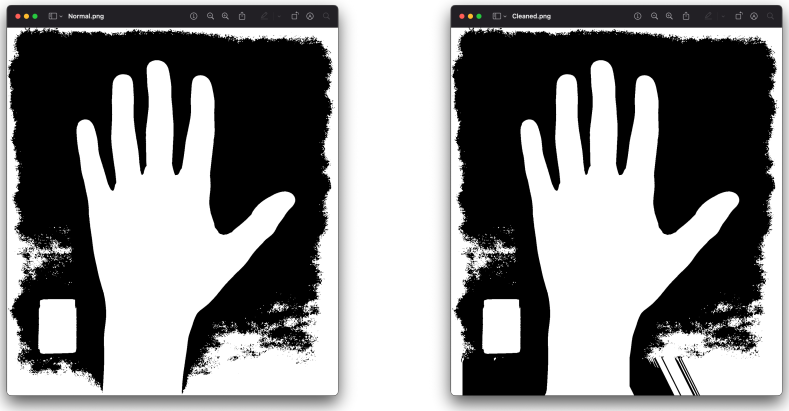
```
for each pixel (x,y) in lower 1/4 of img:
    if pixel x value < (img.width / 2)    // pixel is in left half
        find average of pixels in 3x2 grid,
        where grid pixel is in bottom right

    else:                                 // pixel is in right half
      find average of pixels in 3x2 grid,
      where grid pixel is in bottom left

if average > threshold:
  return 255

else:
  return 0
```

The algorithm traverses the pixel values top-to-bottom, and updates them as it runs, which results in a clear edge along the areas of the wrist, the results can be seen in *Figure 2*.



(a) Just Binary Thresholding (b) Binary Thresholding + Cleaning

Figure 2: Contrast between only using Binary Thresholding and using additional algorithms

As previously mentioned, after the image has been thresholded, a connected component analysis can be performed. This process extracts components (adjacent pixels); because of the earlier histogram normalization and additional thresholding technqiues, the heuristic that the hand would be the largest component is adequate. Consequently, only the largest component was kept. From this component, the contours of the hand can be detected using the openCV library. These contours provide the co-ordinates of the edges of the hand. In turn, these edges allow a convex hull to be created.

The hull provides the ability to detect convexity defects, which are points which correspond to the farthest points for each deviation from the convex hull. Convexity defects represnt points where the contour of the hand diverges from the convex hull. A trianglular shape of points *A, B, C* is formed, where *A* and *B* are the points on the hull where the divergence occurs, and *C* is a point on the contour which is farthest from *A* and *B*. If $\angle ACB < \frac{\pi}{180}$, then the convexity defect represents the points at which two adjacent fingers connect to the palm of the hand. Each defect contains a point corresponding to a fingertip. Using these points, the coordinates of the fingers can be identified.

An issue with identifying the fingertips was that the point at which two adjacent defects touched the convex hull was different - there is an illustration of this in *Figure 3 (b)*. To combat this, the midpoint between adjacent fingertip points was calculated, which provided a suitable point to represent the tips of each finger.

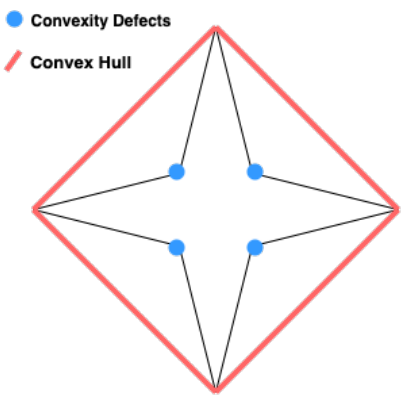
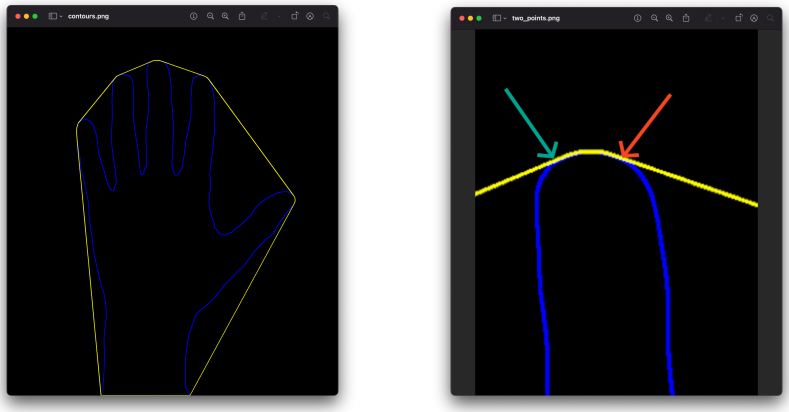


Figure 3: Example of convexity defects



(a) Convex Hull (b) Two points of a fingertip

Figure 4: Drawing the convex hull on the image

There is more text on thew ay.
eweaweaweaweawerawerawer

4. Implementation

4.1. Discussion of User Interface

First Screen

Figure 1 shows the initial screen the user is presented with when launching the software. The purpose of this interface is simply to facilitate the processes of specifying input and ouput directories.

While this is a trivial task, it is essential that it is as simple as possible for the user to correctly perform this process. This is why there is a distinction between this stage of the program, and the next stage (which concerns additional functinoality/visualization). The user is presented with 3 buttons; the first, labelled 'Choose Input Directory', opens a file dialog box where the user locates the directory which contains the images they would like to upload. The second button uses the same process to enable the user to specifiy an output directory - however, the user may opt not to select any directory, in which case the files will be written to the program's internal *./Images/* directory.

Figure 5: Example of overlayed images with altered RGB values

The final button is used, and can only be used if a valid (contains *.png* files) input directory has been specified, to call the methods which perform the processing on the images. There is a slight delay here as the program processes the images, and once it is finished, the user is moved to the second screen.

Second Screen

Figure 2 shows the screen displayed to the user after the images have gone through the processing steps. The user can cycle through the images using arrows, and choose to view the images overlayed with either the unprocessed image, or an 'idea' standard of image by selecting/deselecting the respective checboxes.

The functions available to the user are accessible on the left side of the interface - the user can specify specific parameters to use - *for example*, they can choose the number of clusters using the selection box.

Due to there being multiple different features available for the user to engage with, detailed explanantions of what each feature does and how to correctly use it, are not possible to display on the interface. These descriptions can be found in the user manual; however, it is generally quite tedious to have to refer back to documentation. In an effort to ease this, the user can hover their cursor over a function and be a tooltip will appear, providing a brief description of the function.

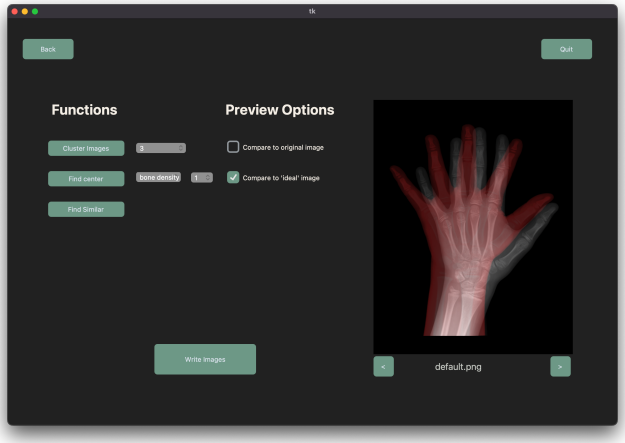


Figure 6: Example of overlayed images with altered RGB values

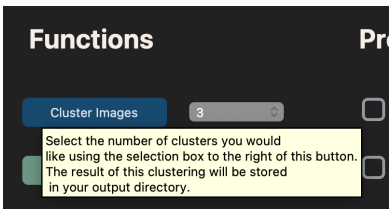


Figure 7: Example of tooltip on hover

4.2. Description of Data Structures

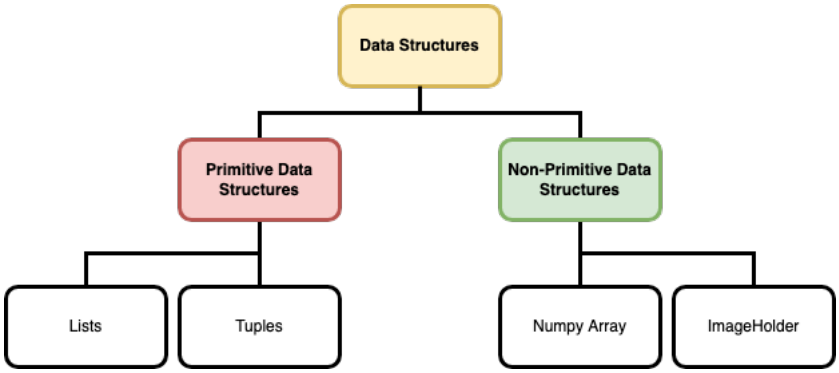


Figure 8: Data Structures Diagram

Arrays

Decisions around which data structures to use in different contexts was an important aspect of the software's development. The nature of the program is such that images need to be read into multi-dimensional arrays - this stores RGB and Alpha values of each pixel of the image, and can require quite a large portion of memory. Consequently, numpy arrays were prefered to conventional python lists. Numpy arrays provide a more memory efficient solution to storing mutli-dimensional arrays, while still allowing the same manipulation ability of array elements.

Image Object Container

An ImageObject class was created to represent each image as an object. This was necessary as it allowed the software to store different states of each image within a single entity, as well as perform processing methods on image data. Instead of simply storing these objects within a list, the decision was made to create a class which acts as a container for all the image objects. Contained in this class is a list with the image objects; however, there is further functionality which enables the image object container to store and manipulate a position, as well as have a 'default' example of an image object (which is not actually part of the inputted images)

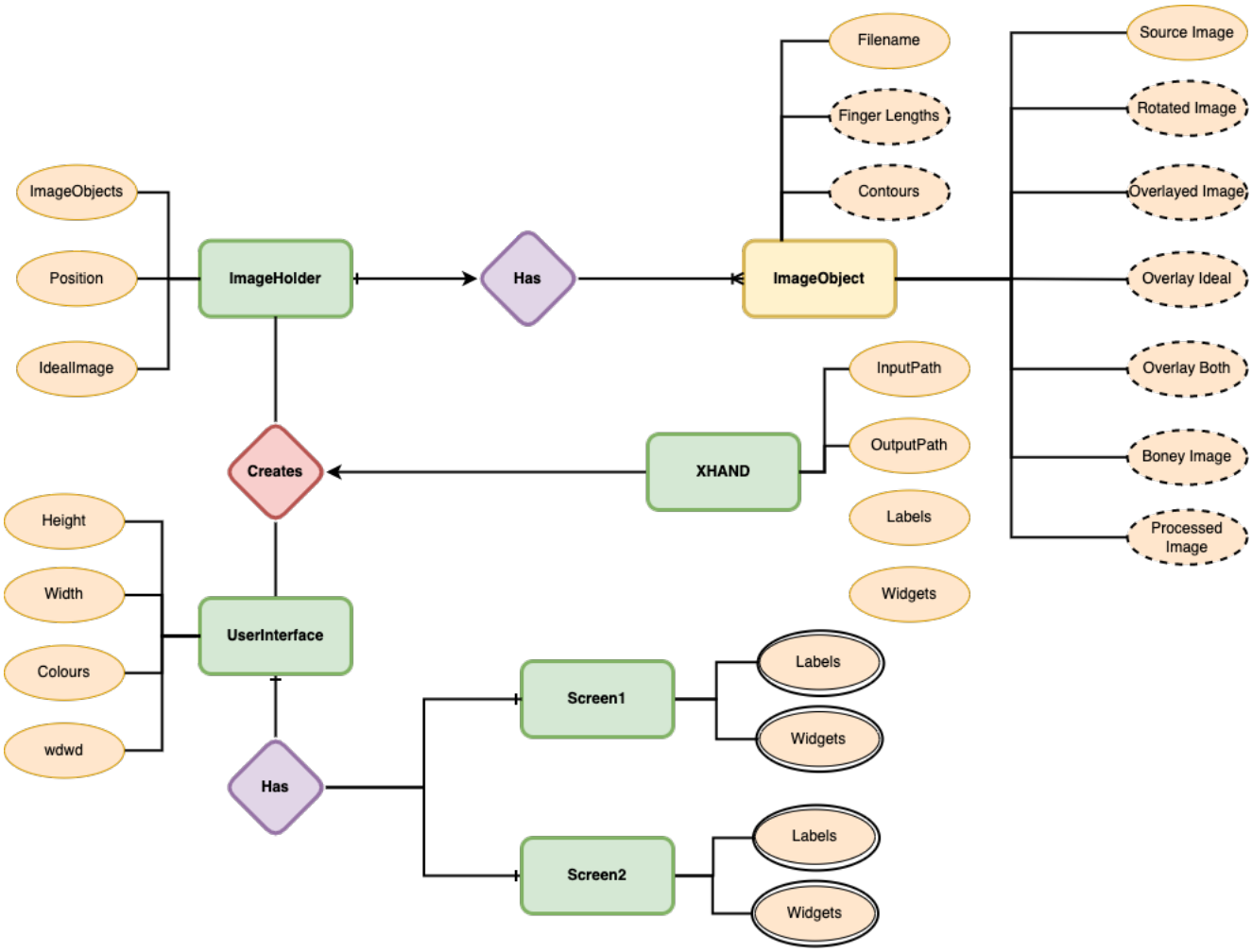
Tuples

Tuples are a data structure which enables multiple items to be stored in a single variable. Once instantiated, the elements, and the order of elements within a tuple are immutable. This enables the creation of finite, static compositions of data; the use of this is analgous to the benefit of having an 'objet' - that there is a way to consistently reference data throughout the program. For example, a tuple consisting of the filename and an associated numpy array is a useful way of accessing/returning image data.

Lists

As previously discussed, numpy arrays are generally a more efficient way to manage large arrays; however, frequently we need a very simple array of a few elements. In this case, lists are used, as they are conceptually very simple, and there is a great deal of familiarity with the syntax and manipulation of lists.

Figure 9: Entity Relationship Diagram



5. Validation

Various testing methodologies were utilised during the development process. The software made use of a model-view-controller (MVC) design pattern, which meant that much of the internal logic of the software existed independently to what was actually displayed to the user. Consequently, unit tests were written to test components of the internal logic, while manual-based testing was used to validate the controller and view aspects of the software.

5.1. Unit Testing

The ImageObject and ImageHolder classes were both fully unit tested. The tests were written to ensure that, given both correct and incorrect inputs, the methods would function correctly.

---

6. Discussion

6.1. Discussion of Core Functionality

The main goal for the development of the XHAND software was to produce a tool which could transform noisy x-ray images of hands into cleaned images, which would be suitable for use as training data for a machine learning model. The approach to achieve this, briefly touched upon in section *3.1* was to used a pipelined stream of image processing methods.

6.2. Discussion of Peripheral Functionality

---

7. References

References

[Kopka and Daly(2004)] Kopka, H. and Daly, P.W. (2004) *A Guide to LaTeX 2ε: Document Preparation for Beginners and Advanced Users* (4th edn). Addison-Wesley.

[Lamport(1994)] Lamport L. (1994) *LaTeX: A Document Preparation System* (2nd edn). Addison-Wesley.

[Mittelbach and Goossens(2004)] Mittelbach, F. and Goossens, M., (2004) *The LaTeX Companion* (2nd edn). Addison-Wesley.

8. Documentation

8.1. User Manual

Installation

All the required python libraries are listed in requirements.txt; you can easily install these by using the command make install, use make clean to uninstall if necessary.

*! If you encounter an error similar to ”python not configured for Tk”, the solution for this is to manually install Tkinter. Use your package manager (pacman/homebrew/paru), e.g. sudo pacman -S tk

This is the result of a known buggy interaction between TKinter and virtual environments.

The software is now ready!

Running the software

You will need to have:

1. A batch of ‘.png‘ images stored in a directory you have access to.
2. Space on your drive to write images to (if you wish to write the processed images to file)

You will need to have:

1. A batch of ‘.png‘ images stored in a directory you have access to.
2. Space on your drive to write images to (if you wish to write the processed images to file)

8.2. Code Legibility