Headstone Photograph Processing System

Comprehensive Design Document

Group 2

Spring 2021

The Headstone Photograph Processing System is a collaborative effort between Trevor Jones, Yubo Li, Robinson Nunez, and James Simmons in coordination with the University of Central Florida Department of Humanities and the National Cemetery Administration.

Table of Contents

Executive Summary

The UCF Department of Humanities in coordination with the National Cemetery Administration are tasked with the management of resources and information pertaining to a variety of cemeteries, mostly military, in Florida. Part of their responsibilities include the management of a database of images of the headstones in these cemeteries. The first part of this process is to acquire photographs of every headstone, but there is no readily available mechanism for denoting which photograph corresponds to which headstone. As such, the department is in possession of thousands of headstone photographs as well as all the relevant information pertaining to each gravesite, but no means of linking the photographs to the information. In other words, given a particular gravesite, there is no mechanism to locate its photograph within the thousands of photographs.

The objective of this document is to comprehensively detail such a mechanism. The Headstone Photograph Processing System (hereafter referred as "system", "program", or "project") shall be able to inspect an image (or, more precisely, a directory containing many images) and determine from the image alone which gravesite it corresponds to. The images are then to be appropriately labeled so as to be searchable later. As an additional objective, the system is also to "clean up" any images to make them appear more presentable; that is, rotate them so that the headstone is properly upright, and crop off any extraneous content besides, below, or above the headstone.

The system shall be organized into four sections. In the first section, the image is rotated so that the headstone is upright. This will consist of a macro rotation (intervals of 90 degrees) and a micro rotation (intervals of 1 degree). The necessary rotation degree is to be determined by a neural network looking to identify the coordinates of the edges of the headstone, the angle of which is used directly to determine the appropriate rotation angle. Second, the image is cropped such that only the headstone remains. This will also use a neural network, which seeks to identify the bounding corners of the cropping region. Third, the headstone is to be inspected by an Optical Character Recognition (OCR) system, also built using a neural network, to extract the text engraved into it. Fourth and finally, this headstone text is parsed and compared against a dataset containing information about the gravesites to identify which gravesite is pictured. The image is then labeled accordingly.

The system should be able to properly rotate and crop images with a 90% success rate, and label images without user assistance with an 80% success rate (bearing in mind that the rotation and cropping must be successful for the labeling to be).

Project Description

Objectives and Function

The objective of this project is to develop a program that can take as input a large quantity of photographs of headstones in military cemeteries and perform two operations on them. First, the photographs are to be rotated and cropped such that the text on the headstone is as close to horizontal as possible and all non-headstone parts of the photograph are cropped out. Second, the program must rename the photograph image filenames to correspond to the entries in a database of names of the people whose graves are in the photographs, either by optical character recognition on the text on the gravestone or by some user-assisted process.

The program is to be as automatic as possible, only requesting user assistance in the most dire of circumstances, such as when a headstone bears a name with multiple unassigned database entries or when the text is completely unrecognizable.

The program should be easy to install on a local Windows machine, ideally with a "single-click" installer file. The user interface is to be comfortable to use but needn't be especially visually appealing. The program will not need to talk over a network and should be able to perform all processing locally on the installation machine.

The program will have a variety of use options, such as a "feedback"/"no feedback" toggle, which determines whether the program prompts the user for assistance in cases of uncertainty or simply moves the uncertain case to an anomaly folder for processing later.

Personal Motivation

Robinson

Artificial Intelligence and Machine Learning for me are two fields that humans have only scratched the surface of. I feel that there is more to be discovered and I want to be a part of that discovery. The field applies so many other disciplines such as mathematics, stats and even physics in some applications. For the project, we will be using robot vision to ensure that headstone data is properly matched to an image. When picking projects, I prioritized ones with emphasis on the above-mentioned fields or just the field of AI in general. How I came to pick this was because it focused on my favorite aspect of AI which is robot vision. On a personal level, I picked the project because not only does it let me do something I'm passionate about, but it also lets me contribute to my school and local community one more time before I graduate.

Yubo

Computer vision for me is not simply a study of the simulation of the human eye. It is more relevant to an abstract conception of how mathematics can visually present and explain the world for human beings. In this SD project, we will investigate an automated technique based on machine learning algorithms and Convolutional Neural Networks to detect and identify the military headstone and the text on the headstone. My passion for this project is not only the techniques and methods we will learn and implement to accomplish such a useful application, but also can our project benefit people to organize their most important memories.

James

I have been fascinated with artificial intelligence and machine learning for a long time now, and all the classic examples of this technology in use are related in some capacity to image recognition and/or computer vision. Suffice to say, the top of my project preference list was filled with projects involving these technologies. This one spoke to me in particular, however, for completely non-technological reasons: I come from a very religious background and many of my relatives served in the armed forces. Respect for the fallen, and the desire to carry on their name, memory, and legacy are extremely valuable to me. Every serviceman whose grave is depicted in these photographs deserves to have their story told, and I'm honored to be able to help in that endeavor.

Trevor

The headstone recognition project was one of the first projects to be pitched and was still the last project on my mind when making my decision. Continuing to learn and progress is the main goal of my college career. Going into this project, my hope is to learn as much about machine learning and computer vision as possible. Artificial intelligence is becoming increasingly popular as technology increases and this project can be the beginning of my studies in the field. As well, being able to implement a full stack web application is very fascinating to me and has endless possibilities in the field of computer science.

Broader Impacts

This project has a clear impact on the armed services community. While it won't aid in any military operations, the personal and emotional impact that it may have on the lives of the members of our armed forces or their families is clear. If we fail to properly record information on our fallen servicemen, their legacy can be lost forever. Knowing that this will not happen would have a positive impact on morale.

Additionally, while the work that goes into this project is localized on military gravesites, the technology developed could be generalized beyond that for use in civilian graves as well. In the modern day, remote connectivity is more important than ever; schools, businesses, and even social events are all moving to online mediums. By digitizing gravestones, we can increase accessibility for people who want to visit them, but cannot travel there physically.

Legal, Ethical, and Privacy Issues

An issue that arose when speaking to Dr. Giroux was the process of gathering additional data to test the applications being created in the project. Dr. Giroux will supply her own headstone data which will range in the thousands. To avoid testing all of her supplied data, instead of saving it for the working product, Dr. Giroux suggested using websites such as BillionGrave[1] and FindAGrave[2] to gather the additional data to be tested. The initial idea was to check whether there were any privacy issues involved with gathering and using their data for our process. With further inspection of their privacy policies, no issues will arise with gathering their data for testing purposes. The gravestone information for these websites is in the public records which can be accessed by anyone who searches for the data.

As far as ethics, we are not using the headstone data for any other reason except for renaming the images. There is no process of defamation or wrong behavior being used to the individuals whose headstone information is being recorded. We are simply automating a process of renaming images that already takes place by Dr. Giroux and her colleagues.

No other legal issues are needed to be mentioned and if any come up, we can dissect them further.

Specifications and Requirements

- The program shall process a folder of images of headstones with a maximum of 6,000 images and max size of 32 GB
- The program shall produce a new folder containing the same images after they have been rotated, cropped, and renamed according to their text
- The program shall launch from a single Windows executable file (.exe)
- The program shall run on a local Windows workstation and shall not require any network connections
- The program shall rotate each photograph at a macro (90° increments) and micro (1° increments) such that the headstone is upright while retaining the original resolution with at least 95% accuracy
- The program shall crop each photograph to include just the headstone and as little extraneous features as possible with at least 90% accuracy
- The program shall recognize the text written on the headstones with at least 85% accuracy
- The program shall rename the image files based on the recognized text and the data within a CSV document with at least 80% accuracy

Preliminary Ideas

Robinson

Front-end desktop application

As per Dr. Giroux's specifications, the front end will only be a means to kickstart the actual image processing. A web application won't be needed as the user base of the app will only be Dr. Giroux and her colleagues that work directly within her office. For the interface, there would only need to be a section where the user can enter a directory for where the photos are coming from. To be safe, there should probably be some kind of check to make sure the file contains the correct input data. Other than that, a simple button that begins the process should suffice.

OCR Strategies

The OCR step would probably involve a bit more complexity than other steps as there are more roadblocks and obstacles to deal with. One of these obstacles is the deciphering of the text. Not all the text on the graves will be perfect, far from it. The text may be modified, have a different font or just be illegible all together. For starters, all the text on the grave should be isolated first. Once the text is isolated, it can begin to be parsed. Anything that is not useful should be dropped immediately. This can range from anything that is not the name or any dates. This approach would be more of a bottom-up approach as we isolate more and more of the text until we have what we want. There will have to be a separate mechanism that actually identifies what the text is to determine if it's relevant or is just too unreadable to even deal with. With parsing, we get the big picture first and remove the unnecessary details later. An alternative approach would be to try to isolate the text on the spot. This would be much harder as you actually have an algorithm that actively searches for where the text might be. This would put more effort on the model as now you have to locate something you may not know the location of and you still have to decipher the text. Parsing should be a less expensive operation and would give the algorithm only one responsibility.

Rotation

For the rotation component, there are two major players, the macro rotation and the micro rotation. The macro rotation will deal with images that need to be rotated completely. These are images that aren't even displaying the gravestone upright. Micro rotations deal with the minor transformations. These transformations deal with getting the image just right and making it straight. An idea for these would be two train these two aspects separately and

give each model the relevant training data. However, this could be a problem as there may not be as much training data for the micro rotations due to the specificity.

James

Web Application or Desktop Application

The user interface can be either a standalone window created by the process directly or it could be run in a browser as a web application. A standalone process window is probably the better choice because we wouldn't need to worry about web application overhead or database management, and starting the program can be as simply as double-clicking an icon, rather than starting up a server and connecting through a browser. The browser could be good for if we wanted to generalize and expand the functionality, but that's out of the scope of this project.

OCR Strategies

The OCR step has two distinct and very challenging parts to it. First we need to locate the section of the image where the text is. This is made especially challenging by the fact that we only really want the name, and not any other text that may be on the stone, though I suppose if we got ALL the text then we could just parse out the name later, which would probably be easier. Either way, once we determine where the text is, we'll also need to do some image manipulation in order for pre-existing OCR libraries to play nice with it. Typically, OCR wants straight black text on a plain white background. Curved text, white noise, and lacking contrast all need to be accounted for and filtered out. One idea is to ramp up the contrast by a huge margin, but that may cause additional artifacting that may mess up the reading. Ideally, we don't need a perfect read. So long as we get most of the text right, we can do a series of fuzzier and fuzzier matches until we get a hit, and query the user for confirmation if we need to increase fuzziness more than some amount.

Continue Processing While Waiting for User Input

In the event that the program needs user input, such as if multiple headstones have the same name or the text is truly unreadable, we would like for it to continue processing the pictures it can process while waiting. Humans are slow, and waiting around for input before doing anything else could make the processing time far too long. One solution would be to send all the special-attention images into a separate folder along with some metadata and have a separate dedicated process ask for user input on each of them. Alternately, we could try to spin up the user-feedback section of the program on a separate thread, and maintain a queue of special-attention images. If the queue is empty, the user can see a progress bar of

the processed images. The main processing thread can add images and relevant data to the queue, and the user-feedback thread can remove and process elements from the queue.

Running Python Scripts on Windows without Python Installed

Since many of the people working with Dr Giroux aren't technically inclined, we'll need a way for them to be able to run our program without installing a bunch of packages from the command line. We should be able to create a Windows executable .exe file using the python package pyinstaller.

Trevor

Rotation/Cropping

The project will have multiple phases to achieve the desired result of correctly relabeled headstone images. The first being rotation of the image. Using image filtering to help detect the edges of the headstone. An idea can be to use the base edge created from the ground and the headstone bottom to act as the anchor point in which the headstone should be rotated. The goal would then be to get the edge as close to 180 degrees as possible. Problems can arise with slanted headstones that may not be upright with the bottom edge rotated, or images that are so rotated to the point that a side of the headstone appears as the bottom edge (in the algorithm's eyes). Others can be to detect the side edges of the headstone and rotate the image to position those edges in the upright format. As for cropping the images, a similar pattern of edge detection will be needed. After the filtering and edge detection, the image should be cropped around the headstone with a "15" pixel buffer. This way there can be some buffer if the detection feature crops out some of the lettering.

OCR Ideas

For the OCR aspect, there are three main aspects to address. The first will be to detect where the lettering is on the headstone. This may be simple for headstones which have all the information in the same area, but some military graves may not. The next will include cropping the image around this lettering, with a small buffer. This way we can zone in and start making detections to only the area we know includes the lettering. The last will be the actual detection. This will be tricky as there is no set model for which the headstones will appear. Some will be the "typical" headstones that sit in the ground, some may be plates that are dug into the ground, while others may be military specific memorials. As well, there will be no specific fonts or stylings of the lettering. So, the OCR needs to be able to detect the information whether it is an unusual font, or even curved in the lettering. As

well, there may be instances where the lettering is discolored, or corroded, in which partial matches will need to be made. For example, if the name is John Smith, but part of the lettering is corroded, it should still be able to read the non-corroded area with placeholders for the other values. It can return the string Joh* *mith if the n and s are corroded away. Another problem that may arise is the issue with lettering that may be slanted. Some headstones may have sunken into the ground over time, which means that even if the headstone is rotated to match the side walls, the lettering still may not be straight. This will need to be addressed in the rotation section and if not may present an issue in the OCR phase for a specific few instances of headstones.

Data Sets

Having data sets to test that include each critical point of detecting and labeling the images is key. For example, pictures that have curved lettering, severely rotated headstones, corroded lettering, etc. Having specific folders for each point of emphasis will make the testing process easier to account for every aspect.

Yubo

Fourier transform for headstone rotation

- Transform image from time domain to frequency domain diagram;
- Applying Hough linear transformation to find the rotation angle of the straight line;
- Rotate the image to the correct angle.

Headstone Detection

Yolo algorithm divides images into N * N cells, each of the cells is responsible for detecting the target of which center is located at the cell. Each cell will generate a number of bounding boxes and confidence scores which include both possibility of containing an object and accuracy of the bounding box coordinates. The measurement of accuracy for bounding boxes will be the IOU between ground truth and bounding boxes. Each bounding box will have 5 prediction values, the first 4 represent the size and position of the bounding box, and the fifth value represent the confidence of having the target.[3]

Resnet Classifier

Resnet is an optimization of Convolutional Neural Networks. The main idea of ResNet is to add a direct connection channel to the network, allowing the original input information to be directly transmitted to the subsequent layers. Theoretically, the deeper the CNN

architecture turns into, the training error is expected to continue to decrease; however, in practice, the training error will temporarily decrease and tend to increase. The vanishment of the gradient is the main consideration in explaining this. In each hidden layer, RELU will be implemented as an activation function, and when the model starts training forward, all the values less than zero will be set to zero, and several feature values disappear, so the gradient correlation is actually decreasing with increasing layers.

Focal loss for unbalance positive and negative

Focal Loss function is an algorithm that deals with the situation when training samples have imbalance positive and negative samples. In our project, we will have much less headstone (positive samples) than the negative samples (background anchors); as a result, implementing Focal loss function will be necessary for the training phase.

Curved Text recognition: Mask RCNNFCN & Faster RCNN

- Applying object segmentation techniques (MASK RCNN etc.) to locate the region of text on headstones.
- Dividing the text region into cycle cells.
- Iterate the cells through the direction of the tangent line and apply Faster RCNN to identify the text.

Block Diagram

Below is a block diagram detailing the complete operation of the system at a high level.

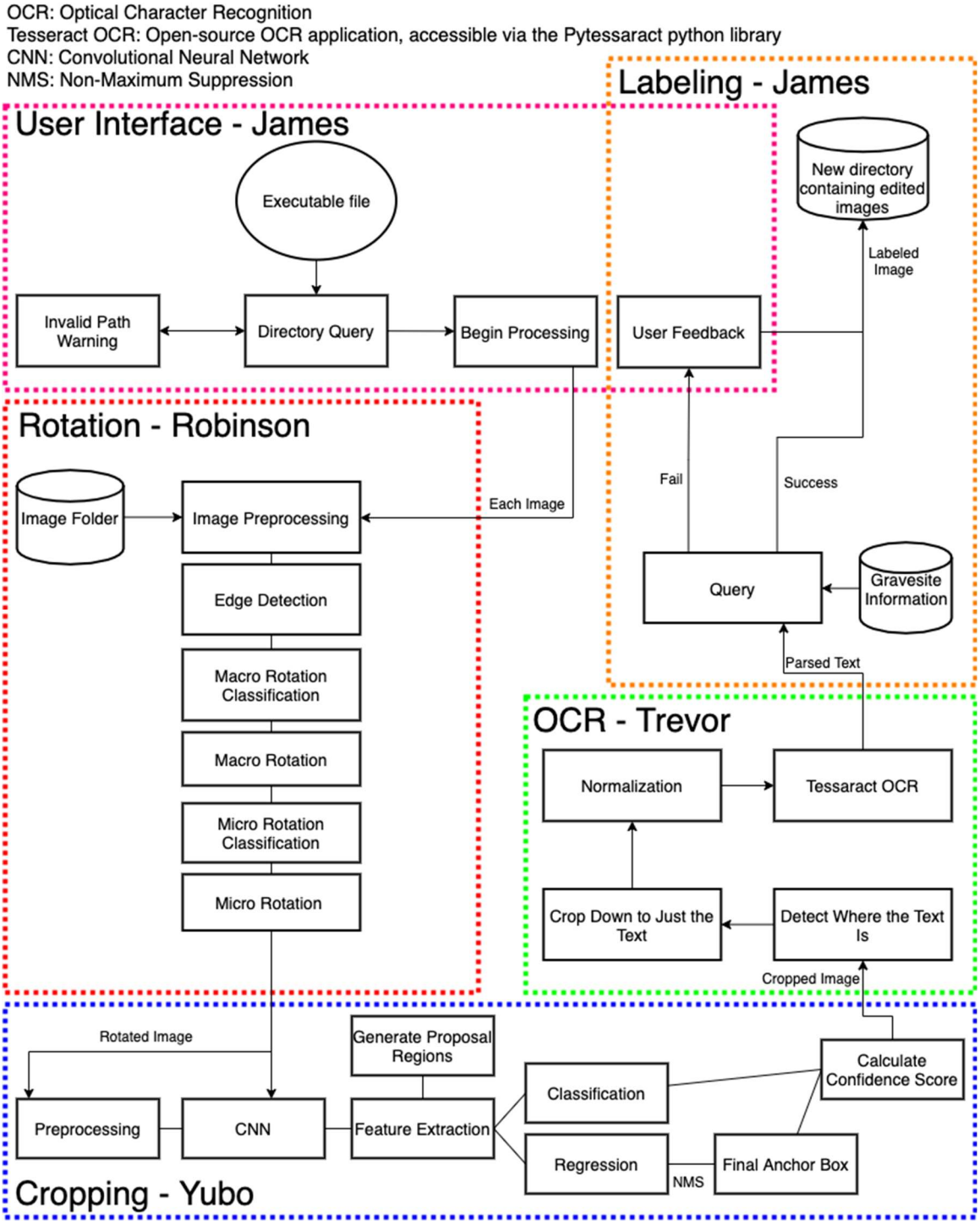

Figure 1: HPPS Block Diagram

Program Driver

Objectives

The program consists of multiple disjoint sections that each exist totally separate from one another, and are to be treated as black-boxes. To join these sections together into one cohesive program that actually accomplishes the stated task, a central driver is necessary to control the operation of the program. Ideally, the driver does as little work as possible, and only facilitates data transfer between the sections. It is the job of the driver to iterate over the images to be processed, pass them between the sections, and save them to the appropriate locations. The driver also communicates with the user interface so that the user can receive updates on the program's progress.

Directory Structure

This program will operate on many images that can be in one of a variety of different states at any given time. The program should also be able to halt as quickly as possible and pick up where it left off. To maintain all of this data, the program maintains a directory known as the "Working Directory" or "Working Folder", which contains the necessary data and subdirectories for operating on the images.

The working directory contains the following items. With the exception of the data file, all items are subdirectories and are automatically generated by the program if not already present.

- ***Data File.*** A regular file in the comma-separated values (csv) format explicitly named "data.csv" which must be present before the program can begin. It must contain the following columns.
    - Section: Any string
    - Row: A positive integer
    - Site: A positive integer
    - Side: Either the string "F" or the string "B"
    - First Name: Any string
    - Middle Name: Any string
    - Surname: Any string
    - Conflict: A string from the set {CIVIL WAR, SPANISH AMERICAN WAR, WWI, WWII, KOREA, VIETNAM}, or a concatenation of two of those strings with the string " & " between them.
    - Birth Date: A date in the form "YYYY-MM-DD"
    - Death Date: A date in the form "YYYY-MM-DD"

    - Fuzziness Score: Leave blank. If not present will be appended by the program.
- ***Image Folder.*** A folder named "images" containing the images to be processed.
- ***Destination Folder.*** A folder named "complete" to which all successfully processed images will be sent once appropriately rotated, cropped, and labeled.
- ***Error Folder.*** A folder named "errors" to which all of the erroneous images will be sent if there is an error with the rotation or cropping processes.
- ***Feedback Folder.*** A folder named "feedback" to which all of the processed images will be sent if the program is unable to label them and requires user assistance to do so.
- ***Processed Originals Folder.*** A folder named "processed_originals" to which all of the original images will be sent after they have been successfully processed and their processed copies have been moved to either the destination folder or the feedback folder.

Program Overview

The general sequence of operation for the program is as follows:

- Initialize the program
    - Determine working directory, create missing subdirectories
    - Load the process queue, which is just an array of strings that are paths to the files in the image folder, using the glob python library[4].
    - Load the feedback queue, which is just an array of strings that are the paths to files in the feedback folder, using the glob python library.
- Iterate over each image path in the process queue
    - Load the image as a numpy array
    - Process the image (Rotate, Crop, OCR)
    - If erroneous, move original to the error folder and create a metadata file for it. Otherwise, move original to the processed originals folder.
    - If feedback is required for labeling, move the processed version to the feedback folder. Otherwise, label and move the processed version to the destination folder.
    - Check if the user has aborted the program. If so, stop. Otherwise, update the user interface to reflect the progress.

Note that the feedback system is controlled through the user interface directly and not the driver.

Global Vars

Certain pieces of information regarding the fundamental program operation need to be accessed from a wide variety of places in the program. In particular, the different files and locations detailed in the directory structure need to be accessed by the driver, the initialization screen, and the feedback screen. The runtime options that control the labeling system (See: “Labeling System: Runtime Options”) need to be accessed by the initialization, the labeling system, and the feedback screen.

Most of this information is created during initialization or the beginning of the driver operation, but is needed elsewhere. The user interface is created using the Tkinter python library (See: “User Interface: Implementation”), and passing a large amount of information between the classes thereof can be tricky, confusing, and error prone. Additionally, each class that needs the information will need to copy it and save it as a local instance variable, but then if a different part of the program needs to update the information, this change wouldn’t get carried over.

To solve this issue, the program will use a class called Global_Vars to store all of this information. An instance of this class is initialized during the __init__.py of the program’s shared package. All classes that access this package can then access the same Global_Vars object, and we don’t need to worry about passing it around.

Error Handling

In the course of the program’s operation, it is natural to expect periodic errors to occur that could disrupt the function of the program if not appropriately handled. There are two types of errors that may occur during the operation of the program: Processing errors and labeling errors.

Processing errors are catastrophic failings that occur during the rotation or cropping sections of the program. They happen when the neural networks undergirding these sections encounter an image that for whatever reason they are unequipped to handle. Processing errors are not resolvable. There is no deterministic way to fix a processing error after it has happened. Processing errors may also sometimes not be detected. If the cropping section over-crops the image and cuts off essential parts, for example, the problem will likely not be detected until the program requests for user assistance in the labeling section. Images that suffer from processing errors are moved to the error folder.

Labeling errors occur when the labeling section is not able to properly match an image to an entry in the data file, which could happen for a variety of reasons. Labeling errors are

identified by the labeling system and are resolved by requesting user feedback by way of the feedback portion of the user interface. For more information, see “User Interface: Feedback” and “Labeling System”.

Processing errors and labeling errors are actually fundamentally different in that processing errors are caused by the processing steps, whereas labeling errors are detected by the labeling step. This is part of why they are differently resolvable.

Image Metadata

In the case of either error, however, it is useful to save some information related to the nature of the error when it was detected. For processing errors, this metadata can help in bug testing and in understanding what exactly went wrong. For labeling errors, the primary metadata to save is the information used for the labeling (which was generated by the OCR section) so that it doesn’t have to be regenerated later. Even in the event of a successfully labeled image, it is useful to store information about it in case we need to reference it later.

All of the images are expected to have unique filenames before the operation of the program (otherwise they couldn’t all be in the same folder). To save metadata about the images, the image’s file extension (typically “.JPG”) is replaced with “.txt”. The resulting filename is used for a text file to contain any important information. Additionally, these files are marked as “hidden” so that they don’t appear when the user opens up the folders to look at the images.

The first line of the metadata file is a single string that identifies what happened to the image during processing and determines the information contained on the subsequent lines. It will be one of the following strings.

- ***rotation_error.*** The rotation section encountered a fatal error and the image processing could not continue. Subsequent lines may contain diagnostic information for bug testing.
- ***cropping_error.*** The cropping section encountered a fatal error and the image processing could not continue. Subsequent lines may contain diagnostic information for bug testing.
- ***ocr_error.*** The OCR section encountered a fatal error and the headstone text could not be evaluated. The image may continue to the labeling section, but there will be no matches and user feedback will be required. Subsequent lines may contain diagnostic information for bug testing.

- ***success.*** The image was successfully processed and the headstone text was evaluated. Subsequent lines contain the headstone text evaluated by the OCR section.

A different way to store this metadata is within the image itself. By using the pyexiv2.ImageMetadata function in the pyexiv2 python library[5], you can add metadata tags to images using the JSON format. This would be advantageous in that it would be stored within the image itself in a single unified file, rather than have two separate files floating around which could get separated. Indeed, this was our original idea for how to store this data. We ultimately decided not to use this approach because once an image encounters an error, it doesn't need to be moved between folders unless the error is resolved. That is to say, once an image encounters a processing error, it will forever be in the errors folder unless the error is somehow resolved, but if the error is resolved we wouldn't need to store the metadata about the error anymore. Similarly, if an image encounters a labeling error it will be in the feedback folder until it receives the feedback it needs, at which point the metadata, which contains information necessary for the feedback, is no longer needed. Ultimately, storing the metadata inside the image itself was considered more complicated than just using a separate file with a similar file name.

Error Reprocessing

Our previous version of our error handling system also included a feature called "error reprocessing". Essentially, if the error reprocessing option is selected during the program's initialization, the error folder would be treated as the image folder. Since all of these images would have encountered an error, they would have metadata detailing what went wrong. This would allow the program to "jump ahead" past all of the work that had already been done, and just revisit the problem areas. The primary advantage of this system was so that images that encountered labeling errors wouldn't have to go through the processing sections again, and could jump directly back into the labeling section. Ultimately, the system proved too complicated and confusing, and simply splitting the error folder and feedback folder into separate pieces (they were originally the same folder, and the contents were differentiated by the metadata), was a much easier solution that solved a few other challenges as well.

Why Not Just Use a Database?

This project requires the loading of lots of gravesite information for use in the labeling system as well as loading many, many images and manipulating information related to them. When managing this much data, usually the best course of action is to use a database of some kind, but we are opting instead to use this directory structure system. Why?

The main reason is that a database would make the system significantly less user friendly. Since we've decided not to make a web application and host it on a registered domain, and instead use an all-in-one local program, a large portion of the management of the database would fall on the user. Pyinstaller can't compile the database and database system, and loading the data into the database would require first loading it into the program anyway. Additionally, we'd also have to get the labeled images out of the database for the user to be able to access them.

By simply using a single directory with descriptive sub-element names, the process will be much more transparent to the user. The user will be able to see in real time as their images are labeled and moved from one folder to another. The system is much more readily intuitive, and the user has complete control over what data the program is working on at any given time.

The strongest argument in favor of the database is that it would be much better suited for storing metadata than the accompanying metadata files detailed above, which are admittedly a bit of a clunky solution. However, after a few prototypes of either solution, we decided that managing the files is ultimately still less cumbersome than managing a database for this particular issue.

User Interface

Objectives

The ultimate purpose of this program is to manipulate a collection of images such that they are all appropriately rotated, cropped, and labeled based on information gleaned from computer vision processes applied to them. More broadly, the purpose of the program is to provide individuals within the UCF Department of Humanities and the National Cemetery Administration with a streamlined mechanism to automate these tasks so that they don't need to perform them manually.

For this reason, it is not only important for the system to be able to perform a series of purely technical operations, but also for it to present a friendly visual interface that makes the user experience as easy as possible. The user interface portion of the system exists as the link between the intentions of the user and the technical operation of the system.

While the program is designed to be as automated as possible, user interaction is necessary in two critical areas. First, the program needs to secure a set of parameters that govern where it can find the images to be processed, the data used to process them, and the location where the user wishes for the end results to be placed; in other words, the user must define the location of the program's working directory on the system. Second, the program will need to secure assistance from the user to help in situations that cannot be algorithmically determined; namely, the user must assist when the program is not able to definitively label an image.

Additionally, there is a level of variability in how the program can operate. This variability has a meaningful impact on the user experience, and so the user needs to be able to specify the manner in which they would like for the program to execute.

Finally, the program must indicate to the user that progress is being made, and provide some estimation of that progress. More specifically, by informing the user of how many images have been processed and how many remain.

For these reasons, the objectives for the user interface portion of the program is as follows:

- ***Intuitiveness.*** Operation of all portions of the user interface needs to be an intuitive user experience wherein every option and feature available to the user feels as if it makes sense. Users should be able to reliably operate the interface without extensive prior instruction.
- ***Initialization.*** The user interface must retrieve from the user all information necessary for the operation of the program before the processing of images.

- ***Variability.*** The user interface must provide the user with a set of options to control the operation of the program in a way that suits their needs.
- ***Feedback.*** The user interface must provide a mechanism to retrieve additional information and advice from the user whenever such information cannot be determined otherwise.
- ***Continuous Operation.*** Wherever possible, the user interface is not to interrupt the processing of images or create a bottleneck.

Program Launch

For this project, we are electing to deploy the finished final version in the most simplified way possible. A simple utility that takes images as input and produces modified versions of those images as output whose intended users are a small set of people working in one department does not need a large flashy scalable deployment on a remote-hosted cloud cluster or the like. Nor does it even need to run as a web application hosted locally.

The simplest, most straight-forward deployment solution we could determine for this project is to compile the underlying code all down into a single Windows executable file. This file can be trivially shared either via physical USB drives or over web-based storage solutions such as Dropbox or Google Drive. To run the application, the user needs only to download the single file to their local Windows device and run it as they would any other executable. The initialization screen of the user interface will pop up thereafter, and the user can begin processing images quickly and easily.

To convert python code into a Windows executable we will be using the pyinstaller python library[6]. The resulting executable does contain a significant amount of code, including all the python libraries used in the system as well as the python interpreter itself. While this does mean that the executable is going to be relatively large (though we still predict it will be under one gigabyte), the user does not need to do any set up beyond downloading the file and clicking on it; they don't even need to have python installed on their machine.

User Authentication

In order to protect sensitive information or restrict powerful operations to a privileged subset of users, many applications employ a system of user authentication. This authentication process, typically performed with a simple username-password combination, requires an additional level of user interaction and an additional overhead of storing and protecting the keys, passwords, or the like.

This system does not store sensitive information, nor does it have special operations that could be destructive if misused. This system is not to be hosted in the cloud or made publicly accessible, and the images which the program will process are assumed to be backed up elsewhere. Suffice to say, this system has a very limited operational scope and is relatively incapable of inflicting meaningful damage on a computer system. For these reasons, this system will have no user authentication process.

Without user authentication, the system doesn't need to store the authentication credentials or send the user through an authentication screen. There will not be multiple user levels; there will only be a single user type, which is anybody that launches the program. Without this additional overhead, the program can better keep true to its "intuitiveness" objective.

Implementation

The user interface for the program will be implemented using the Tkinter python library[7]. In Tkinter, we begin by creating a Tkinter root window in which we can place all of the other frames and widgets that will compose our screens. To maximize the ease of operation for the program and to smooth out the development process, the frames and widgets should be designed to be as modular as possible so that we can simply add or subtract them as needed.

As with most user interface packages in most languages, Tkinter is extremely picky about starting and ending on the main program thread. This means that our original idea of spinning up feedback windows whenever the program encounters an image that needs them wasn't going to work. Instead, the main operation of the program will actually be to create the root window object, start the main driver function on a separate thread (passing the main window as an argument), and then to run the main window's mainloop() function on the main thread. From this point all of the operation of the program will take place either by the methods of the Tkinter classes that we create, or in the driver loop. Most notably, the driver can't destroy any of the Tkinter classes; the only real way to safely close the user interface is for the user to press a button on the user interface itself.

The driver will need to manipulate the user interface in certain ways. Specifically, it will need to start the initialization screen, wait for initialization to finish, then start the processing screen before beginning the primary driver loop to process the images. To perform this waiting operation, we can use a lock object from the python threading library.

While processing the images, the driver will need to send signals to the processing screen so that it can update the information it's displaying. To do this, we will need to implement an update() function on the processing screen that can be called by the driver, which

prompts the processing screen to look up the information it needs in the Global_Vars object. The feedback screen can just perform updates whenever the user provides feedback, or it can also implement an update() function. Presently, we plan to just update on feedback, but we may change this if we determine a sufficiently compelling reason to do otherwise.

Each of the user interface screens will consist of a series of nested Tkinter frames, which contain within them more frames and/or widgets. Each screen is represented by a single class that inherits from Tkinter.Frame. Within the screen's initialization, each of the subframes within it are implemented as their own classes, which also inherit from Tkinter.Frame. This allows for the screen to operate at a very high level without having to handle the details of all of the subcomponents of the subframes, or the methods that operate on their data. Each of the screen descriptions below detail the composition of its subframes and widgets. In these subframe structure breakdowns, anything not expressly labeled as a particular widget is a frame, and anything not expressly labeled as ordered from left to right is order from top to bottom.

Style

In order to create a visually appealing and visually consistent user experience across multiple screens, the user interface will employ the use of a globally-accessible Color_Pallette class. The class contains the colors for foreground, background, two accent colors, and a debug color for error messages. These colors can be determined using a color palette, but can be easily changed in one place, the Color_Pallette class, and have that change propagate across all of the frames in all of the screens. At present, the current color palette to be used is: foreground: #7EC8E3, background: #050A30, accent 1: #000C66, accent 2: #0000FF, debug: #FF0000.

UI Screens

The user interface for the system will consist of three screens: the initialization screen, the processing screen, and the feedback screen. On program launch, the user will see the initialization screen, from which they will begin the primary program operation. During this operation, the user can toggle back and forth between the processing screen and the feedback screen, as demonstrated by the state machine below. Note that all transitions in the machine are controlled by the user, with the notable exception of transitioning from feedback to processing automatically when the feedback queue is emptied. A graphic that visualizes the screen transitions is shown below.

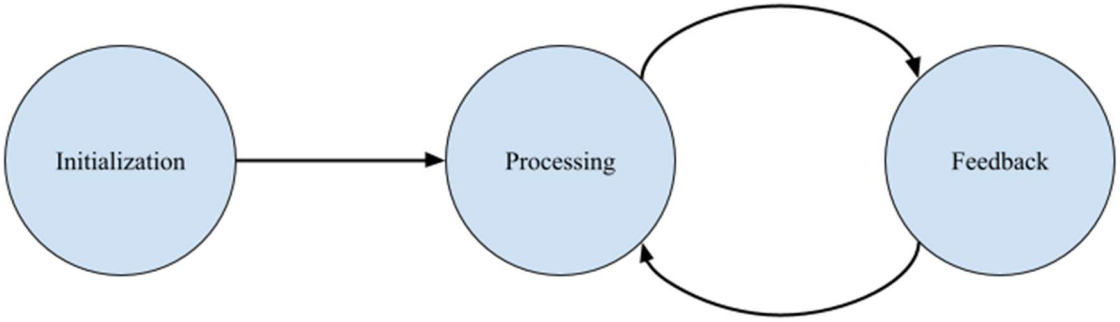

Figure 2: UI Screens

Switching Between Screens

The initialization screen occurs exactly once in the program's operation, and nothing can continue until it is complete. As such, the transition from the initialization screen to the processing screen can be handled by the driver. Switching between processing and feedback, however, is completely driven by the user and should not interrupt the driver's actions, so it will have to work differently.

In Tkinter, a frame or widget is visible in the window once it has been "packed". It is entirely possible to construct a frame or widget object and utilize its methods without ever packing it. Additionally, a frame or widget can be unpacked with the pack forget() method. This removes it from the window without destroying the underlying object.

Once initialization is complete, the screen is no longer needed. The object reference does not need to be saved and it can be deallocated. The driver can create the initialization screen object, pack it, wait for it to complete, then create the processing screen object and pack it. When the processing screen is constructed, it creates a feedback screen but does not pack it, passing along a reference to itself. Both screens then store a reference to the other one. Later, when the user wishes to switch from one screen to the other, the screen unpacks itself and then packs the other. The driver never actually holds a reference to the feedback screen.

Initialization

When the user runs the program from the executable, they will first see the initialization screen. The user will need to specify the location of the program's working directory (See: "Program Driver: Directory Structure"). The starting location of the file manager will be the current directory in which the program executable is located.

The initialization screen will also have several options available to the user to change the manner in which the program labels the images after processing them. These can be used to fine-tune the system to the user's needs at the time of execution. For more information on the labeling options, see the "Labeling System: Runtime Options" section.

When the user has selected their desired settings, they press the "Process Images" button. The program then verifies that the settings are valid. The values entered for the text-entry options are validated as appropriate. The working folder must exist on the system, and it must contain a file named "data.csv", which must contain the column headers specified in the label format. If there are any issues encountered during this verification process, the program issues an error message and the initialization screen remains active for the user to correct the mistakes. Otherwise, the program loads the feedback queue and begins processing the images while the screen changes to the processing screen.

An earlier version of the initialization screen was structured such that the user had to select each of the contents of the working directory separately. This was to allow more customization for where the program's information was stored, but ultimately proved too clunky once the number of these elements became too great. While highly customizable, the implementation was not intuitive, and so we instead opted for this "all in one place" solution. This version also included a checkbox for "Error Reprocessing", which proved obsolete after reevaluating how the program would handle errors (See: "Program Driver: Error Handling: Error Reprocessing")

A visualization of the initialization screen is shown below.

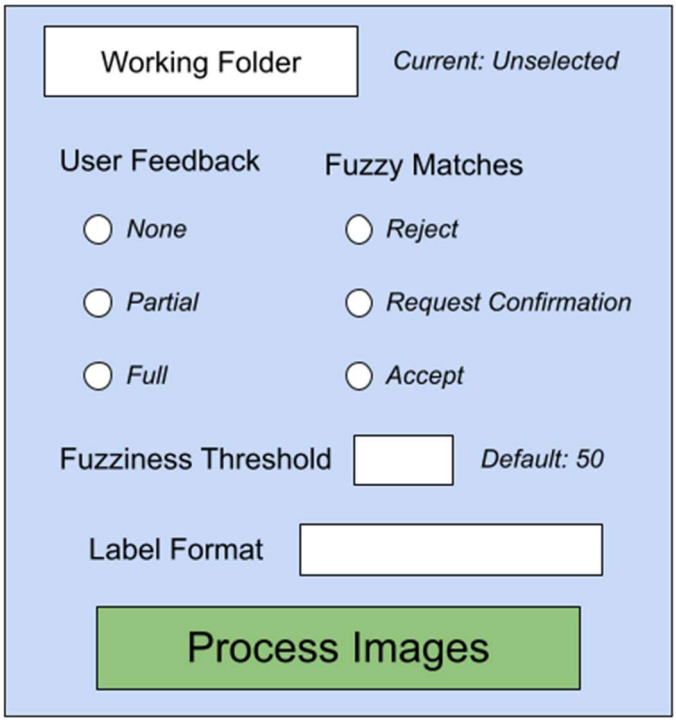

Figure 3: Initialization Screen

The subframe structure of the initialization screen is:

- Parameter Section (left to right)
  - Working Folder (Button)
  - Current (Label)
- Radio Section (left to right)
  - User Feedback
    - (Label)
    - (Radio Button) x3
  - Fuzzy Matches
    - (Label)
    - (Radio Button) x3
- Fuzziness Threshold (left to right)
  - (Label)
  - (Text Entry Box)
  - Default (Label)
- Label Format (left to right)
  - (Label)
  - (Text Entry Box)
- Process Images (Button)

Processing

While the program is running, the user can see the processing screen. On it, a counter informs the user of how many images have been processed so far and the total number of images to be processed during this execution of the program (which is the number of images in the Images Folder during initialization). A separate counter informs the user of the number of images in the feedback queue, waiting for the user to provide manual feedback for their labeling.

If the user wishes to provide feedback on the images in the feedback queue, thus labeling them and completing them, they can press the "Provide Feedback" button, which will take them to the feedback screen.

Should the user wish to stop the program early before it has finished processing all images, they can safely do so by pressing the "Abort Processing" button. After doing so, the program finishes processing the currently-processing image, safely closes any open files (such as the data file), and then stops.

The fundamental purpose for this screen is simply to provide the user with some information regarding the progress of the program, so that they can have some degree of assurance that the program is actually working properly. If the user wishes to change some of the program's options or its working directory, they will have to abort and reinitialize.

One of the key functions within the processing screen class is the update() function, which is not called as a result of pressing any of the buttons. Instead, the update() function is called by the driver class, outside the user interface. It causes the processing screen to check the numbers for which image it's processing and how long the feedback queue is, which may have changed. This allows for live updates for the user.

A visualization of the processing screen can be seen below.

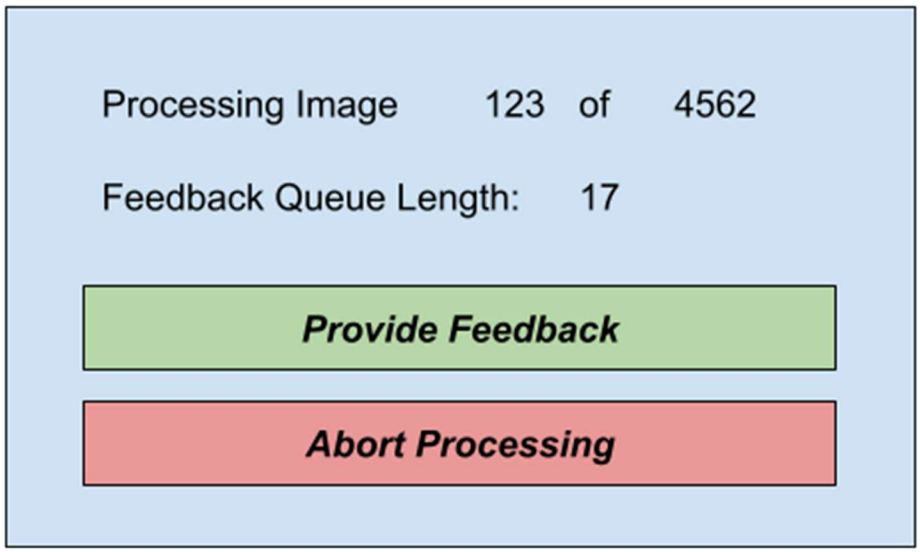

Figure 4: Processing Screen

The structure of the processing screen is just two labels and two buttons. It's simple enough that it doesn't really need any subframes.

Feedback

While the entire point of the system is to automate the process of manipulating and labeling images specifically so that a human doesn't need to do it manually, it is unreasonable to assume the system is going to get things right 100% of the time. Whenever an image is processed that the program cannot properly label, we must turn to the user to provide feedback to the system so that it can label accordingly. With this in mind, the process should still be as smooth and time-saving as possible.

Possible reasons for needing feedback include:

- ***No matches.*** None of the entries in the data file match the image text within acceptable tolerances.
- ***Multiple matches.*** Multiple entries in the data file match the image text equally and the program needs to know which one to use.
- ***Confirm fuzzy match.*** One entry in the data file matches the image text better than any other and is within acceptable tolerances, but is not a perfect match, and the user has selected that the program explicitly confirms all fuzzy matches before labeling images.

Continuous Operation

Introducing a user feedback component into an automated process necessarily raises the concern that the runtime of the program becomes dependent upon how fast the user responds to feedback requests. Ideally, the program should be able to continue operation even while waiting for the user to respond; otherwise, the processing of thousands of images would take an undesirably long time.

To solve this, the program will maintain a "feedback queue" of all the images that could not be labeled properly. Whenever the program encounters such an image, the image and any relevant information (such as the text determined by the OCR section) are placed into the queue and the program continues processing the next image. The feedback system is then loaded on a separate thread from the main system, which continues on to the next image while waiting for the user. In actual fact, the queue is simply a list of filenames of files in the feedback folder, which are loaded as-needed while the user is providing feedback.

An important reason as to why this wouldn't result in major slowdowns is because feedback is really only necessary for the labeling system, not rotation, cropping, or OCR. This means all of the image processing and manipulation is over and all that remains is string manipulation and data file searches. These operations are reasonably fast and inexpensive relative to the computational complexity of the image processing components; the bulk of the program's computational load will reside on the main processing thread, not the feedback thread.

Feedback Screen

On the Feedback Screen, the user will see both the original image and the processed image as well as a curated set of possible matches, which are themselves buttons. Each match includes the name, conflict, birth date, death date, row, and site, as pulled from the data file (See: "Labeling System" for a full description for how these matches are determined). Simply clicking on the match selects it for use in labeling the image.

A series of text-entry boxes on the right side of the screen display the information that the program has been able to extract from the image and use for matching. These entry boxes are editable, and if the user clicks the "Search" button under them the "Possible Matches" section is repopulated with whatever information is in the boxes. This is necessary if the OCR section was not able to extract the proper information and the correct match is not present on the screen.

If a critical error occurred during the processing, such as if the image was rotated or cropped incorrectly, the user can check the "Flag as Erroneous" option to indicate as much. In this event or if the user cannot find the correct entry for the image, they can click the "Abandon Image" button. If the image was flagged as erroneous, it is moved to the error folder once abandoned. Otherwise, it is merely removed from the feedback queue and left in the feedback folder.

At any time, the user can click the "Return" button to stop providing feedback and return to the processing screen.

A visualization of the feedback screen is shown below.

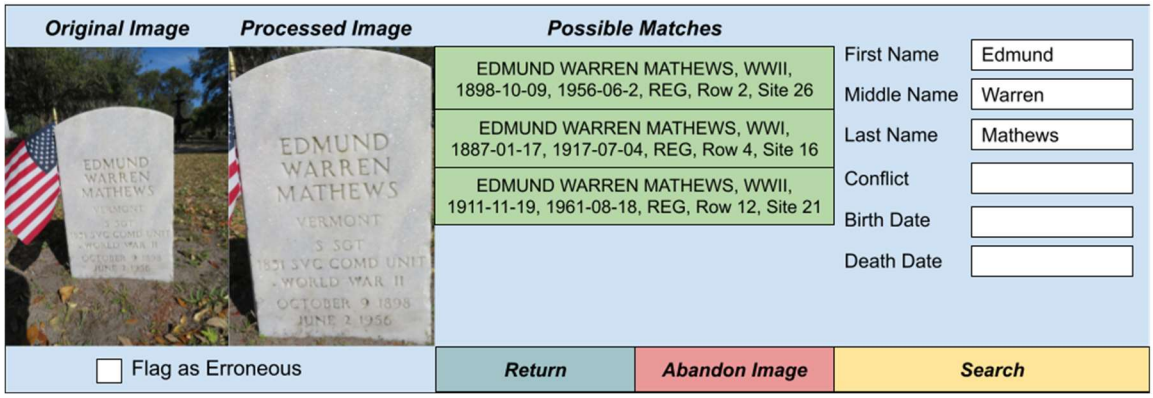

Figure 5: Feedback Screen with Multiple Perfect Matches

Subframe Structure

The subframe structure of the feedback screen is: (left to right at top level)

- Pictures
    - Images (left to right)
        - Single-Image x2
            - (Label)
            - (Picture)
    - Flag (left to right)
        - (Checkbox)
        - (Label)
- Matches
    - (Label)
    - Matches
        - (Button) x[0-5]
    - Control Buttons (left to right, pack on bottom)
        - Return (Button)
        - Abandon (Button)
- Search
    - Label-Entry Pair (left to right) x6
        - (Label)
        - (Text Entry)
    - Search (Button) (pack on bottom)

Rotation

Introduction

The first step that the program must perform is the rotation of the images given as input. As per Dr. Giroux's specifications, the largest number of images will be 6,000 and the maximum amount of data the program will be dealing with is 32 GB. The image rotation will occur in two steps: the first being the macro rotation and the second being the micro rotation. The macro rotation aims to ensure that the image is upright and not on its side or upside down. The micro rotation focuses ensuring the image isn't tilted even in the slighted direction other than straight, or as close to straight as possible. Two models will be created and trained on gravesite images and be tested on images given from Dr. Giroux.

The most important goal of the rotation step is to prepare the image for text extraction during the OCR portion. Text can't be extracted if the image is upside down or rotated on its side. This is the most basic step of the overall product as it only focuses on the orientation of the input while later steps focus on more specific features.

Background Research

Edge Detection

Edge Detection is the process that attempts to characterize the intensity changes in the image in terms of the physical process that has originated them.

The process of edge detection begins with the raw image. Everything in the image must be captured, including the three-dimensional objects in the frame. This ensures all the raw values are captured in the frame. When looking at the image, the most important properties are the actual physical edges. This is because the physical edges are representative of the actual physical and changes occurring in the image. The ultimate goal of edge detection is to find these intensity changes in the actual physical changes in the image. One example of a physical change that can occur in the image are shadows. Shadows can be detected from intensity changes in light. Another example of a physical change that occurs is change in color, which is just an extension of the previously mentioned ideas.

There are two steps that occur in the process of edge detection:

- Identifying intensity changes in the image

- Using the representation of the intensity changes along with high-level knowledge to make assertions about the 3-D surfaces and their properties

The first step is a problem of numerical differentiation. The derivatives of the image intensities (which may be of different orders and type) need to be evaluated. However, before differentiation occurs, the issue of well-posedness needs to be taken care of. This is because in robot vision, noisy data is unavoidable. Noisy data means that the images have random variations of brightness scattered throughout which makes image detection much more complicated. The cause of this is the limitations of real time and the need for high temporal and spatial resolution. Any operations performed on the data must not be sensitive to noise as a result. To take care of this, we must first ensure that the differentiation is well posed. In mathematics, a problem is well posed if its solution:

- Exists
- Is unique
- Depends continuously on the initial data

Differentiation of a function is not well posed because it can be seen as the solution to

$$g(x) = Af(x)$$

Equation 1: Definition of Well-Posed Function

where *Af(x)* is the integral operator.

Regularization techniques

One method of transforming an ill-posed problem into a well-posed one is regularization. Regularization is defined as adding information, in this case, the regularization term λ. The goal of regularization is to find x in the data y such that Az = y. This requires finding suitable norms(usually quadratic) and a stabilizing function. Three main methods of standard regularization exist:

- Among z that satisfy $||P_z|| < C_1$ (where C_1 is a constant) find a function z that minimizes

$$\|A_z - y\|$$

Equation 2: Regularization Equation 1

- Among z that satisfy $||A_z - y|| < C_2$ find z that minimizes

$$||P_z||$$

Equation 3: Regularization Equation 2

- Find z that minimizes

$$||Az - y||^2 + \lambda||Pz||^2$$

Equation 4: Regularization Equation 3

The first method best approximates the data while working within the given constraint. The second method computes the function z that is close to the data and is the most "regular". The third method tries to find a compromise between the regularity of the solution and the closeness of the data using λ.[8]

Filtering

The purpose of filtering an image is to enhance the quality of the edges in the image or sharpening. Filtering takes place after all possible noise is removed.

Changes in intensities and the properties associated with these changes can be characterized by the point at which a derivative crosse the x-axis, or the zeroes of the derivative. An example of this can be seen with step edges and roof edges. Step edges are edges in the image where the change in intensity is instantaneous and changes from one intensity to the complete opposite. Roof edges are edges where the intensity changes overtime and returns to the initial value. Step edges correspond to the extrema of the extrema of the first derivative and roof edges correspond to the zeroes of the first derivative. The idea behind the filtering stage is to create a representation of these zeroes and extremes. This is done so that the different boundaries of the image can be shown and to create a higher contrast between the edges.

Figure 6: Ridge Function Graph

Figure 7: Step Function Graph

The shape of step edge, S(x), is defined as:

$$S(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$$

Equation 5: Step Edge Function

The output of the filter f(x) convoluted with the step edge S(x), g(x) will be defined as:

$$g(x) = F(x) - F(-\infty)$$

Equation 6: Convolution Result

where F(x) is the integral of f(x). This shows that:

The extrema of g(x) correspond to the zeroes of f(x)

The zero crossings of the second derivative of g(x) correspond to the extrema of f(x)

Fourier Transform

About the Wigner Distribution Function

A Fourier Transform is a mathematical transform that decomposes functions based in space or time into functions based on spatial or temporal frequency. It is central to the field of physical optics such that anything related to optics is most likely related to the Fourier Transform in one form or another. The idea of a fractional Fourier Transform is to take a piece graded index fiber and into cut it into fractional pieces. These pieces with length *PL (P < 1),* where *L* is the total length of the fiber, would perform the fractional Fourier Transform on an input image. Adding two pieces together would result to a fractional transform of degree

$$\boldsymbol{P_1 + P_2 = P}$$

Equation 7: Factional Transform with Two Parts

Before moving forward, another mathematical concept must be defined. The Wigner Distribution Function (WDF) is a transform in time-frequency analysis. Mathematically, it is defined as

$$W(x,v) = \int u\left(x + \frac{x'}{2}\right) * u * \left(x - \frac{x'}{2}\right) e^{-2\pi x' v} dx'$$

Equation 8: Wigner Distribution Function

Inserting the Fourier representation of the signal *u(x):*

$$u(x) = \int u'(v) e^{2\pi i v x} dv$$

$$W(x,v) = \int u'\left(v + \frac{v'}{2}\right) * u' * \left(v - \frac{v'}{2}\right) e^{2\pi i v' x} dv'$$

Equation 9: WDF with Fourier Representation

Image rotation based on shearing

Rotation of an image can be described in the form of polar coordinates like so:

$$I(r,\theta) \rightarrow I(r,\theta + \phi)$$

Equation 10: Rotation of an Image in Polar Form

An alternative rotation exists that can be expanded to four dimensions:

$$(x_0, y_0) \rightarrow (x_0 - Ay_0, y_0) = (x_1, y_1)$$
$$(x_1, y_1) \rightarrow (x_1 y_1 - Bx_1) = (x_2, y_2)$$
$$(x_2, y_2) \rightarrow (x_2 - Cy_2, y_2) = (x_3, y_3)$$

Equation 11: Four-Dimensional Fourier Transform Equation

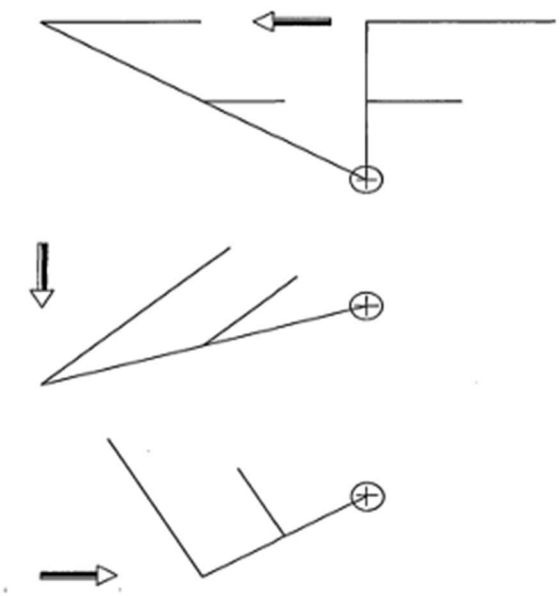

Figure 8: Fourier Transform Visual Representation[9]

LeNet-5

The LeNet-5 architecture is a CNN architecture described by Yann LeCun in 1998 in their paper titled "Gradient-Based Learning Applied to Document Recognition". The architecture uses 32 x 32 shaped images with one color channel (grayscale) as input and works is constructed like this:

- A Convolutional layer that outputs 28 x 28 feature maps using six filters
- An Average Pooling layer that outputs 14 x 14 feature maps using six filters
- A Convolutional layer that outputs 10 x 10 feature maps using 16 filters
- An Average Pooling layer that outputs 5 x 5 feature maps using 16 filters
- A Flatten layer to reduce back to one dimension
- A Dense layer containing 120 units with a Rectified Linear Unit (ReLU) activation function
- A Dense layer containing 84 units with a ReLU activation function
- An output layer containing 10 units for 10 class classification

If implemented for rotation classification, the output layer would contain four and 11 units for the micro and macro rotation respectively. To avoid the vanishing gradient problem, which is an issue that comes from using activation functions that push the gradient values closer and closer to zero, the ReLU activation function (see AlexNet for more details) would be used for all layers. This architecture uses Average Pooling which takes the

average of a set of pixels and creates an output with each average. This creates a smaller image with important features such as edges being more emphasized. However, since the average of the pixels is used, while the features may be smoother, they are not as distinct. For testing this architecture for macro rotation, Max Pooling will be used to highlight features such as edges which is key to identifying what angle a headstone needs to be rotated at.

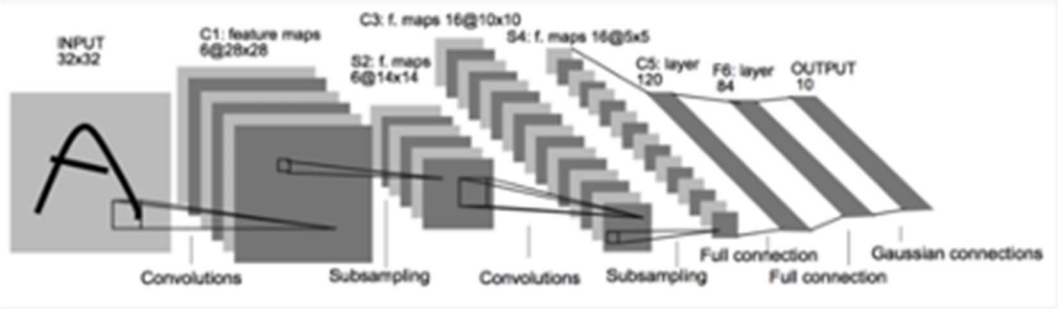

Figure 9: LeNet-5 Architecture

LeNet-5 is being considered due to its high accuracy (99.2%) on the MNIST standard dataset. The MNIST standard dataset is a dataset that contains 60,000, 28 x 28 sized images of handwritten digits ranging from zero to nine. The kernels used for this architecture recognize line patters (horizontal, vertical, diagonal) which would prove useful for rotation classification. For example, a headstone that was rotated 90 degrees counterclockwise would have two parallel horizontal lines with a curve near the left side of the image. If convolved with a filter that detected horizontal lines, the horizontal lines would be emphasized, thus narrowing down the decision to either 90- or 270-degree rotation.

While easy to implement due to its simplistic design and smaller structure, LeNet-5 does have some shortcomings. The LeNet-5 model can easily overfit due to its simplicity. Overfitting is when a model fits too well to a training set and as a result, fails to generalize on unseen data.

AlexNet

AlexNet is a neural network architecture that was popularized in the paper “ImageNet Classification with Deep Convolutional Neural Networks”. The architecture is notable for adding innovations that are standards today.

One innovation was the addition of the ReLU function. The ReLU function returns the max of 0 and the input value. This function takes care of a common issue in Deep Learning

which is the vanishing gradient problem. The vanishing gradient problem is when gradients are back propagated through the network, when they are passed through activation functions such as Tanh and sigmoid, gradients that tend closer to positive and negative infinity result in smaller and smaller outputs. This essentially kills the learning process as a weight with a zero value can't be improved. With ReLU, negative inputs result in an output of zero and positive inputs just return the input.

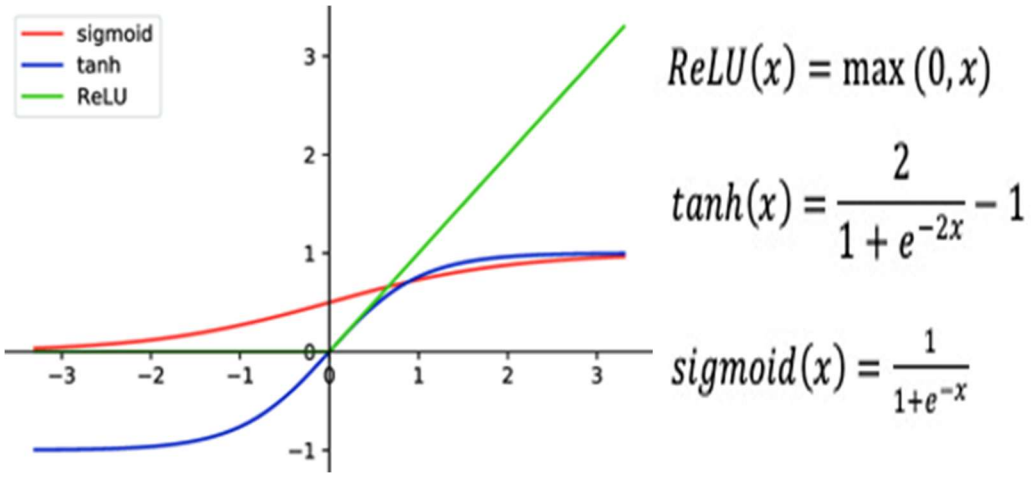

Figure 10: Graph of Activation Functions with Equations

Another change that was made was the addition of Max Pooling layers as opposed to Average Pooling layers. Both methods of pooling are similar but differ because Max Pooling extracts more prominent features. To address the overfitting issue of the LeNet-5, another type of layer, called the dropout layer, was added. The dropout layer will take a percentage as input and drop that percentage of neurons selected at random. Dropout is implemented at the end of the model between fully connected dense layers. Dropout fixes the issue of overfitting by setting a percentage of the inputs to zero. In practice, this is the difference between one and the input. (For example, if the input to the dropout layer is 80% or 0.8, 20% of the input gets set to zero).

For input, the model takes in images with a shape of (224, 224, 3). These are 224 x 224 images with three color channels. The model consists of five convolutional layers (feature detection) and three dense layers (classification). This differs from Lenet-5 in that there are more convolutional layers but has the same number of dense layers at the end. Also, like Lenet-5 is the pattern of increasing the number of filters. In order, the sizes are 96, 256, 384, 384, and 256. However, the kernel size of the model decreases with each layer. The pattern is 11 x 11 to 5 x5 and then decreasing to 3 x 3. Along with the pattern of a pooling

layer following a convolutional layer, AlexNet introduces the concept of two convolutional layers was also used. Both patterns are widely used today.

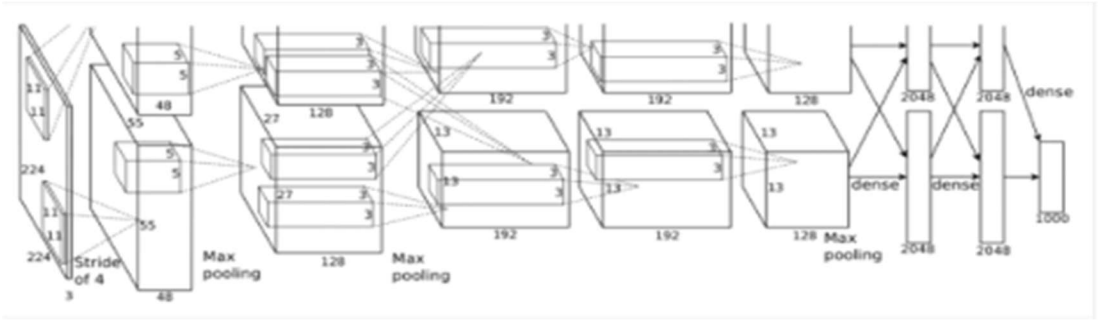

Figure 11: AlexNet Architecture

VGG

The VGG architecture (named after the lab it was developed in, Visual Geometry Group), was developed to further the advancement of neural networks through deeper networks.

The VGG model differentiates itself from other architectures in a couple of ways. Firstly, instead of using a smaller number of large kernels, VGG uses a large amount of smaller 3 x 3 and 1 x 1 filters. The architecture will also stack anywhere from two to four convolutional layers before capping off with a Max Pooling layer. The reasoning behind this was that more layers stacked together with smaller kernels achieved the same effect as one convolutional layer with a larger kernel size. The number of filters starts at 64 and begins to increase to 128, 256 and then finally 512. There are different variants of the architecture that vary in depth: The VCG-16 and the VCG-19.[10]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 12: Comparison of VCG Architectures

Overview of Rotation Algorithm

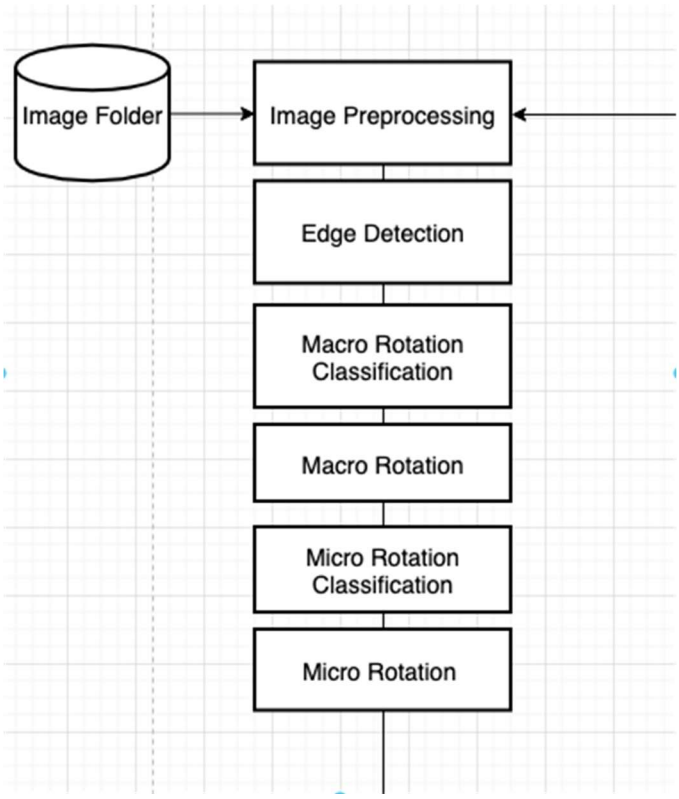

Figure 13: Diagram of Rotation Algorithm

Context of rotation

The first step of the overall project will be the user selecting the images they want processed and selecting the file associated with those images. The Rotation component will receive this file as input. For training, these images will be labeled according to the type of rotation that needs to be completed. When the component is in use, the algorithm will receive images as input and attempt to classify what type of rotations need to be completed. Once the rotations are classified, they are saved and used as input for a rotation function.

Image Preprocessing

Before any computations can be made on the images must be preprocessed. The purpose of the preprocessing stage is to get rid of any noise that any of the images can contain. The operations that will be performed on the images will involve the calculation of gradients and derivatives which are both sensitive to noise. Noise is the random variations in brightness that occur in an image. The goal is to capture the edges of the gravestone to rotate it appropriately. If there is noise, these intensity changes can be harder to spot, thus making the edge detection less productive.

Another step in the preprocessing stage is to convert all the color images to grayscale. This is because edge detection algorithms such as Canny only work with grayscale images as well as the fact that grayscale is all that is needed. This is because grayscale image pixels only measure intensity of light (values from 0 to 255). Color image pixels measure three different types of intensity: the intensity of red, blue and green light (all values from 0 to 255). So, while a color pixel will have a measurement of (132, 200, 11) for example, grayscale pixels are just a single value such as 132 which would be somewhere in the halfway point between white and black. Any more information than just the light intensity would be unnecessary and would add complexity to where it's not needed. Since the output must still be in color, the gray scale will be the form of the image that goes through the preprocessing steps and rotations. Any rotations applied to the grayscale image will then be applied to the original image. Another way to do this would be to perform all calculations on the grayscale format of the image, and once all calculations are completed, revert the image back to color. For this part of the process, the cv2 module will be used as there is a function that converts images to grayscale.

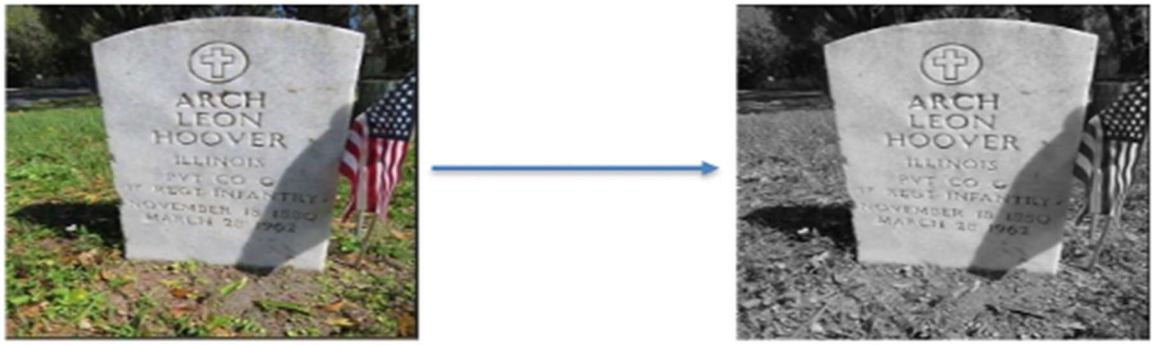

Figure 14: Conversion of an RGB Image to Grayscale

Edge/Rectangle detection

For rotation to be successful, every image needs to be rotated upright. The definition of upright will vary depending on the image in question. For example, many images are of headstones embedded in the ground. The ground may not be level enough to be considered straight and therefore cannot be used as a reference for being upright. This can apply for headstones that are already upright as well. There are some upright headstones that contain obstructions such as shadows, flags or other objects that might have been left in front of the headstone. Since these objects also represent a change in intensity, a threshold must be determined to ensure that these objects are not considered as changes in intensity. Fortunately, the Canny edge detection algorithm is just the method needed for this. The Canny edge detection method can be broken down into five steps:

- Noise Reduction
- Gradient Calculation
- Non-Maximum Suppression
- Double Threshold
- Edge Tracking by Hysteresis

The goal of edge detection in the overall rotation algorithm is to identify the edges of each gravestone so that there is a reference to rotate from. If the edges of the gravestone are not identified, then there is no reference to rotate the image. The classification model can identify a gravestone as upright even if it's upside down because there is no reference to what upright even is.

Noise Reduction

Before edge detection can begin, the image needs to be smoothed out. This is part of the process will occur in the image pre-processing stage. The general steps and motivations behind the noise reduction are mentioned in the Image Preprocessing section. The actual mechanics behind the noise reduction will involve the use of a gaussian filter.

The equation for the gaussian filter of size (2k + 1) x (2k + 1) is:

$$H_{ij} = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right)}; 1 < i,j \leq (2k+1)$$

Equation 12: Gaussian Filter Equation

The filter is calculated with a pre decided value for sigma and k. The size of the filter has a hand in determining the performance of the edge detector. For larger values of k (thus for larger and larger filters), the edge detector will lose sensitivity to noise, making it harder to smooth the image and thus harder to detect the edges. A filter that balances performance and size will be a 5 x 5 filter (k = 2) and will be the filter used in the overall implementation of the Canny edge detector. With sigma set to 1.0, the resulting filter is created:

$$\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 16 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figure 15: Gaussian Mask with Sigma set to 1.0

Once the Gaussian filter is created, it its convolved with the grayscale form of the input image resulting in a more blurred image.

Gradient Calculation

Since edges can point in any of the 8 cardinal directions (up, down, left, right, diagonals), the magnitude of the gradient and the direction of the gradient need to be calculated. The gradient is calculated by taking the x and y partial derivatives of the gaussian function and then applying this formula:

$$G = \sqrt{G_x^2 + G_y^2}$$

Equation 13: Magnitude of Gaussian Gradients

Where G is the magnitude of the gradient. To calculate the angle, the arctan function of the y component divided by the y component is calculated:

$$\phi = arctan\left(\frac{G_y}{G_x}\right)$$

Equation 14: Direction of Gaussian

The direction is rounded to an angle representing one of the four directions (0 for horizontal, 90 for vertical, 45 for the positive diagonal and 135 for the negative diagonal).

Non-maximum suppression

Although the magnitudes of the pixel values have been calculated, further steps need to be taken to identify the sharpest edges. Minimum cut-off suppression of gradient magnitudes or lower bound thresholding is a form of edge thinning and the technique used to the find the sharpest change of intensity values in the image. The algorithm works by comparing each pixel to its neighbors that point in the same direction. If the neighbors have higher intensity values compared to the pixel in question, the original value of the pixel is suppressed or set to zero. This means that in the given direction, the intensity change is not as great as other pixels. If the pixel in question has a higher intensity value than its neighbors in the same direction, the original value is maintained. Example code for this step is provided below.

```
def peaks(I_magnitude,slope,n):
    out = np.zeros(slope.shape)
    for i in range(slope.shape[0]-n):
        for j in range(slope.shape[1]-n):
            if(slope[i][j]<=0.4142 and slope[i][j]>-0.4142):
                if(I_magnitude[i][j]>I_magnitude[i][j-1] and I_magnitude[i][j]>I_magnitude[i][j+1]):
                    out[i][j]=255
            elif((slope[i][j]<=2.4142) and (slope[i][j]>0.4142)):
                if(I_magnitude[i][j]>I_magnitude[i-1][j-1] and I_magnitude[i][j]>I_magnitude[i+1][j+1]):
                    out[i][j]=255
            elif((slope[i][j]<=-0.4142) and (slope[i][j]>-2.4142)):
                if(I_magnitude[i][j]>I_magnitude[i+1][j-1] and I_magnitude[i][j]>I_magnitude[i-1][j+1]):
                    out[i][j]=255
            else:
                if(I_magnitude[i][j]>I_magnitude[i-1][j] and I_magnitude[i][j]>I_magnitude[i+1][j]):
                    out[i][j]=255
    return out
```

Figure 16:Peak Calculation Function

Double Threshold

After the non-maximum suppression step, the pixels that remain give an overall better idea of what the actual edges are. However, due to factors such as color variation and leftover noise, another step needs to be taken to ensure that weaker gradient values are removed and higher, stronger gradient values remain. To achieve this, high and low threshold values are decided empirically and determine pixel strength:

- If the pixel value is higher than the high threshold, the pixel value is considered a strong gradient

- If the pixel value sits between the high and low threshold values, it's considered a weak edge pixel

- If the pixel value is lower than the low threshold value, it is suppressed completely

Edge Tracking by Hysteresis

The final step of edge detection involves filtering out the pixels with weaker gradients. The strong gradient values, (values higher than the high threshold) are values that represent the true edges. These values more closely represent the true edges in the image. However, weaker edge pixels represent outliers caused by noise and color variation still need to be filtered. To accomplish this, each pixel and its eight neighbors are examined. If at least one of the eight surrounding pixels is a strong gradient, the pixel is still considered worth preserving.[11]

Justification for the use of the Canny Edge Detection Method

Another popular edge detection method is the Sobel method. The Sobel edge detection method works by using masks that correspond to vertical and horizontal edge points and perform a convolution with the input image and masks. The masks are:

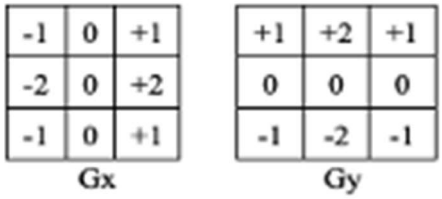

Figure 17: X and Y Partial Derivatives of Gaussian Function

The resulting gradients are then used to calculate the absolute value of the magnitude. A single threshold is used to identify which pixels are relevant to the edge calculation.

Due to a less flexible approach to edge detection, it is harder for the Sobel method to identify thin edges. The Canny method uses a multitude of adjustable parameters that can affect the speed and effectiveness of the algorithm. For example, smaller filters cause less blurring which can assist in detecting smaller lines while larger filters can detect smoother edges. The Sobel method simply does not have this flexibility. This applies to the thresholding as well. If there's only one threshold it may be set too high or too low causing either lack of detection of important features or detecting noise that does not need to be highlighted. It is difficult to find a threshold that can generally handle all cases. Here is an example of the difference in quality of the two methods after using both on MRI scans:

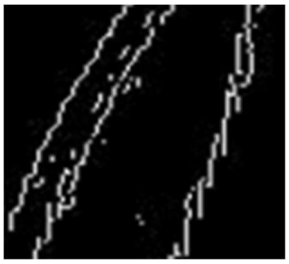

Figure 18: Sobel Results for MRI Scan

Figure 19: Canny Results for MRI Scan

Since the image needs rotated correctly for other stages of the process to succeed, the precision given by the Canny method is preferred. The only advantage that Sobel has over Canny is the fact that it uses less memory. However, this is not important as edge detection is a small part of a larger process.[12]

Rotation

Macro rotation

Once images have been preprocessed, the process of the macro rotation will begin. The macro rotation step will take the image, determine how much it needs to be rotated by based on its orientation and then perform the proper rotation. There are four different amounts that an image can be rotated by, 0, 90, 180, and 270 degrees. The rotation will always be performed in the clockwise direction for consistency direction. The rotation won't be based on how near to the positive y axis the image is as this would add negative directions to the process. For example, if the original image has the headstone pointing to the right, it will take more work to determine what rotation amount is smaller. This would also introduce negative angles to the whole process. The simplest method would be to create four different classes based on these amounts. The class would identify the type of rotation that needs to be performed. An image where the headstone is right facing would be classified as a 270-degree rotation. An image that is already upright would just be identified as a zero-degree rotation. Classification can be accomplished by using a Convolutional Neural Network.

For the training portion, the training data that will be used will include images given to use by the sponsor. As per the requirements, the training needs to achieve an accuracy of 95%. This is because many of the images passed through as input need to be rotated properly

before entering the cropping stage. An image cannot be properly cropped unless it has been properly rotated. Validation sets will also be pulled from images given from the sponsor. The four different types of rotation are shown below.

Figure 20: Examples of 0-, 90-, 180-, and 270-Degree Rotation Classes

Micro rotation

Once the image has been rotated to face upright, one final adjustment needs to be made and that is the micro rotation. While the image itself may be upright, the image might be tilted a bit to the right or a bit to the left. This is where the micro rotation would come in. This step of the process will ensure that the image is prepped properly for the cropping stage. The micro rotation will have 11 classes based on sponsor requirements. The classes are the angles ranging from negative five to positive five degrees inclusive ([-5, 5]). These angles are based on the average rotation interval needed after macro rotation.

Figure 21: Examples of [-5, 5] Degree Rotation Classes

Model Design

Technology to be Used for Model Training and Construction.

For the model construction and, the TensorFlow library paired with the Keras interface will be used. Keras contains many of the functions and classes necessary to build an image classification network. Since the model is going to be simple in design, the model will be an instance of the Sequential class. The Sequential class makes it easy to add different types of layers (Max Pooling, Dropout, Dense) which allows for flexibility in the design.

Due to the high computational cost associated with image processing, both models will be trained in a Google Collaboratory notebook with a Graphical Processing Unit (GPU) attachment. The GPU given when mounted can either be an NVIDIA K80, P100, P4, T4, V100, or A100. This option will be used because it is free and would provide the same results as actually paying for a GPU to mount to a local machine.

Once the training is complete and a desired performance for both models is achieved, the models will be stored in a Hierarchical Data Format (HDF) file, specifically, an HDF5 file. The model will be loaded when the application is run, and the model will be used only to classify.

Internal Dataset storage

For storing the images, the ImageDataGenerator class paired with the *flow_from_directory* function will be used. The ImageDataGenerator class is a class provided by Tensorflow and allows for the rescaling of images. This means that all pixel values are scaled down to values between zero and one. This is accomplished by setting the re-scale parameter to 1/255. The data will be pulled from the google drive storing all the training and validation data. The *flow_from_directory* function will take a directory to training and validation subdirectories which will then contain subdirectories with the images sorted by class. Below is the directory structure needed to properly load training and validation data for both types of data.

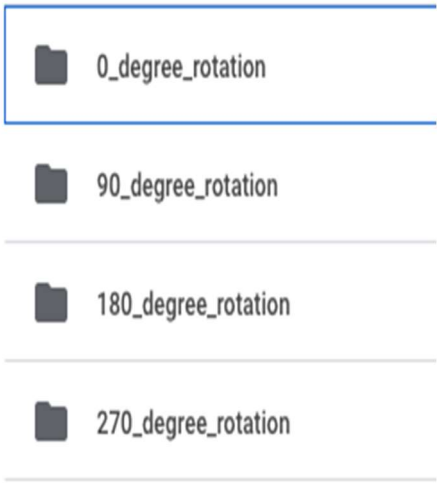

Figure 22: Directory Structure for Macro Rotation Classes

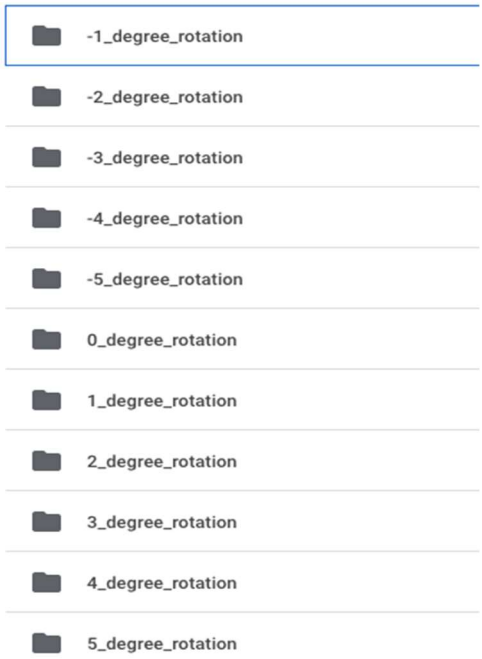

Figure 23: Directory Structure for Micro Rotation Classes

Model Training Parameters

To ensure consistency among the models, the models will utilize different parameters to ensure high efficiency and performance. For the activation functions, ReLU will be used for several reasons:

- It assists in avoiding the vanishing gradient problem by returning the max of the input and zero, therefore not tending towards zero with smaller inputs.
- Most modern CNN architectures (Except LeNet-5) have ReLU as the default activation function.
- Since the vanishing gradient problem will be taken care of, more layers can be used without risk of dead neurons.

The output layer will use a SoftMax activation function. The macro and micro rotation models will have four and 11 units for the output layer respectively. The parameters for compilation will include an Adam optimizer, a categorical cross entropy loss function (for multi class classification) and an accuracy metric. For the optimizer, initial training will utilize a learning rate of 0.001. This is the default given by the Adam object in Keras. Many

hyperparameters will be adjusted similarly based on training results and patterns. The accuracy metric will help with determining if there is overfitting during the training and validation steps. If there is an increasing training accuracy but decreasing or flatlining validation accuracy, there is a chance the model is overfitting to the training data and will fail to generalize on any new inputs. If both the training and validation accuracies are flatlining or decreasing per epoch, the model is underfitting and isn't generating any patterns at all.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 6)         60
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 16)        880
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 120)               69240
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 340
=================================================================
Total params: 80,684
Trainable params: 80,684
Non-trainable params: 0
```

Figure 24: Implementation of LeNet-5 Architecture using Keras and using Max Pooling Layers

Depending on the architecture, batch normalization and Dropout will be used to help with model performance. Dropout will assist with the overfitting issue and batch normalization will help with performance. These layers, however, will be added depending on necessity. Dropout layers, if needed, will be implemented after each Dense layer. If during the training process no overfitting occurs, there will be no need for the Dropout layer. If anything, it will be used for optimization.

Dataset Creation

The data that will be used to train the macro/micro rotation detection models will come from a set of headstone photos given from the sponsor. Any new photos that are given from the sponsor are fed into a custom rotation program. The program will perform the appropriate rotation (Depending on the type of rotation) decided by a random number. The newly rotated photos are placed into a new directory where they await to be labeled. For

labeling, the photos are labeled based on the degree of rotation that the. Three headstone photos sets are used to create the training/evaluation dataset: Standard headstones, outliers, and flats.

Standard Headstones

Standard headstone photos are photos taken in ideal conditions (good lighting, no changing of the perspective) and are of standard shaped headstones. They don't contain any noise beyond the standard decorations such as flags that are kept to the side or excess foliage. These types of headstone photos are what is expected to make up much of the input into the overall system.

Outliers

Outlier headstone photos are usually photos of more uniquely shaped headstones and are usually not taken in ideal conditions (shade covering the text, photo taken from a different angle, usually a lot of decorations or shadows blocking the headstone). As mentioned before, one of the requirements of the system is that the rotation models score with at least a 95% accuracy rate. While only showing photos of standard headstones during training can help achieve this, it is not known what will be passed to the system during actual deployment. The model must be prepared to handle any type of headstone rotated in any direction.

Figure 25: Example of an Outlier Headstone

Flats

Flat headstone photos are photos of headstones that are placed in the ground rather than stand upright. There are two main differences between above ground headstones and flat, in ground headstones in the context of the rotation models:

- Shape – While both flat and above ground headstones are both rectangular, above ground headstones typically have a curved top (with some exceptions). Flat headstones are always rectangular.

- Orientation – Even if both types are the same shape, flat headstones usually have a longer length than height while curved headstones are the opposite

As a result of these differences, different types of headstones will have different definitions of "upright" than others. This needs to be accounted for. If not, the model might identify that a flat headstone needs a 90/270-degree rotation when it is already upright or needs a 180-degree rotation

Figure 26: Example of a Flat Headstone

Issues with Micro Rotation

During the micro rotation and labeling process of the standard images, a couple of issues occurred. These issues were related to how python processes images and the way some of the images are formatted pre rotation.

The first issue occurred when testing different rotation functions on different types of images. No matter the image type (standard, outlier, flat), anytime a rotation in the range of was done, there would be extra black space that would fill up the frame of the image. Parts of the image would also be cut off due to the rotation. The imutils library has a *rotate_bound* function that ensures that no part of the image is cut off. Despite this, the black space issue still occurs. To fix this, a custom rotation algorithm is used instead to perform the micro rotations. The imutils library is still used to rotate the image (cv2 doesn't allow for micro rotations), however, the image is then cropped around the new rectangle that is formed by the *rotate_bound* function. This ensures the image is still maintained while removing the black space.

Figure 27: An original photo (first photo) is rotated -3 degrees with no bounding box (second photo). The photo is rotated -3 degrees with a bounding box (third photo). Finally, the photo is rotated -3 degrees with a bounding box and cropping (fourth photo).

The second issue occurred during the rotation process. Not every image in our image dataset is perfectly aligned. Some images may be taken in a way that has it already rotated on the micro scale. For example, there is an image in the dataset that is already rotated two degrees to the left. When it is rotated *x* degrees (*x* being anywhere from negative five to five degrees), the rotation program will label it as "x_image_name.JPG" when it is rotated two plus *x* degrees.

Figure 28: The Original Image (Right) is Tilted 1-Degree to the Right. When it was Randomly Rotated by -1 Degree, it's True Rotation would be 0 but it got Labeled as -1 (Left).

To resolve this issue, all the rotated images rotated were looked at. If the image rotation and the file name match, then it is labeled as whatever its file name implies (If the file is named "*x*_image_name.JPG", it is labeled as an *x* degree rotation). If the rotation and file name do not match, the actual rotation and image name are recorded in a text file. Once

this process was concluded, a function in the image rotation program parses through the text file and renames the appropriate files accordingly. The renamed files were then labeled properly.

```
def rename_micro_rotations(fix_file, path_to_photos, store_path, file_extensions):
    corrections = create_corrections(fix_file)

    path_to_photos += '/*.'

    for extension in file_extensions:
        photos = glob.glob(path_to_photos + extension)
        print('Renaming images with extension .' + extension + '...')

        for i, photo in enumerate(photos):
            # Isolating name of photo
            name = photo.split('/')[2]

            # Getting the number of the photo to correct
            photo_to_correct = name.split('_')[2][:4]

            # Checking for non existent photos or duplicates
            if photo_to_correct not in corrections.keys() or corrections[photo_to_correct][2]:
                continue

            # This photo has is valid and going through the renaming process
            corrections[photo_to_correct][2] = True

            # Changing the name of the photo with actual rotation
            new_path = os.path.join(store_path, str(corrections[photo_to_correct][1] + '_')) + name[2:]

            # Read in photo
            image = cv2.imread(photo)

            # Removing the wrongly named file from the original directory
            os.remove(photo)

            # Writing image to new directory
            cv2.imwrite(new_path, image)

            print(photo + ' renamed to ' + new_path + ' (' + str(i + 1) + '/' + str(len(photos)) + ')')

    return
```

Figure 29: Code for Renaming Photos that were Mislabeled

The command line program takes in as input a path to photos, a storage path to the modified photos and the appropriate extensions (the data only had .JPG and .jpg inputs). Once the initial input is given and the option to correct the mislabeled photos is selected, the user is prompted to enter the path to a text file containing the corrections needed. The text file will have the rotation given to the image, the actual rotation it represents and the photo number (0 followed by 3 digits). A dictionary is created using the text file with the number being the key and the actual rotation being the value. The *glob.glob* function will create an iterator for paths to the photos ending in the corresponding extension. Each photo is parsed through to extract the name and membership of the photo name is checked against a dictionary containing the correct file name as the value. If the photo number is not a key in the dictionary, then it does not need to be corrected. If the photo number is a key of the dictionary, a new path renamed with the actual rotation is created. The old photo is removed from the directory and the newly named photo is placed in the storage location given by the user. A Boolean flag is also kept as a value in the dictionary to avoid duplicates.

Canny Edge Detection

The canny edge detection implementation is just a straightforward code application of the canny detection algorithm. A combination of the NumPy, OpenCV, and PIL libraries were used for implementation. The process would work like this:

1. The image would be read in using an OpenCV image reading function
2. The image would be converted to greyscale using an Open CV color conversion function
3. The masks would be created.
4. The image would be passed into different custom Canny functions to detect the edges.
5. Threshold values of 10 and 25 would be used to create the final output.

Despite the process being straightforward, there were a couple of issues that arose. The resolutions of these issues would lead to the solutions that would drive the final implementation of the canny algorithm.

Custom Canny Edge detection vs. OpenCV Canny function

The OpenCV library offers its own Canny edge detection function. While convenient, it doesn't allow much customization of parameters. It only allows the adjustment of the lower and upper thresholds. Both implementations work the same and yield the exact same results (Assuming the thresholds are equivalent). With our custom Canny edge detector however, we have more values to adjust thus more freedom. The custom algorithm allows us to adjust the size of the filters, the sigma value, while still allowing for the choice of threshold sizes. This amount of freedom does create more time needed for experimentation to figure out the ideal values, but those ideal values will result in higher precision for the edge detection.

Speed of Algorithm in Relation to Image Size

```
Derivatives calculated
Magnitude calculated
Peaks calculated
Final calculated
Canny edge detection with image of size (5073, 3801) executed in 746.7118396759033 milliseconds
```

Figure 30: Results of Canny Edge Detection with no Resizing

Since the Canny edge detection is only a part of a much larger process, it needs to be optimized as to not slow the user experience down. An issue that occurred during testing of the Canny edge detection algorithm was the speed at which the process ran was way too long for it to be implemented into the system. It took 746 seconds (above) to run.

To fix this issue, the images will be reshaped to a shape of (400, 400). Image size only affects the speed of the edge detection algorithm and nothing more. Furthermore, the image will be further downsized before prediction as the model is trained on smaller sized images. The result of canny edge detection with a 400 x 400 image are shown below.

```
Derivatives calculated
Magnitude calculated
Peaks calculated
Final calculated
Canny edge detection with image of size (400, 400) executed in 6.634233713150024 seconds
```

Figure 31: Results of Canny Edge Detection with Image Resizing to (400, 400)

Results

Each step of the Canny edge detection algorithm has its own significance in that they provide the building blocks for the next. Below, is a grid of the result of each step of the canny edge detection process. For the parameters, gaussian filter size of five was used for both the x and y gaussian filters. A kernel size of three was used to calculate the x and y derivatives. A sigma value of 0.5 was used in all calculations. A high threshold value of 35 and a low threshold value of 10 were used for non-maximum suppression. The “x smoothed” image details the horizontal features being denoised after performing a convolution operation with the x gaussian filter. The same thing is done to achieve “y smoothed” with the y gaussian filter. Obtain the slope of the images, the magnitude images are generated and used for slop calculation. In the “x magnitude” image, changes in the x-axis are emphasized and vice versa for the “y magnitude” image. The “peaks” image displays the global maxima in different directions. The “final” image is generated by suppressing irrelevant pixels while emphasizing relevant ones

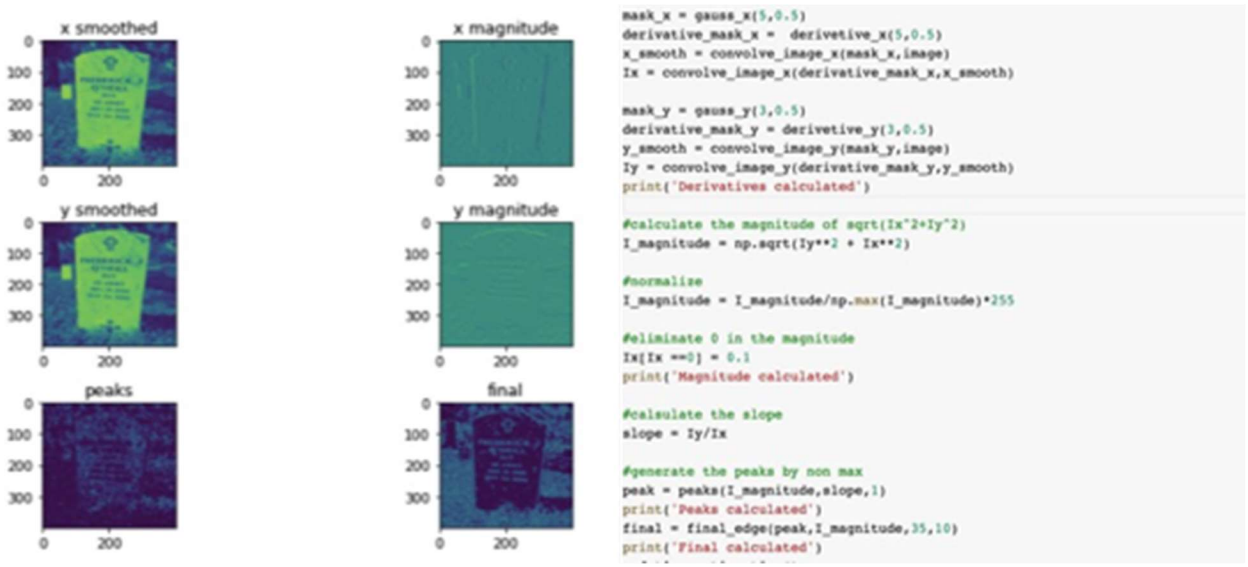

Figure 32: An Image at Every Step of the Canny Edge Detection Process (left) with Corresponding Canny Driver Code (right).

After all the steps are completed, the result should be a black and white version of the original image where all the changes in intensity have been emphasized. There should be a smooth curvature surrounding the headstone along with everything surrounding the headstone. Depending on how weathered the text is, some text may appear. Whether text appears is not a factor in the success of the algorithm. The most important factor is the edge surrounding the headstone. If periphery objects aren't captured, it doesn't matter. From this point, any features such as the curvature, contour, the corners, and anything to do with the geometry of the headstone can be captured to assist with the edge detection process.

On the top left, the Canny algorithm didn't capture any of the text from the original (bottom left) because the text is the same color as the headstone, thus an intensity change cannot be detected. A lower high threshold could reveal more text allowing for lower intensity changes.

The image on the top right corner however contains the intensity changes for the text because the original (bottom right) has black text on it, making the intensity change more apparent. Smaller details such as the color changes in the stone even appear in the edge detected image for the same reason.

Figure 33: Images (Bottom) with their Edge Detected Versions (Top)

Classification and Rotation

Predict Function and Probabilities

Once the image passes through all previous steps, the final step is to predict and rotate accordingly. The *predict* function returns an array of probabilities (all between 0 and 1, with a sum of 1) that represent the probabilities for classes [0, n – 1] for each image passed in. Each probability represents the likelihood that the specific image is a given class. For this specific case, only one image will be passed through the *predict* function at a time.

Since the *predict* function is an extension of the model, it follows all the input guidelines the model has set. This means that a batch dimension and a color channel is needed. To do this, the image's dimensions will be expanded from (width, height) to (1, width, height, 1) where the first number represents the batch size (how many images are being passed through the model) and the last number represents the color channel (in this case 1 for grayscale images). The function needed to do this is the NumPy *expand_dims* function which takes the NumPy array to be modified, and the dimensions to expand to. The image will also have to be resized to match the input width and height that the model requires. This is all dependent on the architecture used (See Design).

The rotation angle will be based on the index of the highest probability. For example, if a macro rotation prediction was being made, and the probability array for the image was [0.2, 0.5, 0.1, 0.2], the highest probability is that of the *90_degree_rotation* class.

Error Checking

After the rotation is classified and rotated, there will be a check to confirm that the image is upright before finishing the process. The image will enter a loop that will only run if the image classifies as anything else other than a zero-degree rotation after initial rotation. In the loop, the image will be re-classified and re rotated. A count of each reclassification will be kept. The loop will run five times. If the image has not reclassified as a zero-degree rotation in those five iterations, the image will be rotated in the negative direction of the most common reclassification. If all reclassifications occur equally, the smallest rotation will be performed instead This is to ensure that the micro rotation classifies correctly and in the case that. The model will have a 95% accuracy, but this step will cover that small, but still possible chance of error.

Cropping

Objectives

The proposal fully details the process of headstone classification by extracting features from headstone dataset with utilizing Convolutional Neural Network architecture and points out the implementation of predicting the position information of headstones, in addition, it will design an automated system to de-noise the headstone dataset after cropping process. As will be mentioned in the following sections, a series of sorted headstone dataset will be provided.

Challenges and Goals

Following on from the previous part which has completed the rotation process for the headstone images, the goal of this part is to identify the headstone from images, accurately locate the position of the headstones in the images and additionally de-noise the obstructions on the headstone. Three difficulties will need to be solved to achieve the expectation:

- The shape of headstones is not unique, multiple styles of headstones will increase the difficulty of classification.
- From the training dataset, multiple headstones show up in the background which will become a false target; however, there is only one headstone of each type will need to be detected in a single dataset, so the detect system must be able to identify the correct target headstone from images.
- The quality of images is different, some of the headstones are extremely blurred, so the system needs to be able to relatively against noise.

In this section, we will apply machine learning techniques and CNN algorithms to introduce an automated system to predict the position information and the classification information of the headstones and an autoencoder system will be subsequently designed to increase the quality of the images to solve the noise issues.

- The first goal is to design and operate an automated system to identify the headstone and locate the coordinates of the headstone from the images to generate the cropped headstone.
- The second goal is to eliminate the obstruction around the boundary of the headstone.

- The third goal is to de-noise the text region on the headstone to make the text more instinct.

Background and Related Work

The following sections illustrate the methods and techniques we will apply to the application and elaborates on the outcome we expect to achieve. In addition, the method indicates majority of essential elements we need to utilize to accomplish our results.

Machine learning, Neural Network, and Computer vision

Machine learning is a concept that settles practical applications with mathematical algorithms by abstracting practical problems into mathematical models. As a branch of artificial intelligence, machine learning is focus on building up different smarts system and models to accomplish the real artificial 'self-study' algorithms. Neural Network is one of the machine learning algorithms which was originally created by the idea of animal's brain. A group of connected neutrals that as same as the neurons in animals' brains is called artificial neurons. Each neuron stores information and the information can be transited between different neurons by the connection which is called synapse in animals. Each artificial neuron that receives the signal can process the information and decide to continue to send the processed information to the next neuron.

In machine learning applications, a neural network is a sort of computation model which composed of lots of nodes. Each of the nodes represents a specific output function that is called activation function. Each node will have a weight that represents the importance of the node. The output of the network can be different depending on the connection method of the network, the weight value, and the activation function. The network itself is usually an approximation of a certain algorithm or function in nature. Usually, a neural network will be composed of three components: input layer, hidden layer, and output layer.

Input layer is the original dataset after preprocessing and will be sent to the machine learning model; the input dataset will be computed by a specific equation with the nodes to generate an output of the current layer; the hidden layer is also called black box which contains a set of layers that is predesigned to continue the operation in the previous layers; the output layer is also called output layer which will generate the final results and predictions.

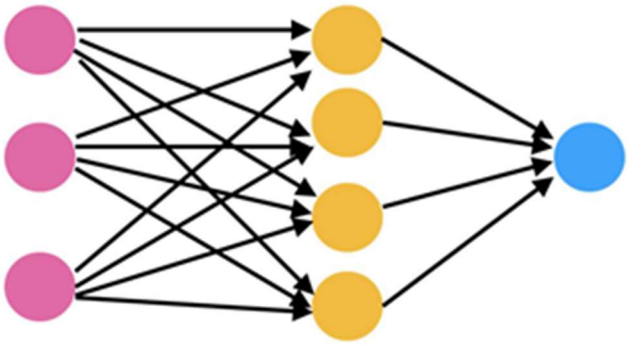

Figure 34 The basic structure of fully connected neural network.

The classic classification architecture shows the simple idea of Neural Networks. The output of each layer will always be the input of the following layer. Once the training begins, the image will be converted into a column vectors and a linearized vector algorithm is completed at each neuron in the hidden layer. After the 'activation' (RELU) function implemented, as same as previous layers, a repetitive step will be implemented and generate a final output after cost function that usually represents the possibility of prediction in classification applications. More complicated models such as Convolutional Neural Network will be introduced in the following.

Computer vision is a study that teaches computers to complete the application as human eyes do through mathematical algorithms. As a branch of machine learning, tons of machine learning algorithms, such as SVM and Naïve Bayes, has been extensively used in computer vision applications. As a general computer vision application, for instance, visual tracking is also regarded as a dichotomy problem, which can identify the difference between the target and the background of each frame. In the tracking process, by collecting positive and negative samples and training the classifier, choosing the largest response result from the classifier is typically a classical machine-learning algorithm. In this project, detecting the headstones and identifying the text information on the headstones will also be defined as a computer vision application. We will utilize the convolutional neural network structure to build up a feature extractor to extract the feature information from preprocessed images in different scales and apply the object detection techniques to accurately predict the position information of the headstones in the image.

Convolutional Neural Network

Convolutional Neural Network is a feature extractor which helps us to extract the headstone features from images and combined with fully connected layers to classify the two types of headstones and background. In each layer, a n * n matrix which is called filter (mask) will be applied to convolved with the current feature map and to generate a new feature map. The n*n filter will travel through each position of the whole image pixels. For one movement, each value of the filter will find the associated pixel from the image and multiplies with the pixel value, the summation of all the products will be the new pixel value. CNN also introduced the concept of receptive fields to extract features from larger areas. The activation value of each layer in the Convolutional Neural Network could be regarded as a scaled feature map of the original images. In other words, the deeper the CNN architecture, the higher-level features will be extracted. In this project, we implemented a headstone classifier by implementing CNN architecture to identify two different types of headstones.[13]

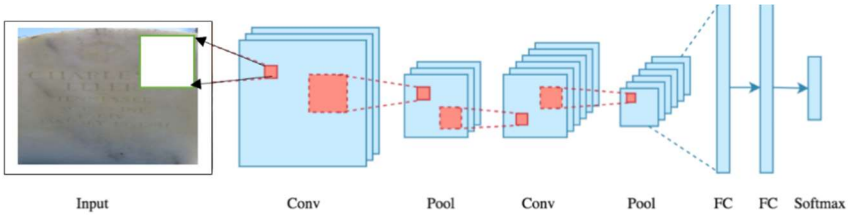

Figure 35 The Construction of Convolutional Neural Network

Convolution

Interpretation of the convolve process in the perspective of intuition of transformation, the convolution process can be regarded as a mathematical process of linear transformation and mapping into new values at each associated position of the image. The convolution filter is kind of a weight. If the position of each value on the filter can be represented as w, represent the pixel at the corresponding position of the image as x, as a result, The result value of the corresponding position will be w.T * x . In other word, it will also be as same as the vector inner product. From this perspective, multi-layer convolution is performing layer-by-layer mapping and forming a complex function. The training process is learning the weights required for each partial mapping. The training process can be regarded as a function fitting process.

Considering it from the perspective of template matching, it will be thought that convolution and correlation can be equivalent in calculation. In lots of real-world applications, correlation operations are commonly used for template matching. Since the convolution kernel usually defines a specific mode, at the same time, the convolution operation will be calculating the similarity of each position corresponding to the mode. The more similar the current position and the pattern are, the stronger the response will be. The hidden layer in the convolutional neural network can be regarded as template matching as well. Each element in the graph represents the degree of similarity between the current position and the mode. The features learned by the shallow layer are simple edges, corners, textures, geometric shapes, and some simple features, while the features learned by the deep layer are more complex and abstract, such as faces, car components, and animals.

The weight of the convolution kernel of each layer for a convolutional neural network is learned by data-driven learning. It is not manually designed. As human beings, only simple convolution kernels will be able to be designed and used in real applications such as edges detection. It will be extremely difficult for human beings to clearly interpret the convolution kernel with a complex mode. Convolutional neural network will be able to extract the image and learn features from simple to complex with deeper layers. The complex patterns are composed of multiple simple patterns. In real practice, the first few layers will do the down sampling to the original image and extract the simple features, but as the layer goes deeper, more comprehensive features will be showed. One of the reasons why we need to use convolutional neural network is that the CNN will be able to extract all different sizes of the objects on the images because of the down sampling process, so when the convolutional neural network completed the convolve process, it usually needs to follow by a pooling layer and an activation function. If there is no non-linear activation function, the network can still learn the mode, but the expression ability will be relatively decreased. Through the cooperation of pooling and activation function, the CNN model can be seen that the characteristics learned in each layer of the reproduced are very simple.

One essential contribution of CNN architecture is that we can easily change its capacity by controlling depth and breadth. In addition, compared with completely fully connected Neural Networks, the fully connected Neural Networks will need to transform each dataset to a column or row vector in each layer, so it has a high opportunity that it’s losing the relationship of adjacent pixels on the image, when the model goes deeper and deeper, this situation will be more serious, but the Convolutional Neural Network will always extract the feature information by combining the area of the pixels, so as the layers goes deeper, what we will have is the whole features information in different scales; in addition, CNN contributes much fewer connections and parameters, so the implementation of training can be simpler and more efficient.[14]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 36 The Demonstration of different Resnet Architecture

Resnet 50 and VCG

We compared different types of CNN architectures based on the research, the most important concern of the application is the accuracy, so it turns out that Resnet which is an optimization of Convolutional Neural Networks will be selected and implemented as the feature extractor in this project. The main idea of ResNet is to add a direct connection channel to the network, allowing the original input information to be directly transmitted to the subsequent layers. Theoretically, the deeper the CNN architecture turns into, the training error is expected to continue to decrease; however, in practice, the training error will temporarily decrease and tend to increase. The vanishment of the gradient is the main consideration in explaining this. In each hidden layer, RELU will be implemented as an activation function, and when the model starts training forward, all the values less than zero be set to zero, and several feature values disappear, so the gradient correlation is decreasing with increasing layers.

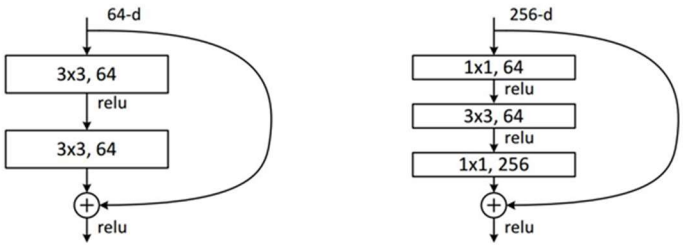

Figure 37 Architecture of Resnet Block

Based on the structure of ResNet50, it mainly introduced the 1x1 convolution which reduce the parameters for each layer. Specifically, what the 1x1 convolutions do is not only updating and reducing the total number of channels and to additionally achieve the linear combination of multiple feature maps with maintaining the same feature map size as the original, but also when it's compared with other sizes of convolution kernels, it can greatly reduce the computational complexity. For instance, if 3x3 convolutions layers are used, there is only one Relu, but when 1x1 convolution is used, there will be two Relu, which introduces more nonlinear mapping. Additionally, calculating the computational advantage of 1*1 convolution.

Resnet Block

A 'shortcut' called residual block is developed and composed of a duplicating unit in CNN architecture to deal with the gradients vanishing problem in Deep Neural Network. Resnet can efficiently reduce the effect of gradient disappearance. In order to have a better performance in headstone classification, we will implement the Resnet technique and implement the headstone classifier based on Resnet50 architecture.

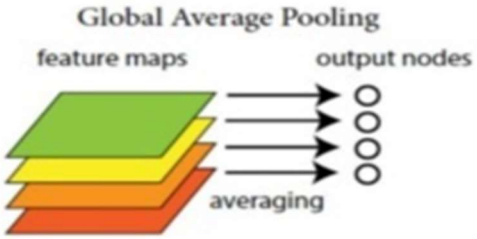

Figure 38 Architecture of Global Average Pooling

Fully Connected Layer and Global Average Pooling

The fully connected network has been known as the standard structure of the CNN for most classification applications in the real-world. In the most cases, there will be an activation function such as Relu and sigmoid for classification after a full connection layer. The most important reason why full connection network works is because that it's able to transform the incoming feature maps obtained by the previous layer of convolution into a row or column vector and multiply with a vector to reduce its dimensionality. The input will be sent into the SoftMax layer to get the corresponding score of each category. One essential problem that fully connected network could have is that the fully connected layer usually contains tons of parameters in each fully connected layer, so over-fitting will be an essential issue that need to be handled. In most applications, there are some techniques that specifically solves the problem of over-fitting such as dropout. The difference between global average pooling and average pooling lies in the word "global". Both global and local are literally used to describe the pooling window area. Local is to take a subregion of the feature map to average, and then slide this subregion; global obviously averages the entire feature map.

Assuming a network end up with 1000 categories, as a result, the feature map output by the last layer of convolution will only have 1000 channels, and then apply global pooling to this feature map, and output a vector of length 1000. This It is equivalent to removing the characteristics of the black box operation of the fully connected layer, and directly giving each channel the actual category meaning.

In the most popular CNN structures, the final output is usually generated by applying a fully connected layer for classification; however, the fully connected layer in practice is much easier to overfit than global average pooling which average the value of the whole feature maps. Dropout techniques increase the generalization ability of the entire model for the middle layers, but the fully connected layer more or less cut off the relationship between

the output and the middle layers. The global average exactly uses a simple average to establish the relationship between the feature maps and the category. The fully connected layer relies too much on dropout to avoid over-fitting whereas the global average pool can be regarded as a structured regularity.

One-hot technique

One-hot is a data transformation technique that is usually used in the multiple classification applications, it will generate a row vector that represent the possibility of each object. The mechanism of one-hot is to transform a single digital number into a vector to be used as the number of the class. For instance, in a 3 objects classification problem, when we define 0,1 and 2 represent 3 different classes, the one-hot technique will transform each object into a vector of [1, 0, 0], [0, 1, 0], [0, 0, 1]. The size of the vector is the number of classes.

MSE and Cross-entropy and SoftMax

As we talked about, there will be one loss variable that represents the confidence score of the classification. In our case, we will have 2 objects to predict. After comparing with MSE (mean square error) loss function, I will implement cross-entropy combined with SoftMax as the loss function to generate the classification prediction. The SoftMax function is basically an idea that transforms all the output into a possibility. In our case, the system will need to predict two objects: headstone and short headstone, after one-hot transformation, each output of classification prediction will be a row vector that contains two values which represent the possibility of each object. The vector will need to be combined with cross-entropy loss function to calculate the total loss.

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -[y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)]$$

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Equation 15 The equation of SoftMax and cross entropy

The reason that least squares are the best approximation point for the observation vector in the vector space:

1. It is simple and easy to implement in applications, so we don't need to spend too much time on solving the completed math equations.

2. It provides a measure of similarity with good properties such as non-negative, unique certainty, and symmetric.

3. The physical properties are clear, and the properties remain unchanged after transformation in different representation domains.

4. Easy to calculate when it's usually the derived problem is convex, with symmetry and derivability. Usually there is an analytical solution, and it is easy to solve iteratively.

Prepare the Loss Function

Loss function is a mathematical equation that calculate the difference between the target and predictions. In our project, we will totally have four predictions that we will need to calculate the losses which are object, position, and classification. The final loss will be the summation of all the four losses. The first loss will be the possibility of containing an object; the second loss will be the possibility of not containing an object; the third loss will be composed of 4 variables (height, width, center_x, center_y) that calculate the difference between the predicting bounding boxes and the ground truth; the last loss will be the classification loss. After computing the summation of the losses in each iteration, we will apply the back propagation technique to calculate the partial derivatives for each variable of each layer. All the parameters will be updated by the computed derivatives and generate a new set of parameters for the next iteration. The purpose is to update the parameters which could minimize the loss function.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$
$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$
$$+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Equation 16 Loss functions for Training Process

For the confidence loss of a box without an object, assign a small loss weight, which is recorded as 0.5 in YOLO. The confidence loss of the box with the object and the loss weight of the loss of the category are normally set to 1. In the prediction of boxes of different sizes, compared to the prediction of the large box, the prediction of the small box is more intolerable. The sum-square error loss is the same for the same offset loss. In order to alleviate this problem, we will take the square root of the width and height of the box instead of the original height and width. The horizontal axis value of the small box is small, and when the offset occurs, it will reflect that the y-axis is larger than the large box.

Focal Loss

For object detection process, we will encounter a potential issue that the possibility we could detect an object is much less than the possibility of detecting a background, if we keep the weight of all the loss function same, it will cause the model parameters updated quickly and the loss will decrease quickly as well, but the model is not robust and efficient because it does not have enough positive training samples in training. As a result, I will give a factor that is between 0 and 1 for the second loss which represent the possibility of not containing an object to lower the influence of the background.

In the field of object detection, an image may generate thousands of candidate locations, but only a few of them contain objects, which brings about imbalance in categories. If we keep the regular algorithm without taking care of imbalance issue, the training process will be inefficient as most locations are easy negatives that contribute no useful learning signal; additionally, the easy negatives can overwhelm training and lead to degenerate models.

The number of negative samples is too large, accounting for most of the total loss, and most of them are easily classified, so the optimization direction of the model is not what we expected. In fact, there are some algorithms to deal with the problem of unbalanced categories, such as OHEM (online hard example mining). Although the OHEM algorithm increases the weight of misclassified samples, the OHEM algorithm ignores samples that are easy to classify. Therefore, in response to the problem of category imbalance. It can reduce the weight of easy-to-classify samples, making the model focus more on difficult-to-classify samples during training. [15]

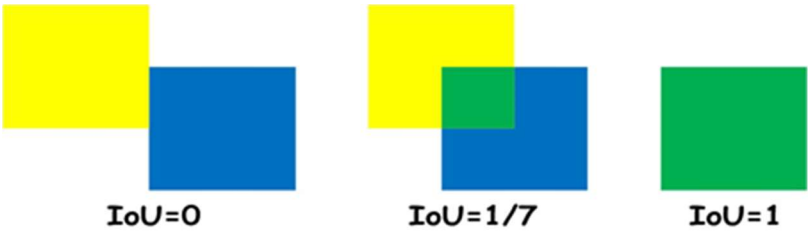

Figure 39 Intersection over Union in three situations

Intersection over Union

Intersection over union (IoU) is regarded as a useful equation for measuring overlap between two bounding boxes. The calculation is the ratio of overlapped area between ground truth and anchors over the total area. For the detection model design, two anchors will be generated, but in each iteration, only the better anchor which has the higher IoU score will be used to participate in the training process.

Figure 40 The headstone in the front is the target headstone; multiple headstones in the background is the noise for detection.

From observation of the training dataset, a small part of headstone images contains multiple headstone which could cause a misclassification problem since there will be only one or two headstones that need to be identified. In this case, all the multiple headstones are really smaller than the one we need to detect, so the solution for this problem is to set up a threshold that indicates the minimum size of the headstone that need to be detected. All the result that is smaller than the threshold will not be regarded as an object.

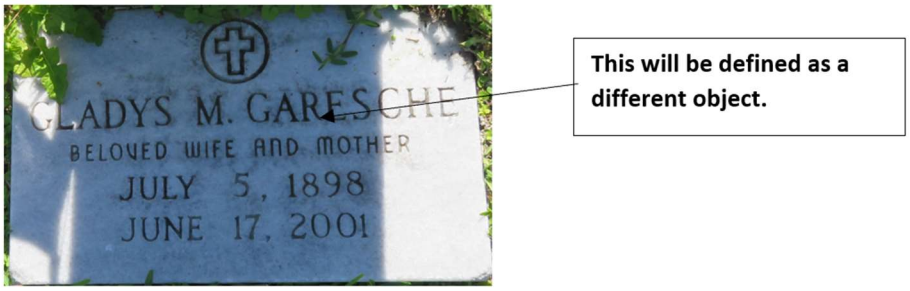

Figure 41 training dataset of short headstone

Another potential problem is that we will have multiple types of headstones. In our project, most of the headstone has a shape of tall and narrow, but a typical shape of short and wide headstone also exists as in-ground headstone. According to the inspection, the two types of headstones have a big difference for identifying, so I will define the in-ground headstone as another object other than headstone we have defined, as a result, there will be totally two objects we will need to detect in the classification process.

Autoencoder Denoise System

Autoencoder includes two processes: encode and decode. The cropped headstone images will be added noise and convolved with filters and down-pooling to deeper and narrower in the encode process to generate a low dimension vector. The generated vector will oppositely un-pooling to generate a matrix which has the exact same shape as the original image. It will compute the loss function by comparing the two data and minimize the difference between them. The differences are used to train the parameters of the encoder and decoder in this network. The purpose of training autoencoder is to make the input as closer as the output, so it will need to calculate the loss between prediction and the original image.

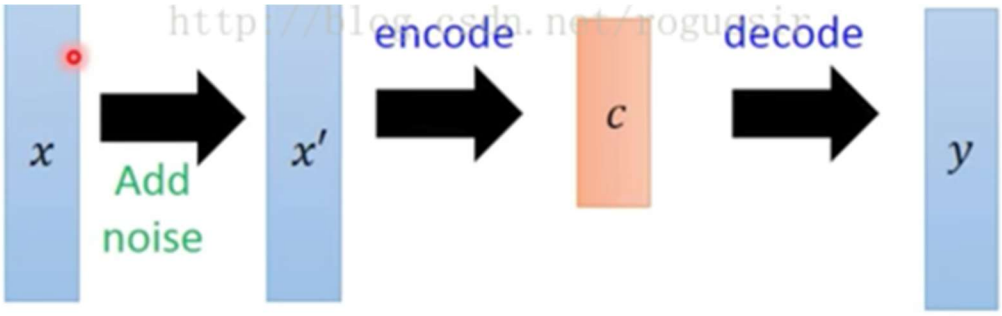

Figure 42 Structure of Autoencoder technique

As the figure showed above, we will utilize Gaussian distribution to randomly add noise into the original headstone image each iteration to make sure that the model will be able to fit all the potential situation. The noise inputs will be sent into autoencoder and generate an output which needs to be compared with original image without noise to calculate the difference to update the parameters in each layer.

Methodology

A CNN-based feature extractor will be designed to identify and detect the position information of headstones. In this section, the first component will be the Resnet50 architecture network to extract the feature of headstones, and the second component will be a global average pooling layers to transform the feature maps into a N * 5 * 5 * 12 matrix. Each of the cell will be able to identify if there exists a target and generate a bounding box. The second section will apply autoencoder techniques to build a de-noise CNN architecture to eliminate the potential noise on the cropped dataset.

Headstone Detection

The whole dataset contains 1000 headstone images that are distributed into two groups: regular headstone and short in-ground headstone. Additionally, all the data samples will be divided into 3 groups as training dataset, validation dataset, and testing dataset based on 70%, 20%, and 10%.

Each of the training samples will be transformed into a 3-channel image with a size of 448 * 448 * 3(could be changed according to the training result) as inputs to send into the CNN-based headstone detector. 448 * 448 is the width and height of the dataset, and 3 is the channel number. The training dataset will be used to train the model, optimize the parameter and minimize the total loss. The validation dataset will be used for optimizing and adjusting the hyperparameters to improve the model performance. The testing dataset will finally evaluate the performance of the detection process. We will also change some other factors to improve the performance such as number of datasets depending on the evaluation.

Data Preprocessing

Labeling Dataset

As mentioned in the previous section, we will collect 1000 headstone dataset. In order to be able to train the CNN model, a software which is designed for object labeling will be used to label the location and classification information of the headstones.

Operation of LabelImg

- Install and setup environment for the LabelImg.
- Load training raw dataset to LabelImg UI.
- Creating bounding box and store the info's into local file.

```
0 0.519805 0.501350 0.484568 0.662809
```

Figure 43 The labeling process and the label files

Store the Coordinates.

For each dataset, there will be 5 variables to store into the file. The first variable represents the class of the object, in this specific case, '0' represent headstone; the rest of 4 variables will store the position information, in the above sample, '0.519805 0.501350' represent the position of the object center relatively to the center of the image. '0.484568 0.662809' represent the width and the height of the object relatively to the width and height of the image. In our case, we will totally have 2 types of objects except background: headstone and short headstone.

Implementation of Headstone Detection

The following section will illustrate the ideas and the techniques of implementing the CNN headstone detector.

Preprocessing Raw Images

As mentioned in the previous section, the first step is to label the training samples and stored in the .txt file with 5 parameters. For the raw images, we will apply convolutional mask to each raw headstone dataset to eliminate the potential noise on each image. Based on experience, the mask size will be a size of 3 by 3 max pooling after labeling. The reason why we will apply Gaussian filter for each dataset is to improve the performance of CNN model and decrease the influence of the noise. Since each individual dataset can be instinctively different, we might apply multiple filters to dataset in the preprocessing step based on the results. The convolutional process will change the size of training dataset, so in order to keep the original size, padding will be applied before the convolutional process to keep the same size. Additionally, even though convolutional calculate will also change

the depth of the dataset which is originally 3 channels to one in the preprocessing to simplify the calculation, the purpose of cropping process is to predict the bounding box to locate the position of headstone, the final output for cropping will be generated from the original dataset which maintain the same depth.

In our training dataset, we will have multiple types of headstones that are demonstrated in the dataset. In order to standardize the training dataset types, we generally divide all the different types of headstones into two predictive object group: headstone which has the narrow rectangle shape and short headstone which has a shape of short rectangle. The headstone will be labeled as 0 whereas short headstone will be labeled as 1 to represent another group of datasets. The output label will be transformed into a N * 5 * 5 * 12; N represent the number of training samples, 5 * 5 represent the number of the cells how we split the image into potential prediction positions; 12 represent the position information and the possibility of containing object for 2 bounding boxes that generated by each cell.

Split Each Single Image to cells

In order to ensure the output predictions after CNN model and the label keep the same shape to be able to compute the total loss, we will transform each label into the same shape as the prediction which is 5 * 5 * 12. In the label files, we will load through each image as input and look for the labeling file that has the same file names. In the labeling files, the five values represent the class type, x position, y position, width, and length. As introduced in the previous sections, all the position values are stored relatively to the original dataset, so all the stored position information will need to be transformed from relative value to the real position information of the training dataset. For the center positions, the two relative values will be multiplied with the real length value and width of the training datasets; additionally, based on the shape of output cells, the cell that the headstone is located at will be labeled as one which represent has object. As same as the center values, the length and width will also be multiplied with the original width and length to generate the bounding box.

For training phase, in order to predict the position of the headstone in dataset, each headstone image will be averagely split into smaller cells. As we mentioned, all the shape of output and prediction will need to be matched up to each other. As a result, each training dataset will be divided into a 5 by 5 cells. Each cell will generate two anchors which will be able to predict the headstone that associated with the center coordinates of the cell. In other words, the width and height of the prediction output will be respectively 5 and 5. Since each cell will generate two bounding boxes, in order to make the prediction more accurate and more comprehensive to fit all the possible shape of headstones, one of the bounding will be defined as tall and narrow with respectively 3:1, another bounding box

will be oppositely defined as short and wide with also respectively 1:3. Based on the reality, only one types of headstones will need to be identified in each training dataset, as a result, in each iteration of training, the two anchor boxes will be in a compete relationship based on the NMS (non-maximum-suppression) techniques to select the better bounding box which participate the optimization process. Each of the cell will finally only have one best anchor to predict the result and generate the bounding box.

Create Anchors for Each Cell

As showed in the image above, each cell will generate two anchors that are responsible for making predictions. In order to make our model more efficient and closer to the real situation, we will define that one of the anchor boxes will be short and wide, another one will be the narrow and tall. Each of the anchor will include three predictions: object, classification, and position. The anchor where the center of the ground truth headstone is located will be responsible for predicting the bounding box and detecting the headstones. All the other anchors will not participate into the training process.[16]

Training Process

As we introduced in the previous section, we will start using real dataset to train the model, we will firstly apply the pre-train step which utilize the COCO dataset to pre-train my CNN model to improve the performance. Each input training dataset will be transformed into 448 * 448 * 3 matrix to send into our Resnet50 feature extractor. I will also implement a pre-train process with other dataset such as COCO before I start using headstone dataset because the CNN model can be pre-trained with similar resolution images before getting into the real training. After the feature extracted, each feature map will be transformed into 5 * 5 * 12 prediction matrix.

Each of the cell in the matrix can be regarded as a 12-dimensional vector which is composed of 2 anchors and a classification score (5*2 +2). Since we have the labeled output which has the same shape as the prediction matrix as we mentioned, we will implement the loss function for each component and calculate the summation of the total loss. We will implement the back-propagation techniques to update the weights in each layer.

Explanation of each loss

1. Loss of the prediction for center coordinates

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \ell_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{(y_i)}^2]$$

Equation 17 Loss of the prediction for center coordinates

The first loss function computed the total loss of the prediction for the center coordinates of x axis and y axis. x and y are the prediction value, $\hat{x}$ and $\hat{y}$ are the real center coordinates in the images. λ is a constant that we can adjust it based on the training accuracy. This function calculates the summation of difference between each grid cell and the selected bounding boxes. In our case, we will totally have 5 by 5 cells, so the total loss will be the summations of 25 losses.

Another parameter in the function L represent the possibility that a specific cell contains an object. The value will be either 1 or 0 depend on the existing object. If a grid cell contains the target headstone, we will define it as 1, on the other side, if there is no target object in the grid cell, the value of l will be assigned as 0, in the process of loss computation, the loss value for the cells that do not contain any target will be also 0. Since each cell will generate two potential boxes with different shapes, in the training phase, we expect that each cell only needs to predict one bounding box instead of two, so a technique called IOU will be applied in each iteration of training to select the better bounding box. The basic idea of IOU is to compare the overlapped area between the potential bounding boxes and the ground truth, so the IOU value will be a number between 0 and 1. The bounding box that has a larger IOU value will be selected to compute the loss.

2. Loss function of length and width for prediction bounding box.

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \ell_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + \sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

Equation 18 Loss function of length and width

The second loss function is total loss of the length and width between ground truth bounding box and the prediction bounding box. As we can see from the equation, it has lots of similarity to the loss function of the center loss function. w and h are the prediction values, and $\hat{h}$ and $\hat{w}$ are the prediction values. It also has the constant λ that can be adjusted in the training phase depends on the prediction performance. Same as the previous loss function, L is the variable that represent the possibility of containing an object in the specific grid cell. Additionally, the way of calculating the total loss is also to travel through each grid cell and add all the losses to generate the total loss. The only difference is that for each grid cell, both of the predicted value and ground truth value will need to be

computed the square root before calculating the loss. The reason why we will have to apply the additional calculation is because in the real applications, the deviation that bigger bounding boxes generated is smaller than the small bounding boxes, so we choose to measure the square root of the length and width of bounding boxes instead of directly measure the original length and width.

3. The loss of the classification

$$\sum_{i=0}^{s^2} \ell_i^{obj} \sum_{j=0}^{B} [(p_i(c) - \hat{p}_i(c))^2]$$

Equation 19 The loss of the classification

The third loss function is the loss of the classification. In our case, we will implement two different loss equation to make a comparison and choose the one that has higher accuracy. In the above, p(c) is the prediction of the classification possibility. Since we totally have two types of objects to predict, the value will be either 0 or 1. The total loss will travel through each grid cell as same as the other loss functions. If the current grid cell contains the target, we will calculate the difference between the predicted probability and the real class value, and multiple the possibility of containing an object in the specific grid cell. If the current grid cell has no object, the loss for this grid cell will also be 0. Additionally, only the bounding box that has higher IOU will participate the calculation, all the other bounding boxes in each iteration will not participate in the optimization process.

The second loss function will be the combination of cross entropy and SoftMax. Compared to the MSE, cross entropy will be more sensitive to the prediction error, so we will also transform the classification value from 0, 1 to 00, 01 by applying one hot technique.

4. The loss function of containing an object

The last function is the summation of the loss of containing an object and not containing an object. We will generate a score for each grid cell to predict the possibility of containing an object for the current grid cell. If the current grid cell contains an object, the value of L(obj) will be one, and the L(noobj) will be 0. We will only need to compute the loss of containing an object for the current grid cell. Oppositely, if the current grid has no object, the value of L(obj) will be 0, and the value of L(noobj) will be one. In this case, we will only need to compute the loss of not contain an object. λ is a constant that control the influence of each loss in the optimization process. Based on the training dataset, each training dataset will only contain one or two target that need to be detected in the training process, as a result, the grid cells that contain an object will be much less than the grid cells

that do not contain any target objects. In order to balance the prediction performance, the value of λ for containing an object in loss function should be set up larger than one and the value of λ for not containing an object should be set up less than one to decrease the negative influence of unbalance samples.

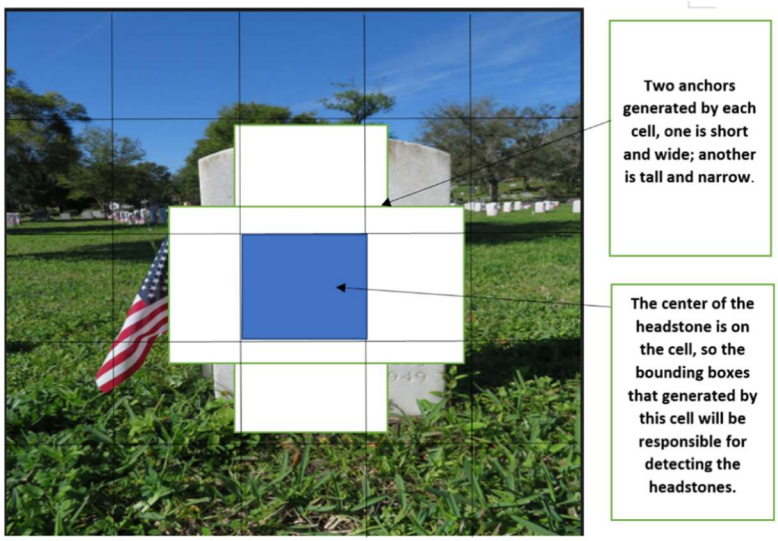

Figure 44 The process of generating anchors

In the training process, the output channels will have 12 values, as we talked, it includes location information of two anchors and the possibility of containing an object. In each iteration, each anchor will have a possibility value, we will only calculate the one with a higher possibility and compute the loss with the associated position information for updating.

De-noise Obstruction and Text Precision

Data Collection

The dataset will contain 500 pair of cropped headstone images which received from previous section. In order to train the model, we will define input data samples by using 500 noised headstone images, whereas the labels which is associated with each input image will be the cleaned and de-noised headstone image. In the training dataset, there will be some headstones which contains different kind of noise, in order to have an expecting result, in each round of training, the system will automatically add a random noise to each training data. Each of the cropped noised headstone dataset will be sent to train the autoencoder model to de-noise the obstructions and increase the precision of the text

region. Additionally, we will have 100 images as testing dataset to evaluate the performance.

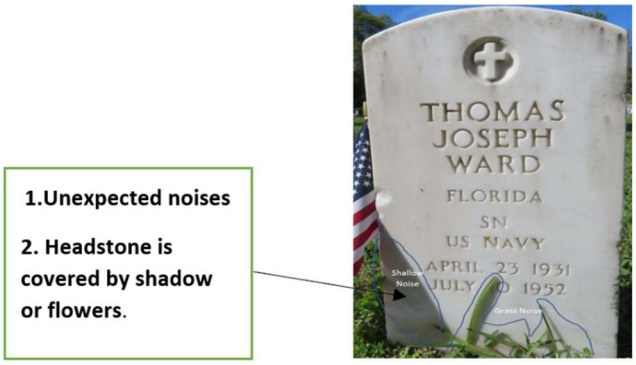

Figure 45 Noise on the images

After observation of raw headstone dataset (already rotated), partial of headstone in the image is obstructed by some object such as flowers and flag, and the text on the headstone is not clear enough for identifying. As a result, it will increase the difficulty of detecting text which are located on the headstone. In order to increase the accuracy of OCR process, we will introduce an automated system to eliminate the noise on the headstone. The idea of the system will need hundreds of pairs of headstone images. The input should include headstone images which are not clean and blurred, whereas the output label will be the cleaned images. In order to build the training dataset, we will collect 500 cleaned headstone images and manually add noise on it as the input. The output will be the original cleaned headstone images. The purpose of training process is to build an automated system to get rid of the noise.

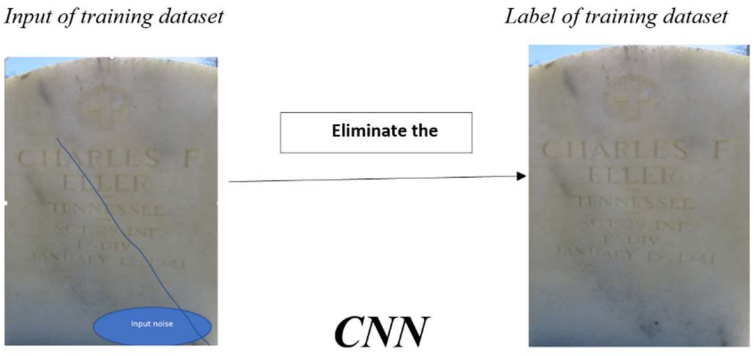

Figure 46 The left image showed the added noise images as inputs; the right headstone images will be used as a label.

The left image is the input dataset which has been manually added noises for training purpose; and the right image is the label in training phase; the purpose of this step is to train an auto system to get rid of different noises and create a clear output for the OCR process. We will design and operate an autoencoder model to solve those potential issues. The training dataset will include two parts: 1000 cleaned headstone images as labels; manually adding noises to the 1000 cleaned headstone images as input dataset.

Implementation of De-noise Obstructions

After cropping process, the headstone should be able to generate and as an input for the OCR process; however, lots of headstone edges and faces has been covered by some unexpected objects such as flags or flowers. For the purpose of having a higher quality input for OCR, the following section will introduce the process of implementing an autoencoder techniques to get rid of noise as much as possible.

The idea of de-noise autoencoder is to add noise to the input training data and make the autoencoder learn to remove the noise to obtain a real input that has not been polluted by noise. As a result, it will force the encoder to learn to extract the most important features and learn more robust representations from the input data. It's also the reason why its generalization ability is stronger than the other general encoders.

[17]Preprocessing the training dataset

After the headstone detection process, all the headstone datasets have been successfully cropped as a series of new dataset which will be sent to the OCR process; however, the text information on the images are more or less blurred and the edge of headstones have been obstructed by some objects such as flowers. We will create a set of training data that includes noise-added input and original-cleaned output to send into the autoencoder model.

We will be adding noise to each of the cropped training dataset based on the Gaussian distribution. In the most of the headstone's images, the noise on the images is randomly located, so in order to make the autoencoder model fit the reality as close as possible, in each iteration, the noise will be randomly assigned to each training dataset again and optimize the parameters in the model. The label of the model will be the original cropped dataset which has no noise on it. The purpose of this section is to eliminate the possible noise on the headstones to increase the accuracy for the OCR process.

1. Design and operate the encode section.

The encode section can be regarded as layers of convolution. The purpose of encode is to extract the feature from the image and transform the wide and short matrix into a deeper. In this section, we will also implement a convolutional neural network as the feature extractor. At the final layer of the encode, the image matrix will be completely transformed into a column vector which regarded as the input for the decode section. The encoding process of the original data X from the input layer to hidden layer:

$$\mathbf{h} = g_{\theta_1}(\mathbf{x}) = \sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right)$$

Equation 20 The unit function of encoder

2. Design and operate the decode section.

The decode section is the process of expand the column vector that generated from the encode section. Decode process has same layers of convolutional operation as the encode process. After lots of layers of un-pooling operation, the final output will be a matrix which has the same size as the original headstone image. Decoding process from hidden layer to output layer:

$$\hat{\mathbf{x}} = g_{\theta_2}(\mathbf{h}) = \sigma\left(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2\right)$$

Equation 21 The unit function of decoder

3. Loss Function

The loss of the autoencoder system is the difference between the prediction matrix and the original headstone image. In the training process, we will implement the backpropagation in each iteration to update the parameters. The purpose of autoencoder system is to train a system that could be able to take image as input and automatically eliminate the noise on the headstone image. The loss function is written as the distance between the X and $\hat{X}$:

$$l = \frac{1}{2}\sum_{i=1}^{N}(x_i - \hat{\hat{x}}_i)^2$$

Equation 22 The Loss Function of Autoencoder

Summaries and Expectation

At the end of the research, out system will be able to automatically (1) Read and preprocess headstone images; (2) Identify the class of headstone and predict the location coordinates; (3) De-noise the obstructions from headstones.

OCR

Objectives

The primary objective of the OCR portion of the application is to receive an image of a headstone and be able to extract the various text segments corresponding to that image. This will involve making text detection predictions using a convolutional neural network as well as built in python libraries in order to recognize the segments that were found using the detection model. The strings generated from this process will then be sent to the labeling portion where they are used as search tokens to find the corresponding data attached to that headstone.

Introduction

The motivation behind this project circulates around finding a method to ease the labor-intensive work done by the National Cemetery Association. Our sponsor Dr. Giroux, as well as other NCA workers currently compare images and data sets for thousands of individuals on end. The process entails taking photographs for each of the headstones in a certain cemetery and comparing those images with a csv file of data for all the individuals in the cemetery. This is done to rename each of the images for every individual so that they can be correspondingly labeled and identified by a unique key. The uniqueness will come from a combination of which section of the cemetery they are buried in as well as the location in that section. This way, each image can be correctly labeled and can be retrieved with ease when necessary.

The pictures will need to be rotated and cropped accordingly in order to have a standardized photo of the headstone. Some pictures may be taken at an angle, or some headstones may be aged to the point where they have sunken into the ground and are no longer upright. This is where the rotation comes into play. As well, the picture may be taken at a distance where there is space or even other graves in the background. This is where the image is cropped around the headstone with a small picture buffer so only the headstone is present in the new image. The next step is where my OCR portion begins, which will detect and pull the text from those images.

The application's functionality revolves around the ability to modify the headstone images in the necessary manners in order to read the lettering for the renaming process. This will be done through the process of optical character recognition. Once the photo has been processed through the rotation and cropping stages, the next task is to decipher the text of the headstone image using machine learning and computer vision techniques. The process

is simple for a human eye, but many factors will have to be involved to perform text detection and generate results with great accuracy. Many of the headstone images may have poor lighting, may be partially covered, have a very faint contrast between the text and the headstone, as well as many other challenges that will need to be addressed before any text detection takes place.

A set of predefined accuracies have been set by Dr. Giroux, where each portion of the project needs to reach a level of accuracy high enough for the following process to achieve their own consistent accuracy marks. The rotation portion of the project will optimally achieve success rates of 95%, which will then lead to a 5% loss for each of the remaining portions. The cropping will yield 90% accuracies, the OCR 85%, and the labeling system with 80% accuracy.

There have been many methods of achieving OCR, many of which are used in everyday applications. However, many of the OCR techniques on the market were designed to detect computer generated text, either from a computer application or from a piece of paper. OCR text detection for natural images takes the application a step further as the images can be affected by several factors that could compromise results. Factors include lighting, image distortion, angle of the image taken, and text spacing. As well, the issue arises with the lack of standardized font, spacing, and angles. This means that real world applications can have a multitude of colors as well as styles that the font can be. For our project, the OCR must be responsible for text that is oriented in a traditional horizontal manner, as well as text that is oriented in a downward arch format.

The application will thus be responsible for filtering the images into a state where text detection is possible. After that is achieved, the remaining steps are to detect the text, localize it, and then finally recognize the various text segments to pass to other parts of the application. The application for text detection will use the Attention OCR model. From there, we can perform any post-processing features in order to get the text into a legible format for the text recognition functions to classify the text.

The Attention OCR model is pre-trained using many standardized text data sets that cover a wide variety of factors that need to be taken into consideration. These data sets include over 30,000 test images with an estimated 100,000 text segments. The model should then produce accurate text detection results, but if not, another specialized data set could be implemented. This data set would be custom for the headstones that the application would factor in. This would ensure that the model could detect the text from the headstones if the base model does not suffice. In general, this step will most likely not be needed as the standardized pre-trained model should generate high accuracies of text detection.

Background and Research

Many methods of OCR as well as machine learning models were researched in order to come up with the best solution to the problem at hand. The main section of research came from the text detection step of the OCR. This step is essentially the portion that detects text from the image and places bounding boxes around those segments. The methods researched are displayed in *Figure 23.*

EAST Model

The first text detector researched was the East Text Detection model[18]. This model was researched for its high accuracies when detecting natural image text. The process of text detection is broken down into three parts, the FCN, thresholding, and lastly non maximum suppression. The FCN is a fully convolutional network which differs from most convolutional neural networks through the fact that it is not fully layered. Instead, the model performs many smaller convolutions, but is still trained with large data sets. The idea arose as there are many pre-trained instances of FCNs on data sites where it may be less practical to train the model from scratch. As well, thresholding involves the process of turning traditional images into a binary image, which can be better interpreted by digital processing. Lastly, non-maximum suppression is the process in which bounding boxes are created for, in this case, the text detection. The process will entail being able to create one true bounding box for the text segment from the many that may be detected. This method is discussed further in the document.

TextTubes

Some models were researched that were significantly considered and may be used as inspiration but did not meet the necessary standards that the project needs. One being TextTubes[19], which is a text detection process which creates “tubes” instead of bounding boxes around the text segments. These tubes are much more flexible as they have many different anchor points that can be placed around many different natural image texts. For example, text that is curved in any manner, and text captured at distorting angles can be achieved at high results. There are no general downfalls for this application, but the application may be too computationally expensive. TextTubes’s methodology is very good for different orientations of curved text. This model is very convenient for completely random instances of text detection, but is our application completely random? The headstones will only contain either horizontal or downward arched text. In this case, the process might be over-exhaustive if less computation is required.

Stroke Width Transformation

The second process is the process of Stroke Width Transformation[20]. This model might be the best model for detecting text in natural images. It involves finding the gradient direction of the edge pixels and generating the width of different edges. Since the text will follow similar structure and width as opposed to the other instances of edges, it would be easier to separate the text from the image if this process was used. The only issue, however, happens to be a critical one. The bounding boxes are structured in a way where there are only 4 bounding axes. This means that for text that is not completely horizontal or close to it, the method will not produce very accurate results. This would include the instance of arched text, which is needed for the OCR portion to function correctly for the NCA to use.

Figure 47: Top-Left: TextTube, Top-Right: EAST model, Bottom: Stroke Width Transform

As well, many of the images of headstones which will be used by the NCA will need to be pre-processed in a many that can create a sharper contrast between the headstone background and the actual image text.

The image shown below is the most extreme case of lighting discrepancy found in the images provided:

Figure 48: Example of Outlier Headstone with Low Contrast

This is an extreme case, but there will be headstones in which need some post-processing affects in order to be legible for the text recognition portion of the OCR. An idea was to use a Canny edge detection of the images that are being sent through the system. A Canny edge detector uses Gaussian filters in order to generate the x (horizontal) and y (vertical) derivatives of the to detect the edges of the image. This process would leave an inverted image where only the edges would appear. Since characters are only edges in themselves, they would be detected by the canny edge detection system. Then with a process of inverting the image back would leave an adjusted image where the edges would be black with most of the background noise removed. As well, a simple contrast filter that could more heavily distinguish lighter pixels from darker pixels could be necessary in ensuring that the text is legible on the headstones.

Case Study: EAST Model

We will dive deeper into one of the models studied, the EAST text detection model. This model can be broken down into three classifications, the Fully Convolutional Network, the thresholding process, and non-maximum suppression. The process can be summarized as using a trained neural network in order to predict the areas that contain text segments, then applying a thresholding function which will reduce pixel noise and eliminate false positives, while lastly adding a non-maximum suppression algorithm which will be able to use all the predicted text boxes to generate a bounding box of best fit to accurately surround and cut the corresponding text segments.

Fully Convolutional Network

The Fully Convolutional Network was chosen in this process to be a simple method that can work across images of varying dimensions while being less time consuming. Instead of a network that includes many dense layers, the FCN will instead use numerous

convolutions to classify pixels which will be the predictions for the text segments. *Figure 25* displays the overall structure of a FCN and the corresponding convolution layers. This process acts as semantic segmentation distinguishing the text (objects) from the background image.

The first area involves using the trained model in order to perform numerous convolutions that will determine specific features which can be extracted from the image. These features can then be merged together with similar features in a localized area which can allow for predictions to be made. When many features have been determined in the same area, a prediction bounding box can be made. These bounding boxes can be rotated to any degree to match the surrounding features. At the end, the neural network would output all of the predicted text regions, which have been signaled by bounding boxes. These boxes are only predictions of the features which may not be entirely accurate, which is where the post-processing effects come into place.

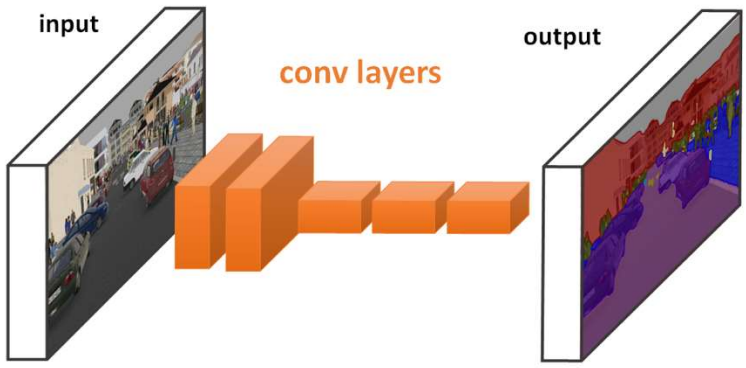

Figure 49: Example of FCN.

This model is very simple and cuts out much of the middleman as well as many different post-processing steps for the text detection phase and works for situations where semantic segmentation is key.

Thresholding

The thresholding process involves scanning through each local region and determining the pixel values in comparison to surrounding pixel values. The premise involves using a histogram for a portion of the image in order to determine all the local pixel values in that area. In order for a pixel to be determined useful, a thresholding value will need to be assigned and act as the median value, where values over the threshold are considered text, and under are considered background. In this regard, the background pixels will be ignored while the text pixels will be retained. The process as a whole is to reduce the number of false positives in the text regions by dismissing pixels that may have been recognized as

text but aren't actually text. As well, knowing where exactly the text pixels reside will allow for more accurate and efficient bounding box generation in the non-maximum suppression stage.

An example histogram is as follows:

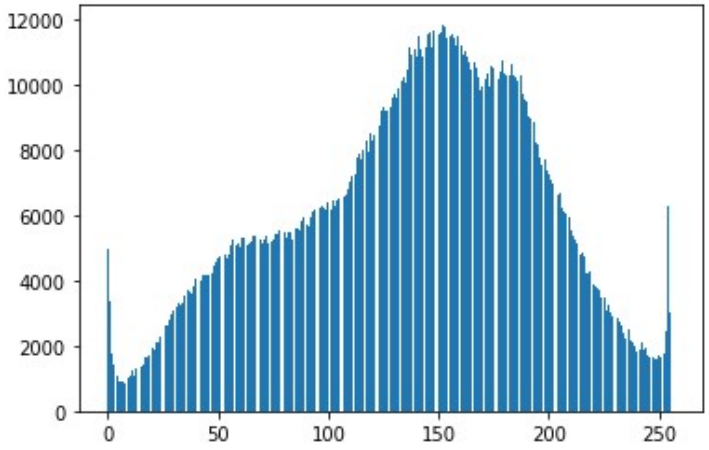

Figure 50: Example of Thresholding Histogram.

The y axis contains all the pixels in the region, while the x axis are the actual pixel values. A threshold value would be preassigned, for example we can use 155. If that were the case, all values under 155 would not be considered, while the rest would for the non-maximum suppression stage.

Non-Maximum Suppression

The process of non-maximum suppression will take into consideration all of the presuming bounding boxes created from the text prediction step, and with the image thresholding, generate a bounding box of best fit around the final predicted text region. This is done through a process called merging. From the FCN stage, multiple geometries (bounding box regions) are created from the various predictions made, so the NMS is the process that actually merges the predicted bounding boxes into one final bounding box surrounding the text.

Once the process of non-maximum suppression has concluded, each line of text will be surrounded by a bounding box that best fits that text. This bounding box will be the basis for every ensuing step which includes the non-linear regression function, post-processing, as well as the OCR text recognition.

The problem of the EAST Text Detection model is the lack of functionality when it comes to curved text regions. As stated early, being able to detect downward arched text as well as horizontal text is crucial in achieving high OCR results. The model does not use polygonal bounding boxes, which is needed in order to fully surround text that is not rectangular. So, if curved text was to be detected with the EAST model, features will have to be added which could be time consuming and may not function properly. An idea to do so could be to merge the different rectangular bounding boxes in the NMS stage, but instead keep many of the bounding box points and formulate a polygonal box around the text.

The idea is illustrated below:

Figure 51: Proposed Idea for NMS.

This could be feasible but relies on the rotation of bounding boxes and a rewriting of the NMS algorithm. Another problem occurs as the EAST model was created in C++, which in term has some limitations for the model when using it in python. A specific incident occurs as python does not have the same libraries and functions as C++, and vice versa.

In the end, this fault led to more research on different models until the right model which can handle both types of text were found. This model was the Attention OCR model[21] which relied heavily on attention mechanism.

Premise of Model

The model is as shown below:

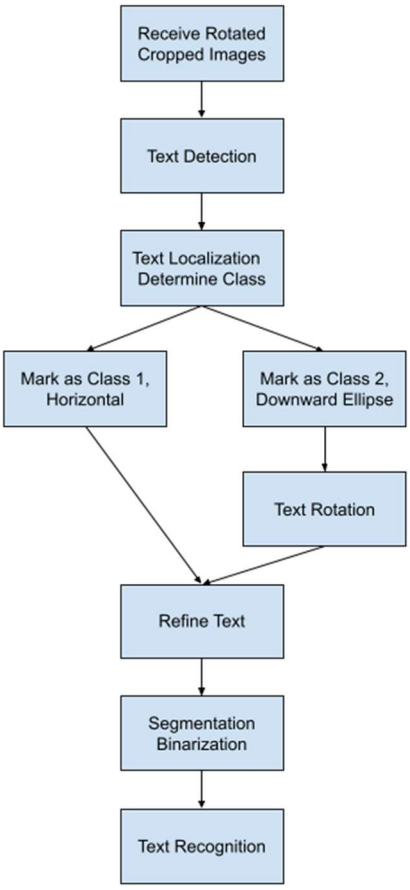

Figure 52: Overall View of OCR Model.

In essence, the model described will be able to determine the areas where text is present and localize those text characters. The Attention model predicts bounding box points in order to create the polygonal bounding boxes. After localization, a non-linear regression can be performed on the remaining polygonal text boxes in order to determine class. The regression will generate a plottable line which will represent the centroid axis of that bounding box. Then the slope of this line can be determined and used to measure the orientation of the text. The first class will be horizontal text and the second will be the downward arching text. The next step will be to refine the text to remove noise, and any filtering needed in order to correctly separate the text from the image. If the text is determined to be in class two, a rotation will be made on the text to straighten it out to resemble the text in class one. Then using built in PyTesseract functions, the text can be recognized. Only the text that could potentially be names or dates are sent to the labeling portion, and the rest are discarded.

Attention Mechanism

The Attention OCR model[22] is based upon the idea of Attention Mechanism[23]. The whole idea of attention revolves around the process of concentrating on certain aspects while being able to ignore others. As humans, this process comes easy and naturally. For machines, it is a process in order to have neural networks focus on some features deemed "important" and discard the rest.

Attention lies between the encoder and decoder and has several components which are displayed in *Figure 29*. One is the context vector, which is the information gathered from the encoder. The encoder processes the input data into this context vector which represents a sequence of the input images. This differed from the normal process as previously only one value of the vector was used (no attention). This context vector is used with an attention function to generate weights which will represent different aspects of an image (for example, sky, tree etc.). More relevant regions of the image are weighted higher, and thus are decoded with their corresponding target words. The weights are sent through a SoftMax and nonlinearity function which normalizes them so that they can represent probabilities. The other words are not decoded and are thought of to be "blurred" for the network.

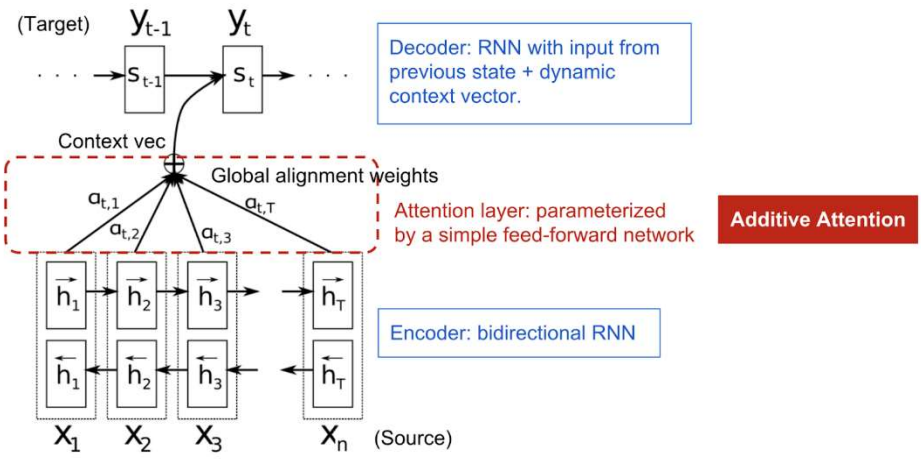

Figure 53: Attention Mechanism Model.

The encoder takes the input sequence and attaches weights. These weights produce the context vector for each input value, which can be decoded or not if the weight is high enough.

The formula for context vector is provided below:

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s$$

Equation 23: Context Vector

The context vector is described as the combination of the attention weight (ats) and the source image sequences (hs). The context vector is used to establish the attention vector represented by (at).

The equation for the attention vector is as follows:

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}_t; \boldsymbol{h}_t])$$

Equation 24: Attention Vector.

The attention vector generated is a specific context vector that is then processed by the decoder. This process provides a specific context vector instead of a generic context vector processed by the entire image sequence.

This process is followed in the Attention OCR model which provides the same approach to text detection. This model can detect many variations of natural scene text and is the basis for generating bounding boxes.

Text Detection

The model that is implemented into our headstone project is the Attention OCR model. This model uses the attention mechanism along with a convolutional neural network to detect text features from an image and classify the text regions consisting of similar character features. This process uses the attention mechanism model displayed in *Figure 29* and a CNN to generate the image sequences previously described. The model is broken down into two segments: the CNN, and the attention method.

CNN

The convolutional neural network is modeled after the Faster R-CNN architecture[24], which aims to surround objects (or characters in this case) with the corresponding bounding boxes. The structure for a R-CNN can be seen in *Figure 29.* The Faster R-CNN includes three selections:

- **Convolution layers:** processing input image to detect features that correspond to the objects used in training (for this purpose, characters).
- **Region Proposed Network:** neural network which makes predictions on the bounding box areas based on the features detected.
- **Classification:** CNN that takes the predictions made by the RPN and formulates the bounding box around the image (text region).

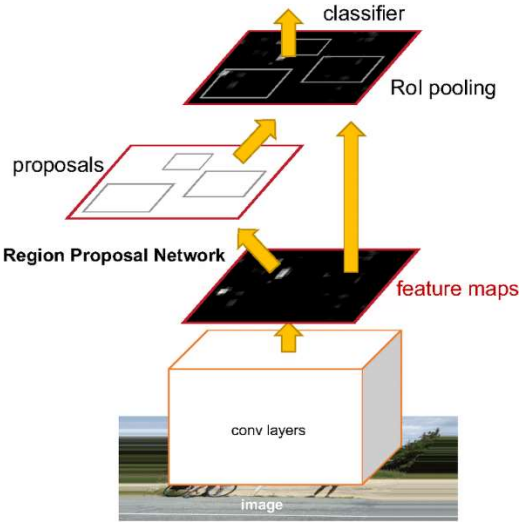

Figure 54: Structure of the Faster R-CNN followed in the Attention OCR Model.

The R-CNN is generated with a minimal loss function in order to predict character sequences with the highest level of likeliness to the ground truth.

Attention Method

The attention method used is very similar to the attention mechanism described earlier. The network takes these image sequences and runs them through the attention model in order to generate the necessary weights for the context vector to use. The Attention OCR process also uses a LSTM (Long Short-Term Memory)[25] network which is used to make the necessary predictions for items in a sequence. The LSTM is used along with the attention mechanism model in order to generate the resulting attention weights and context vectors needed in order to find which features to zone in on and which to ignore. In the end, the model is trained so that text features will be concentrated on and the rest will be dropped (thus where the attention name comes into play). The resulting bounding boxes will surround the various text segments and allow for those segments to be cut from the image and solely be the focus for the rest of the process.

The result of text detection is having the bounding box regions which are polygonal in shape and can fit both horizontal and downward arching text with high accuracies. The next step in the OCR process is to determine the classification of the text, and depending on the class, perform post-processing techniques to have text segments ready to be recognized by OCR libraries.

Determining Text Class

To define a text class, there are only two types of text structure that will be seen in the corresponding headstone data. The first is the traditional horizontal text, which will amass to most of the text that will be tested. This classification of text will be defined as class 1. The second class is for the rare occasions where the headstone contains arched text, a downward ellipse to be specific. The method of defining if a line of text will fall into which category entails performing a regression on the bounding box in order to find a central line through that box.

By using each of the bounding box points as individual points, we can plot those coordinates and perform a non-linear regression. All the points are independent points which would allow for the results to vary by whichever text image is being used. This process of plotting the points on an X, Y plane will allow for an accurate best fit line to be created and further evaluated based on which class it fits into.

An example of the created centroid line for a horizontal bounding box would appear as shown below:

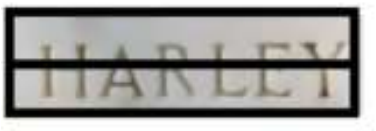

Figure 55: Centroid Axis Line for Horizontal Text Segment.

This process of non-linear regression is performed through creating a sigmoid function and running the coordinate points for each bounding box through to generate values for the prediction points. These prediction points are then mapped with matplotlib in order to display the actual regression curve.

The function can be defined as:

$$Y=f(X,\beta)$$

Equation 25: Non-linear Regression Function.

where f is the actual regression function, X is the list of predictions that will be created, β is the bounding box point coordinates, and Y is the function that is created. This function can be performed in python by using the SciPy library, which has a function labeled "least_squares". This library allows for parameters to be predicted given the data points and function. Once plotted, the regression line is then passed to the next section and represents a method of defining the text's class. Since the line was created with the bounding box points, it will have the same size as the images which allows for more information to be determined from this one specific line.

In order to find the class, we can further evaluate the regression line that is created from the above process. To do so, the derivative is taken of the regression line and evaluation of the derivative can decide the class. A linear line is represented as a line that has a constant rate of change. Since our horizontal class may not be perfectly rectangular, the results will need a cushion so that a class 1 line does not get mistaken for a class 2 line. If the results of the derivative are constant or close to it, we can infer and classify this text in class 1. If the derivative has sporadic changes from negative to positive (such that an upside-down parabola would have), then we can ensure that this will be a class 2 text. The process of both determining and handling each class is described below.

Class 1, Horizontal

The first classification of text will consist of horizontal text that will be bounded by a rectangular shaped box. For this instance, no huge changes will need to be made to the text segment, but post process filtering will need to occur for the OCR to detect the lettering at a high accuracy. This will include skewing to make sure the horizontal text is not slanted in any direction. As well, contrast filtering will be needed as many of the headstone photos gathered so far do not have great contrasts between the image and the actual text. The post processing will be detailed later in this document. The number of text segments of headstones would be near 95% in the class 1 category, with the second class being more the outlier.

In order to determine if a text segment is class 1, the derivative of that centroid line can be taken. In an ideal scenario, the derivative would be 0 as the line would be constant. This is not always the case when dealing with natural images, so some images may be horizontal,

but the axis line may not be perfectly straight. Thus, a threshold point would be taken to give some room for error. In this case, multiple points are taken from the derivative function and are compared to see if they are the same value or within the threshold value. If this is the case, then the text segment can be classified as class 1. If not, the text segment would be classified as class 2 and additional steps would be taken into consideration to straighten class 2 text into class 1.

Class 2, Downward Arch

After determining the class by using the corresponding bounding box and the centroid line, there needs to be a process to make the class 2 text legible to the computer. The goal is to determine if the text is class 1 or class 2 and if the text is class 2, then perform a set of actions to turn it into class 1. Essentially finding a method that can straighten the arched text into a horizontal legible pattern will be crucial to achieving good results for this data. I have thought of two methods of which to go about this action, they are listed below. They both follow similar patterns but vary towards the end. One involves creating a tangent line for the straightening process, as the other uses a central point to straighten the text segment.

Tangent Line Method

The whole premise of this function is to find the maximum point of the central line and use that point to be the basis of our new horizontal transformation, which can be seen in *Figure 31*. The first step is to take that central line and find its derivative. From here we can use that derivative function and find where the x value transitions from positive to negative. The point where this transition happens is a critical point and is the maximum of the parabola. We can conclude this because the only shape the arch can be in is an upside-down parabola, so the critical point is guaranteed to be a maximum point. Using this maximum point, we can find the tangent line of this point. Since it is a maximum point, the only tangent line that can be created is a line that is completely horizontal (Tangent line will have a slope of 0). Determining this line is the crucial step because it will correspond to the foundation of the new straightened out text line.

Figure 56: View of Tangent Line created on Maximum Point.

The corresponding tangent line will be the line in which we can merge our text onto. This process will transform the central line into the tangent line, and the corresponding image will transition as well. Once we perform this action, the text should be in a manner which can be legible by the pyTesseract functions and yield results of over 85% accuracy.

Ellipse Center Point Method

The second method also involves the centroid line but takes a different approach of actually performing the transformation. Instead of calculating the slope to find the tangent line, we look to find the overall shape of the line. Since the line will be arched in a roughly symmetric manner, we can infer that the line is in the shape of an ellipse or a circle. The goal is to use the portion of the line to trace where the remaining portion of the ellipse would be. Once having a complete circle, then the central point of the circle can be retrieved and the angle away from that central point will be used to straighten out the arched text, shown in *Figure 32*.

Figure 57: View of Circle Created from the Centroid Axis Point.

There are some built-in functions such as HoughCircles[26] (OpenCV[27]), which can find the center point of the circle, and logPolar[28] and rotate, which can transform that text into the desired horizontal formatting.

The reasoning for implementing the first method discussed instead of this type comes down to general feasibility. For the second method to work at a high level, the curved text would have to resemble that of a circle. But what if the text happens to be curved in the same manner but not necessarily as arched.

For example, the headstone below is classified as arched but not necessarily in the same circular pattern as the prior headstone:

Figure 58: Outlier in Which Ellipse Case Would Fail.

In a case like this, the second method may not know how to resolve the curved text and may ultimately fail all together. The first method is not reliant on circular patterns and can be achievable for all types of downward arching text.

Both methods are logically sound, but preferably the tangent line approach will be used. The ellipse center point method is dependent on finding the corresponding circle or ellipse to the parabola line created by the regression. If this cannot be found in specific cases, then the process of straightening out the text would fail and the text would be deemed unreadable. The tangent line approach may not have as much built-in functionality through python libraries, but there are less instances to fail once set up accurately. The next step after transformation is to follow the post processing methods described in the class 1 section in order for accurate OCR results to be achieved.

Post Processing

The next step in the process is to take those horizontal text segments and do any final post processing transformations and filters needed to guarantee high levels of accuracy when performing text recognition. Multiple precautions need to be considered as there are no guarantees for any type of image consistency in these images. For example, some images may have very high brightness from the sun glare, others may contain shadows. A contrast filter will be added in order to define the lettering from the headstone. The contrast will be useful for both dark and light pictures and sharpen the difference in pixel value.

An example of a post-processing contrast change is shown below:

Figure 59: Contrast Filter Being Applied

The larger text may not show a substantial difference but the lower text which features a smaller font is much more recognizable. After the text detection and classification, the OCR relies on this post-processing contrast filter to correctly be able to recognize those headstones that may have heavy shadows or lighting issues.

As far as modifying the images for skewing, the OCR libraries used to recognize the text will account for any skewing. This way, if an image is slightly not horizontal in its positioning, the OCR can still recognize and obtain accurate results. The only post processing besides straightening class 2 text is the contrast filtering, which will ensure sharper differences and yield higher results. The text segments will then be ready to move to the text recognition portion where they will be translated into words and used as searching criteria to find the appropriate individual in the data file.

Text Recognition

The following step is to take the generated image text segments and process them to determine the corresponding characters. The main libraries used in this instance will be PyTesseract[29] and OpenCV. PyTesseract is a Python wrapper to the Tesseract-OCR Engine. Tesseract is sponsored by Google and has been dedicated to providing efficient methods of OCR for many years.

The overall process for determining text from a headstone would resemble the image below:

Figure 60: Standard Class 1 Text recognition

The red boxes translate to the bounding boxes generated from the Attention OCR model. As stated, class 2 text has the additional step of straightening the text. Tesseract is able to recognize text that is on the same line, or on a similar trajectory. This enables the text to both recognize each individual character then segment them back into their specific words due to the amount of pixel space between them. Each character will be detected and surrounded by a bounding box, much like the text detection step, where their features can be extracted and then labeled accordingly due to the pre-trained model. As well, Tesseract can detect spacing which is how each individual word gets processed and separated from others. Built in functions allows for features to be recognized and processed to the point where the system can classify the correct characters[30].

An example of the single word character detection is shown below:

Figure 61: Example of Character Bounding Box Detection

It may appear that the text boxes overlap, but they are generated for each character, so each character is processed and then concatenated where needed in order to produce the words being described in the text segments.

There will be some images which include corroded lettering, or some may have the sun glare covering up a portion of the word. These cases will need to extract all of the information that they can, which for example may be “HARL**” if the last two letters of “HARLEY” cannot be read (Previous Figure). Any non-letters, numbers, or characters that

the dates will use (slash and dash) will be deleted if the Tesseract program uses these characters as foreign keys to hold undetectable characters. This will allow for partial searching on a name or data where the entire name or date could not be retrieved which may still yield high enough confidences to still label the image or give the user the chance to manually label the image.

Case Study: Natural Language Processing

The idea of natural language processing arises from trying to create an intermediary function between the OCR portion and the labeling portion of the application. The goal is to have some third-party model in order to read the string arrays that will be created from the text recognition portion and make predictions of what the data could represent. In general, 3 cases would be suitable for the text:

- **Potential Names:** Text segments that resemble the names of individuals. Many words can be discarded that will never represent a name and reduce the search to only those that can represent a name (Words such as “the”, “war”, “Private” are not potential names).
- **Potential Dates:** Any text segments that resemble common patterns for a date. In general, most segments that include a four-digit number (resembling a year) as well as any form of the twelve months will be added into this category.
- **Unneeded Information:** Military headstones contain information such as where the individual was from, what wars they fought in, what unit they were a part of, and sometimes a message about the individual. These are unnecessary as they will not be useful matching an individual in the data file.

The alternative would be to do an exhaustive search for all the text strings created. The main columns from the csv data file that will need to be searched are the first name, last name, date of birth, and date of death. These four columns, when presented with the correct text segments, will be able to find the correct individual in the data file in order to label an image with high confidence. If there is no process of text classification in place, then all the text segments that would be pulled from the headstone would have to be run through all the four columns previously mentioned.

For example, let's take into consideration the image previously discussed in the text recognition section which will be added below:

Figure 62: Example of Exhaustive Search Method.

For this headstone image, there are 8 text segments that are pulled from the headstone. Since there is no method in place to distinguish the text lines, all of them will have to be run through each of the four columns in order to find if one has matching elements. This means that for every column, there will be 8 partial searches, meaning a total of 24 total searches just for one headstone. If this is done for a data set of hundreds or even thousands of photos, the time and resources needed will be immense.

The method of labeling potential names and dates would reduce the search exhaustion by taking each of the text segments and using those fields only where necessary.

This process would look resemble this in contrast to *Figure 37*:

Figure 63: Example of Reduced Search Due to Classification.

As seen, there would be three potential names and two potential dates for this headstone. Instead of the previous exhaustive searching method, now we can distinguish which columns can be discarded and don't have to be run through the search at all. In this example,

there would be three lines removed from the searching process all together. As well, the potential names would only need to be used for searches on the first name and last name columns. The same goes for the potential dates, which will only need to be run through the date of birth and date of death columns. This means that instead of 24 total searches, that amount would be reduced to 10 (6 for the names and 4 for the dates). If this can be achieved at a high level of accuracy as well as drastically cutting down the number searches, then it is an ideal situation that would greatly enhance the application to a much higher degree. Research was done to try and find a method in which this could be feasible and implemented smoothly. In general, this feature is looked upon as not a necessity, but rather an idea that could make the application more efficient. Methods were researched but ultimately a decision was made due to the patterns found on most of the headstones being analyzed.

Bag of Words

The first method researched was a model called Bag of Words[31]. Bag of Words is a model in which text can be represented and classified using machine learning techniques. What we need the model to do is be able to make predictions on the headstone words for each text type previously discussed. The model would need to be trained with data sets with thousands of words that includes both names and dates. With more research done, the Bag of Words model did not correspond to what we were looking for in terms of natural language processing. Instead, the Bag of Words reads a block of text and builds a vocabulary that makes predictions based on that vocabulary. Features such as the occurrence of each word, and predicting a word based on surrounding words is the main premise behind Bag of Words.

The vocabulary is built as such:

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

Figure 64: Example of Bag of Words Technique.

Ultimately, this type of feature is not useful in the manner we need it to be. The model does not coincide with our process, and there doesn't seem to be data sets involving the text needed. This led to other research ideas to be further investigated.

Word2Vec

The second architecture researched was Word2Vec[32]. Word2Vec had similar features to the Bag of Words model. Word2Vec uses specific algorithms in order to translate words into vector numbers which can then be used to train the context of a word. This means that words are trained with surrounding words and can then be used to predict the occurrence of a word due to the context surrounding a particular word. Once again similar to the Bag of Words, this does not extract the features needed in order to make predictions on the style of words (name or date). This led to a reconsideration on what the actual task was and led to a simpler observation that could be just as effective.

Actual Processing Used for Classification

Once out of options surrounding neural networks, the idea arose to always use certain lines of text to begin the searching process. The lines are the first three and the last two text segments. Since the headstones that will be processed by this application will be military headstones, they generally follow particular patterns and standards. A pattern that has previously been mentioned is the orientation of the text, where it either is the traditional horizontal text or downward arched text. The pattern seen for this idea is that the first three lines include the first name on line one, and the last name on line two or three (with the middle name on line two if last name isn't). As well, the last two lines of the headstone include the date of birth on the second to last line, and the date of death on the last line.

An example of this can be seen below:

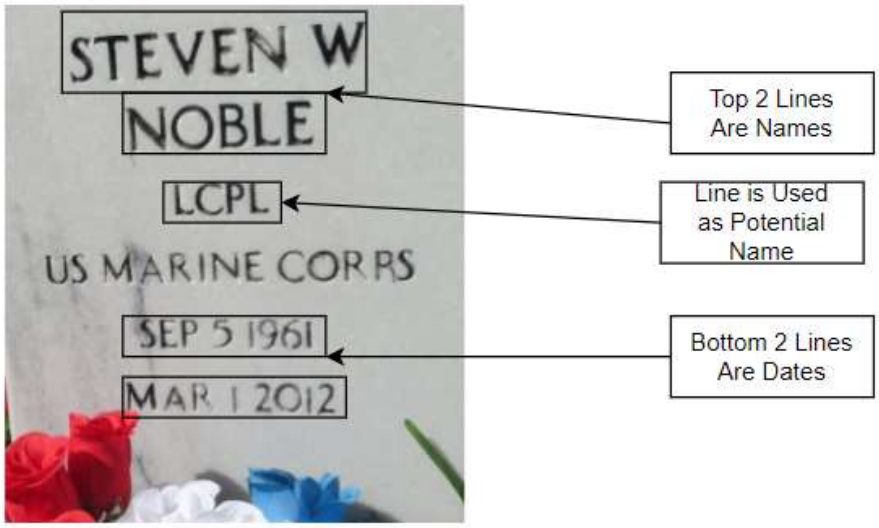

Figure 65: Example of Language Processing Chosen.

This approach would pick up the first three lines (“Steven W” and “Noble” included) and label those as potential names and the last two lines (“SEP 5 1961” and “MAR 1 2012”) and label those two as potential dates. This approach can only work if the majority of headstones follow this generic pattern. If not, there will be inaccuracies as the searches will not be using the proper text generated from the OCR. From analyzing the data sets provided as well as per Dr. Giroux’s discretion, this format would be acceptable as very few cases would not fit this process. The cases that would be considered rare cases (headstones that don't follow this format) could still be acceptable as some of the text segments could still represent what they are supposed to and achieve high enough confidences to label the images correctly.

Text Segmentation

After the text segments are generated as strings from the text recognition portion, they are then sent to the next part of the OCR process, which would be the natural language processing section. But before this, an idea was thought of in order to ease the remaining steps. This idea was to split each of the text strings that were created by spaces so that each line held on one word and not multiple words. A headstone that has the entire name on one line would have a text segment with each word on the same row. For example, if the first line of the headstone read “John Doe”, then the first text segment would also read “John Doe”. The idea was to separate the words so that the first string would be “John” and the second would then be “Doe”. This idea was thought of in order to have individual words to search through the columns of the csv file as well as enhance the natural language processing stage. If the lines were split, the text classification model can see every individual word and make a prediction rather than having to split these segments during this process. This idea would be acceptable and effective for most of the text on a headstone but would degrade a specific section that is crucial for the labeling process to occur.

The problem that occurred when researching this method was regarding the birth and death dates.

Looking at the following headstone segment provided:

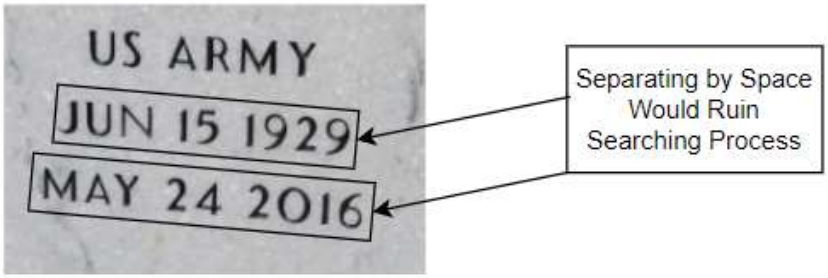

Figure 66: Failed Methodology of Separating by Space.

Each section of the date is separated by a space. If the proposed method was created, then there would be three different strings for each date. For example, the first line would have "August", second would be "25", and third would be "1897". There would be no reliable way to search the dates in this manner. Thus, the names and other strings will have to be kept on their same lines and methods of partial searching would have to be used in order to get accurate searching results. Since there were no discoveries for a method of natural language processing, separating the lines is not needed and would not matter towards the searching process. Instead, a partial search on each text segment would suffice in finding the corresponding individual in the data file.

Data Training/Testing Sets

Working with text detection is complex, but there have been numerous discoveries and research methods done over many years which have changed the landscape of OCR. In OCR and text detection, there are many models which have been created. These models are normally entered into OCR competitions and trained with standardized data sets using imageNet[33]. These data sets include tens of thousands of images with each image including various text segments. In contrast, the rotation and cropping sections of the project are very niche instances of machine learning and computer vision which will need models and data sets built from scratch. For text detection, the model can be pre-trained using one of the defined models that have become standards in the OCR scene. For this example, the model being pre-trained is the Attention OCR text detection model. In this instance, if accuracies from the pre-trained model are not to the level needed, an approach could be to also refine the model afterwards. The model could then be trained with headstone data generated from the data sets received from Dr. Giroux, where the neural network would be able to predict the headstone data with higher results. Since there are only two different text orientations to train the model with, there will be less ambiguity and higher results should be yielded than if the model was being applied to random natural image text.

The original data sets that the Attention OCR model is trained with are standardized data sets that are used to train many of the robust text detection models that contend in OCR competitions around the world. The three data sets are ICDAR2019-LSVT, ICDAR2019-ArT, and ICDAR2019-ReCTS. Each of the three are data sets from different competitions in the ICDAR 2019 Robust Reading Challenge[34] where the top OCR models receive cash rewards. All three have over 30,000 training images in order to train the model effectively.

The potential headstone data sets that could be used will consist of both a horizontal text set and a curved text data set. This would be separated into 700 horizontal text images and 300 curved text images. This is because the horizontal text will be more present in the actual applications functionality, so the model should be trained accordingly. The testing data would range at about 300, where 250 would be of horizontal text and 50 would contain the curved text. As well, the data sets need to ensure there are images that range in shadowing, lightness, contrast, as well as images that could be blocking some of the lettering. If this step is not needed in the event that the model already yields high results, the headstone data set will be entirely used as testing data in order to ensure effective results throughout the entire project creation duration. This is most likely the case as the Attention model was designed to detect natural image text, which is the goal of the OCR portion of the application and should not encounter failures in that regard.

Summary

The necessary step that the OCR section must complete is being able to extract the text from each headstone and send those segments to the labeling section, which can search for a matching individual in the csv data file. The first step is to use a pre-trained Attention OCR model to perform the text detection portion of the OCR. This will involve detecting features from the image and making predictions on where the bounding boxes will be for each text segment recognized. After this step, the bounding boxes are sent through a non-linear regression function which will determine which class the text belongs to. Class 1 is horizontal text and class 2 is downward arching text. The class 2 text is then transformed into class 1 by straightening the text into a horizontal format using a tangent line algorithm. A post-processing contrast filter is applied to make values between the headstone and image more defined. Then, using PyTesseract and OpenCv functions, the text is retrieved from the text segments, and the potential names and dates are sent to the labeling portion of the application. Please refer to Figure X which displays a headstone image and the corresponding text that is deciphered from that image.

Labeling System

Objectives

It is the goal of the labeling system to assign to each image an identifier (label) that provides descriptive information regarding the headstone featured therein, which can also then be used as a unique filename for the image. This information can be used in conjunction with the data file to determine all other information related to the headstone.

The labeling system and labels produced thereof must have the following properties:

- ***Uniqueness.*** Each image must be assigned a unique label which can be used to differentiate it from other images. Most notably, two headstones bearing the same name must have two different labels.
- ***Determinism.*** The process for determining the label must avoid the use of randomness. A given input must always produce a particular output.
- ***Reversibility.*** Just as the headstone must be used to produce a single label, the label must be able to be used to determine which headstone to which it refers. This applies primarily to the data file entry, not the image, since the label will be attached to the image as its file name.

Fundamentally, then, it is the objective of the labeling system to form a bijection between the content of each headstone photograph, based solely on the text recovered during the OCR process, and the entries in the data file.

OCR Cleanup

The OCR portion of the system reads the text on the image, but the text needs to be categorized based on what it is. When the OCR reads the text, it first places bounding boxes around each horizontal line of text and parses the text within each of those bounding boxes separately. This means that while we can't know with absolute certainty which text corresponds to the name, birth date, death date, conflict, or anything else, what we can know is which text appeared first on the headstone close to the top, which appeared next, and which appeared last near the bottom.

Using this, we can simply assume that the first 2-3 lines of text on the headstone is the name (2 if there is no middle name or if the middle name is just an initial, 3 if there is a middle name), the last line is the death date, and the second-to-last line is the birth date (if the birth date is present at all). There is also the added level of difficulty from the fact that

the bottoms of the headstones are sometimes obscured, in which case we may not read the dates, but we can easily verify that a line of text is a date by checking that it has a 4-digit number in it that begins with 17, 18, 19, or 20 (typically just 18 or 19).

This approach of assuming where each component is expected to be on the headstone works because most of the headstones follow the military standard. We believe that the proportion of the time this approach is wrong is likely to be less than the proportion of the time a machine-learning-based classification system would be wrong, which is the other approach we considered.

Entry Matching

Now that the text on the image has been classified as the First Name, Middle Name, Surname, Conflict, Birth Date, and Death Date, this information can be compared against the entries in the data file to find potential matches. If one of these fields could not be determined from the headstone, it is ignored for the matching process.

First, the entries in the data file are filtered based on those that exactly match both the First Name and Surname fields. If there are multiple exact matches, these are filtered again by Middle Name, then by Conflict, then Death Date, and finally Birth Date, stopping early once there is exactly one match. If there are no exact matches, the program instead attempts a fuzzy match, provided the Fuzzy Matches option is not set to "Reject" during initialization.

Runtime Options

The initialization screen will have a number of options available to the user to change the manner in which the program executes during the labeling system. These can be used to fine-tune the system to the user's needs at the time of execution.

Feedback Options affect how much user feedback is requested during program operation. A high level of feedback would require active user participation during the execution of the program, while a low level of feedback could run on an idle device while the user leaves to go do something else. The feedback options are as follows:

- ***None.*** No user feedback requested at any point. All unlableable files moved to the feedback folder and not added to the feedback queue.
- ***Partial.*** User feedback is requested for confirmation when the program is uncertain, but not for when the program is clueless. Examples include fuzziness confirmation

and selecting one entry from a curated list of possible matches. In theory, the user should never have to press the "search" button on the feedback screen.

- ***Full.*** User feedback is requested for all anomalous files, including those where the program has found no matches whatsoever. For these cases, the user will need to search for the entry manually.

Fuzziness Options control how much the system is allowed to rely on fuzzy matching when comparing the text on the headstones as determined by the OCR process to the information contained within the data file. A high level of fuzzy matching would result in better automation and less reliance on user feedback, but could result in improper naming. The fuzziness options are as follows:

- ***Reject.*** Fuzzy matches are not considered at all.
- ***Request Confirmation.*** Whenever the program would like to use a fuzzy match, the user must provide feedback to confirm that the fuzzy match is in fact correct. This option is not selectable if the Feedback setting is set to "None".
- ***Accept.*** The best fuzzy match is treated as if it were a perfect match and the image is labeled accordingly.

The **Fuzziness Threshold** is a numerical value from 0 to 100 entered in a text entry box, with a default value of 50. A higher threshold means that fewer fuzzy matches will be considered, and a lower threshold means that more fuzzy matches will be considered. If fuzzy matches are allowed, but no entry in the data file achieves a fuzziness score greater than the threshold, the image is treated as having no matches. A threshold of 100 is identical to not allowing fuzzy matches at all.

The **Label Format** option is a text string entered in a text entry box, with a default value of "{Section}-{Row}-{Site}-{Side}". The label format determines what information from the data file will be used by the labeling system to label the images. Any text enclosed by curly-braces will be treated as the name of a column in the data file, and the label for an image will consist of the format string with these enclosed sections replaced with the corresponding values in those columns of the matched entry.

Fuzzy Matching

To find fuzzy matches, we use the "fuzzywuzzy.fuzz.ratio" function from the "fuzzywuzzy" python library[35]. This function compares two strings and returns a value from 0 to 100 representing how close of a match it is based on their Levenshtein Distance[36].

In order to account for the multiple fields, we set the Fuzziness Score of a particular entry to the average of the scores of the First Name, Middle Name, and Surname fields (rounded down). This is the value that is compared against the Fuzziness Threshold set by the user during initialization (See: "User Interface: Runtime Options"). Any scores below the threshold are not considered.

If the Fuzzy Matches option is set to "Request Confirmation" during initialization, then any fuzzy match will need to be confirmed via user feedback regardless. As a result, there's no point in attempting to determine the true best match, and the program will just select the top five highest scores to show to the user for confirmation.

If the Fuzzy Matches option is instead set to "Accept", the program must attempt to find the single best match. To do this, the entries whose scores exceed the threshold are iteratively filtered until only one remains. First, the year of death must be an exact match. Second, the year of birth. Third, the conflict. Fourth, the month of death. Fifth, the month of birth. Sixth, the day of death. Seventh, the day of birth.

The only way for there to be more than one remaining entry is if there were two people with similar names who fought in the same conflict and were born and died on the same exact day. This is probably never going to happen, but if it does the program selects whichever had the greater Fuzziness Score to begin with.

Label Formatting

Once the correct data file entry for the image has been determined, we label the images based on the data file columns specified during initialization in the "Label Format" field. By default, we use the location of the headstone within the cemetery. The formatting pattern for the images will be each of the properties listed below, separated by hyphens:

- ***Section.*** A unique abbreviation code specific section of the cemetery in which the headstone depicted in the image is located.
- ***Row.*** A zero-padded two-digit numerical value denoting the row of the cemetery in which the headstone depicted in the image is located.
- ***Site.*** A zero-padded two-digit numerical value denoting how many gravesites over from the left end of the row the headstone depicted in the image is located.
- ***Side.*** A single letter, F or B, denoting whether the image is of the front side or the back side of the headstone, respectively.

For example, a picture of the front of the headstone located at row 2, site 5, in section REG would be "REG-02-05-F".

Since no two headstones can be located in the same physical location, by naming the files based on the location of the headstone necessarily there will be no conflicts where two files would receive the same name. Other file name schemas could result in this conflict. Had the naming schema used the name of the deceased, for instance, there could be a conflict should two or more persons have the same name, which does happen occasionally.

Double Assignment

Perhaps the most challenging obstacle with the labeling system is the issue of double assignment. That is, while direct comparison between the OCR headstone text and the data file entries can be used to map a particular photograph to one of the entries, how does the program guarantee that no two photographs are both mapped to the same entry?

It is noteworthy that this problem only occurs when fuzzy matching is enabled and the confirmation thereof is not required. If fuzzy matching is disabled, only perfect matches are allowed and the event of multiple perfect matches is already accounted for. If user confirmation is required for fuzzy matches, then we simply assume the user is correct. The double assignment problem is therefore only relevant when the best fuzzy match is incorrect. (Technically it could also occur when a photograph is a perfect match for a single wrong entry, and not a perfect match for its correct entry, but this is negligibly unlikely).

As an example, suppose photographs 1 and 2 are to be mapped to entries A and B, and that the correct mapping is 1 to A and 2 to B. Suppose that photograph 2 has very clear text and is a perfect match for entry B, but the text on photograph 1 is distorted such that its fuzz ratio is actually higher for entry B than for entry A. In other words, both photographs are mapped (assigned) to the same entry.

A crude solution would be to simply mark a given entry as “used”, and to no longer consider it for assignments, instead just passing over to the next-best candidate. However in the example above, if photograph 1 is processed before photograph 2, it could “steal” entry B away from it, and then they would both be wrong. Alternatively, we could simply require user feedback on all collisions, but that doesn’t solve the problem that in the above example the collision occurs while processing the true photograph, after the mislabeling has already taken place.

In addition to marking an entry as “used”, we must also store whatever its fuzziness score was at that time. This is the purpose of the “Fuzziness Score” column of the data file. If another photograph is mapped to the same entry later and has a greater fuzziness score (or a perfect match), the new image gets the assignment and the original image needs to be reassigned. This could result in a reassignment cascade, wherein the second-best match for

the reassigned photograph is itself already used and must also be reassigned to its second-best match, and so on, though this is unlikely.

Complete Process

The complete labeling process of fuzzy matching, checking for double assignment, and labeling, is as follows. The name printed on the headstone has already been determined (as best as possible) by the OCR portion of the system. If the process requests user feedback, but feedback is set to "None", send the image to the feedback folder. If the process requires fuzzy matching but fuzzy matching is set to "Reject", send the image to the feedback folder. In both cases, also store a corresponding metadata file to contain this information.

- Determine best match
    - Calculate the fuzziness score for each entry in the data file against the name on the headstone.
    - If exactly one entry has a score of 100 (perfect match), select it.
    - If more than one entry has a score of 100 (multiple perfect matches), request user feedback and select whatever the user indicates.
    - If no entries have a score above the threshold (no fuzzy matches):
        - If feedback is set to "Full", continue as below ("some fuzzy matches"), but ignore the threshold.
        - Otherwise, send the image and metadata file to the feedback folder.
    - If no entries have a score of 100, but some are above the threshold (some fuzzy matches):
        - If user confirmation is enabled, choose the top 5 results above the fuzziness threshold and request user feedback, selecting whatever the user indicates.
        - If user confirmation is disabled (the Fuzzy Matches option is set to "Accept"), select the entry with the greatest score.
    - In any case, record the score of the selected entry in the data file.
- Check for double assignment
    - If the selected entry has already been selected (indicated by a fuzziness score already recorded in the data file), the entry has been double-assigned.
    - If the existing score is greater than the score for this image, repeat "determine best match" as if the selected entry was not in the data file.
    - If the existing score is less than the score for this image, load the image that was previously assigned to the entry (along with its headstone text that was saved with it), and recursively evaluate this complete process for it as if the

selected entry was not in the data file. Update the score of the selected entry accordingly.

- Label the image
    - Change the filename of the image to according to the label format.
    - Place the image in the Completed folder.

Trade Study: Fuzz.Ratio or Process.Extract for fuzzy matching?

The fuzzywuzzy python library includes both the fuzz.ratio and process.extract functions for finding a fuzzy match score between strings. Ratio compares two strings, and outputs the score between them. Process.extract compares one string against an array of strings all at once, and outputs the scores between each of them. For our labeling system, we need to compare a guess string against every entry in the data file in order to determine which entry is the closest match. One would think, then, that process.extract would be more suited for this, but is that true?

To test which one is actually better, we prepared a simple test. Given a guess string and a data file, which function could return the top 5 highest fuzzy matches the fastest? We wrote a function for each one. For these tests, we will only compare against the first and last names.

For ratio, we also extract the section, row, and site from the data file. We then use a list comprehension to create a list of fuzziness scores for each of the names, and create a new dataframe that is a copy of the existing one, but with a new column to hold the fuzziness ratios. We then sort the dataframe by the column, and return a slice of the first 5 rows. This has the added benefit of returning a dataframe, and also the section, row, and site of that dataframe, which we would need later for the labeling system.

For process, we simply call process.extract on the names list with a limit of 5 items and return it. This returns just a list of tuples of names and their ratios, not the other info.

Despite process doing less work and being explicitly designed for matching against long lists of strings, in all of our tests it consistently took 4-8 times longer than ratio. This surprised us, and it’s why we used ratio for our program.

The full code we used to run this test is shown below.

```
from fuzzywuzzy import fuzz, process
import pandas as pd
import time

def extract_names(df):
    df = df[['First Name', 'Surname', 'Section', 'Row', 'Site']]
    names = df[['First Name', 'Surname']].values
    names = [" ".join(name) for name in names]
    return df, names

def ratio_test(df, names, guess):
    fuzz_scores = [fuzz.ratio(name, guess) for name in names]
    names_and_scores = df.assign(Fuzziness = fuzz_scores)
    names_and_scores = names_and_scores.sort_values(by=["Fuzziness"], ascending=False)
    return names_and_scores[0:5]

def process_test(names, guess):
    return process.extract(guess, names, limit=5)

guess = "JOHN WEST"

df = pd.read_csv('data.csv')
df, names = extract_names(df)

t1 = time.time()
ratio_matches = ratio_test(df, names, guess)
t2 = time.time()
process_matches = process_test(names, guess)
t3 = time.time()

print(t2-t1)
print(t3-t2)
```

Figure 67: Test Code for Comparing Fuzz.Ratio and Process.Extract

General Trade Studies

Should We Utilize Google Cloud Vision?

The Google Cloud Platform (GCP) Vision API is an already-existing computer vision API offered by Google[37]. The software used in the GCP Vision API is the same used in the free "Google Lens" app available by default on all Android devices. Preliminary examination of the Google Lens app demonstrated that the GCP Vision API is extremely powerful at certain tasks, especially text detection. It was for this reason that we considered attempting to leverage this existing framework for use in the headstone photograph processing system.

According to the Cloud Vision API website, the platform supports the following features: crop hints, document text detection, face detection, image properties, label detection, landmark detection, logo detection, object localization, safe search detection, text detection, and web detection. Of these, the most intriguing to us for this program were the crop hints, which are used to "determine suggested vertices for a crop region on an image", and the text detection, which could be used by the OCR section of the program to detect the text on the headstones. We could not determine any directly-applicable features to aid in the rotation process.

With these two features we could do one of two things: either we could apply them directly for the actual operation of the program, or we could use them merely for the training process in order to aid in the training of our own custom-built models.

If used for training, we essentially just spare ourselves some of the hassle of manually preparing training data. We would still have to develop and train our own models, so while this would save us some time, it's not really worth it.

Direct application would, of course, be extremely convenient for the construction of the program. In theory, the machine-learning system at Google is likely to be much more powerful than anything four college students could whip up in the span of six months, and this would allow the cropping and OCR sections of the program to be extremely streamlined. There are two caveats, however. First, the GCP models are generalized to all images, not explicitly headstone images; by training our models only on headstones, it could potentially be more powerful in performing that specific task. Second, the GCP Vision API is, as the name would suggest, a cloud service. Meaning, its use would require a stable internet connection and the program would need to talk out to Google servers multiple times for every image, which could severely slow down the program's processing speed and "no-network" simplicity.

Since GCP is a paid service, relying on it would make the program pay-to-use. At a price of $1.50 per 1000 images for text recognition, and an equal charge for crop hints, performing both operations once each on 50,000 images would cost $150[38]. While this isn't completely unreasonable for a senior design project, it is certainly not remotely desirable.

Ultimately, the final consideration for whether or not we should use the GCP Vision API for the headstone photograph processing system is this: This is our Senior Design project, and as such there is a grander goal before us than simply spitting out a program that works. We're here to build something, to learn something, and to acquire a story to tell. No matter how convenient, effective, optimized, or practical it may be to use preexisting utilities, there comes a point where the tool is just a little too good. Theoretically, this API would allow us to bypass up to 90% of the work and challenge of the cropping and OCR portions of the project, and in the process we would be losing all opportunity to learn about interesting machine learning concepts. For this reason (and money), we will not be using the GCP Vision API for this project.

Administrative Content

Budget and Financing

This project will only use open-source software (mainly python libraries). The final product is a locally-installed application that does not need to be hosted anywhere, and the graphic design requirements are minimal (could be done with only basic shapes with Tkinter python).

We would like access to the UCF Newton GPU for Machine Learning Training.

Milestones

Team Milestones

Milestone	**Date**
Begin Research	2/12
TA Check-In 1	2/16
Research Complete; Begin Designing Models	3/1
Project Review	3/1
Begin Data Collection and Preprocessing	3/1
TA Check-In 2	3/29 - 4/09
Model Design Complete; Start Prototype Build and Training	4/1
Prototypes Built and Trained; Start Finishing Design Document	4/20
SD1 Final Design Document Due	4/28
Start Building Models (Version 1)	5/1
Models Complete; Start Training	6/1
Training Complete; Evaluate Performance	6/15

Evaluation Complete; Start Building Models (Version 2)	6/20
Models Complete; Start Training	7/10
Training Complete; Apply Final Touches	7/20
Summer C Ends	8/1

Individual Milestones

Week	**Yubo**	**James**	**Trevor**	**Robinson**
2/15	Research Object Detection	Research Python user interfaces	Research OCR Methods	Research Edge Detection Techniques
2/22	Research Object Detection	Research Python Threading	Research OCR Methods	Research Edge Detection
3/1	Research TextField	Preprocess Training Data	Research and Familiarize with PyTesseract Techniques	Research Image Preprocessing
3/8	Research AutoEncoder	Preprocess Training Data	Research and Familiarize with PyTesseract Techniques	Research Frequency Domain Transformation
3/15	Research VOC_2012&LabelImg	Preprocess Training Data	Sketch/Plan model, get general structure outline	Research Frequency Domain Transformation
3/22	Labeling Training Data	Design UI	Designing the Model	Begin Designing Model
3/29	Labeling Training Data	Design UI	Gather/Preprocess Data	Begin Designing Model

4/5	Labeling Training Data	Prototype UI	Design Document (20 pages complete)	Gather training data
4/12	Labeling Training Data	Prototype UI	OCR Model design	Label Training Data
4/19	Design and Building Detection Model	Prototype Labeling System	Finish Final Design Document, revision stage, plan presentation	Data Preprocessing
4/26	Building Detection Model	Finish Final Design Document, revision stage, plan presentation	Finish Final Design Document, revision stage, plan presentation	Build rotation model
5/3	Training Model	Preprocess More Training Data	Begin Training Model	Train Model
5/10	Training Model & Optimizing Hyperparameters	Redesign UI	Letter Cropping Complete	Train Model, gather more training data
5/17	Optimization Model	Redesign UI	Segmentation/ Feature Extract	Tune Hyperparameters
5/24	Optimization Model	Build Final UI Implementation	CNN training	Evaluation
5/31	Merge With Other Components	Build Final UI Implementation	CNN training	Merge with other components
6/7	Merge With Other Components	Redesign Labeling System	Evaluate/Plan for second version	Evaluation

6/14	Evaluation Model	Build Final Labeling System	OCR Refinement	Evaluation
6/21	Evaluation Model	Redesign User Feedback System	OCR Refinement	Final System
6/28	Final System	Implement Final User Feedback System	Retest System, determine accuracies	Final System
7/5	Final System Complete	Final Full System Integration	Final System with Correct Accuracies	Complete system
7/12	Finish Final Document, Plan presentation	Finish Final Design Document, revision stage, plan presentation	Revision, Plan Presentation, finals tests	Finish Final Document, Plan presentation
7/19	Finish Final Document, Plan presentation	Finish Final Design Document, revision stage, plan presentation	Revision, Plan Presentation, finals tests	Finish Final Document, Plan presentation
7/26	Project Complete	Project Complete	Project Complete	Project Complete

Gantt Chart

Below is a Gantt Chart spanning both the Spring 2021 and Summer 2021 semesters, detailing our targeted completion schedule of the project.

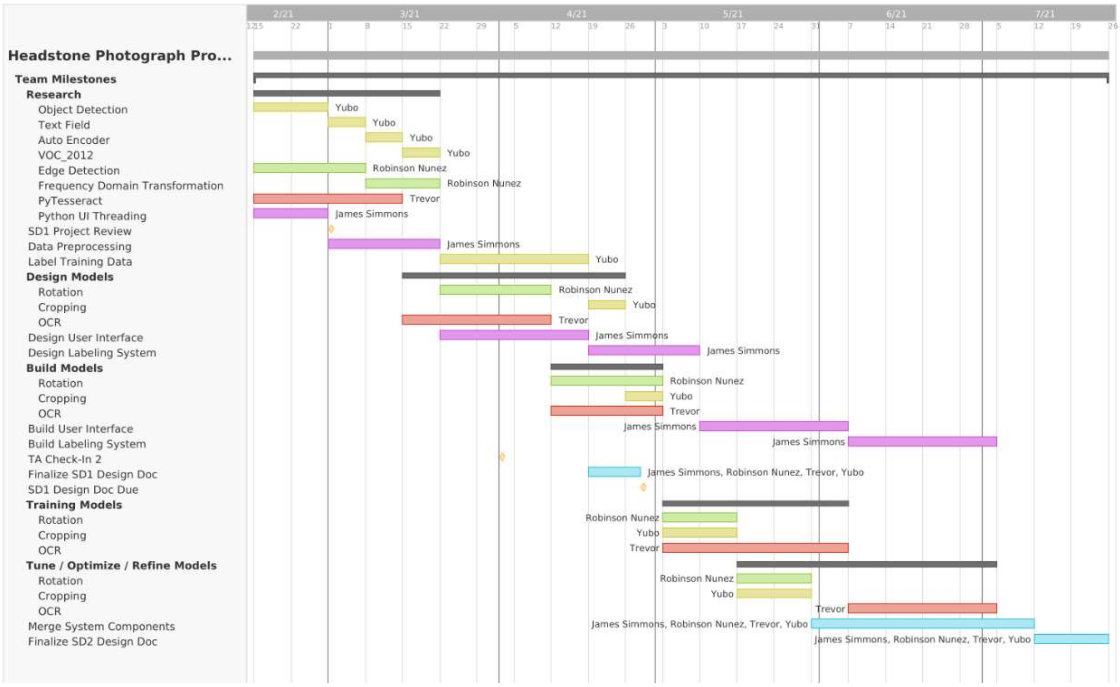

Figure 68: Project Gant Chart

Table of Figures

Table of Equations

[1] *BillionGraves*, billiongraves.com/.

[2] "Millions of Cemetery Records." *Find a Grave*, www.findagrave.com/.

[3] Chen. (2019). A Robust Vehicle Detection System by Using Tracked YOLO Networks. Amsterdam University Press.

[4] Glob - unix style pathname pattern expansion. (n.d.). Retrieved April 26, 2021, from https://docs.python.org/3/library/glob.html

[5] LeoHsiao1. (n.d.). LeoHsiao1/pyexiv2. Retrieved April 26, 2021, from https://github.com/LeoHsiao1/pyexiv2

[6] PyInstaller Quickstart. (n.d.). Retrieved April 26, 2021, from https://www.pyinstaller.org/

[7] Tkinter - python interface to tcl/tk. (n.d.). Retrieved April 26, 2021, from https://docs.python.org/3/library/tkinter.html

[8] Torre, V., & Poggio, T. A. (1986). On Edge Detection (Vol. PAMI-8, pp. 147-163, Rep.).

[9] Lohmann, A. W. (1993). *Image rotation, Wigner rotation, and the fractional Fourier transform* (10th ed., Vol. 10, pp. 2181-2185, Rep.). Erlangen, Germany.

[10] Brownlee, J. (2019, July 05). Convolutional neural network model innovations for image classification. Retrieved from https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/

[11] Sahir, S. (2019, January 27). Canny edge detection step by step in python - computer vision. Retrieved March 2, 2021, from https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

[13] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning For Image Recognition. [online] arXiv.org. Available at: [8] Lin, et al. "Network In Network." ArXiv.org, 4 Mar. 2014, arxiv.org/abs/1312.4400.

[14] Krizhevsky, A., Sutskever, I. and Hinton, G., 2017. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84-90.

[15] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dolla´r. Focal loss for dense object detection. In IEEE ICCV, pages 2999–3007, 2017.

[16] Fang, W., Wang, L., & Ren, P. (2020). Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments. IEEE Access, 8, 1935–1944. https://doi.org/10.1109/access.2019.2961959

[17] Majumdar, A. (2018). Graph structured autoencoder. Neural Networks, 106, 271–280. https://doi.org/10.1016/j.neunet.2018.07.016

[18] Zhou, Xinyu, et al. "EAST: An Efficient and Accurate Scene Text Detector." *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, doi:10.1109/cvpr.2017.283.

[19] Seytre, Joel, et al. *TextTubes for Detecting Curved Text in the Wild*. arxiv.org/pdf/1912.08990.pdf.

[20] Epshtein, Boris, et al. "Detecting Text in Natural Scenes with Stroke Width Transform." *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, doi:10.1109/cvpr.2010.5540041.

[21]zhang0jhon."zhang0jhon/AttentionOCR." *GitHub*, github.com/zhang0jhon/AttentionOCR.

[22] Zhang, Jinjin, et al. "A Feasible Framework for Arbitrary-Shaped Scene Text Recognition." arxiv.org/pdf/1912.04561.pdf.

[23] Express, American. "Attention Mechanism In Deep Learning: Attention Model Keras." *Analytics Vidhya*, 28 Nov. 2020, www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/#:~:text=The attention mechanism emerged as,natural language processing (NLP).&text=The encoder LSTM is used,state of the LSTM/RNN.

[24] KHAZRI, Achraf. "Faster RCNN Object Detection." *Medium*, Towards Data Science, 9 Apr. 2019, towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4.

[25] Hochreiter, Sepp, and Jurgen Schmidhuber. "LONG SHORT-TERM MEMORY."

[26] Rosebrock, Adrian. "Detecting Circles in Images Using OpenCV and Hough Circles." *Pyimagesearch*, www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/.

[27] Opencv. "Opencv/Opencv." *GitHub*, github.com/opencv/opencv.

[28] *Log-Polar Transform*, sthoduka.github.io/imreg_fmt/docs/log-polar-transform/.

[29] "Pytesseract." *PyPI*, pypi.org/project/pytesseract/.

[30] Gori, Robley. "PyTesseract: Simple Python Optical Character Recognition." *Stack Abuse*, Stack Abuse, stackabuse.com/pytesseract-simple-python-optical-character-recognition/.

[31] Hussain Mujtaba, et al. "An Introduction to Bag of Words in NLP Using Python: What Is BoW?" *GreatLearning Blog: Free Resources What Matters to Shape Your Career!*, 20 Apr. 2021, www.mygreatlearning.com/blog/bag-of-words/.

[32] Alammar, Jay. "The Illustrated Word2vec." The Illustrated Word2vec – Jay Alammar – Visualizing Machine Learning One Concept at a Time., jalammar.github.io/illustrated-word2vec/.

[33] ImageNet, imagenet.stanford.edu/.

[34] "Overview - ICDAR2019 Robust Reading Challenge on Large-Scale Street View Text with Partial Labeling." *Overview - ICDAR2019 Robust Reading Challenge on Large-Scale Street View Text with Partial Labeling - Robust Reading Competition*, rrc.cvc.uab.es/?ch=16.

[35] Seatgeek. (n.d.). Seatgeek/fuzzywuzzy. Retrieved April 27, 2021, from https://github.com/seatgeek/fuzzywuzzy

[36] Levenshtein distance. (2021, March 09). Retrieved April 27, 2021, from https://en.wikipedia.org/wiki/Levenshtein_distance

[37] Vision AI | Derive Image insights via Ml | Cloud Vision API. (n.d.). Retrieved April 27, 2021, from https://cloud.google.com/vision

[38] Pricing | cloud vision api | google cloud. (n.d.). Retrieved April 27, 2021, from https://cloud.google.com/vision/pricing