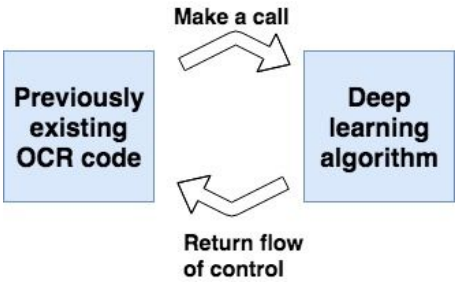
Better Classifiers for OCR

Leah Woodworth, Atanas Golev, Blake Price
CS 499-001
Fall 2018
October 1, 2018

1. High Level Design

Dr. Finkel has previously existing code that executes the OCR program, trains the system, and carries out image evaluation for some specific language. Very little will be changed in that already existing OCR code. The goal is for the existing code to be able to make a call to our program that will then perform the deep learning algorithm, make the changes to the system to increase accuracy, and return flow of control back to the existing code.

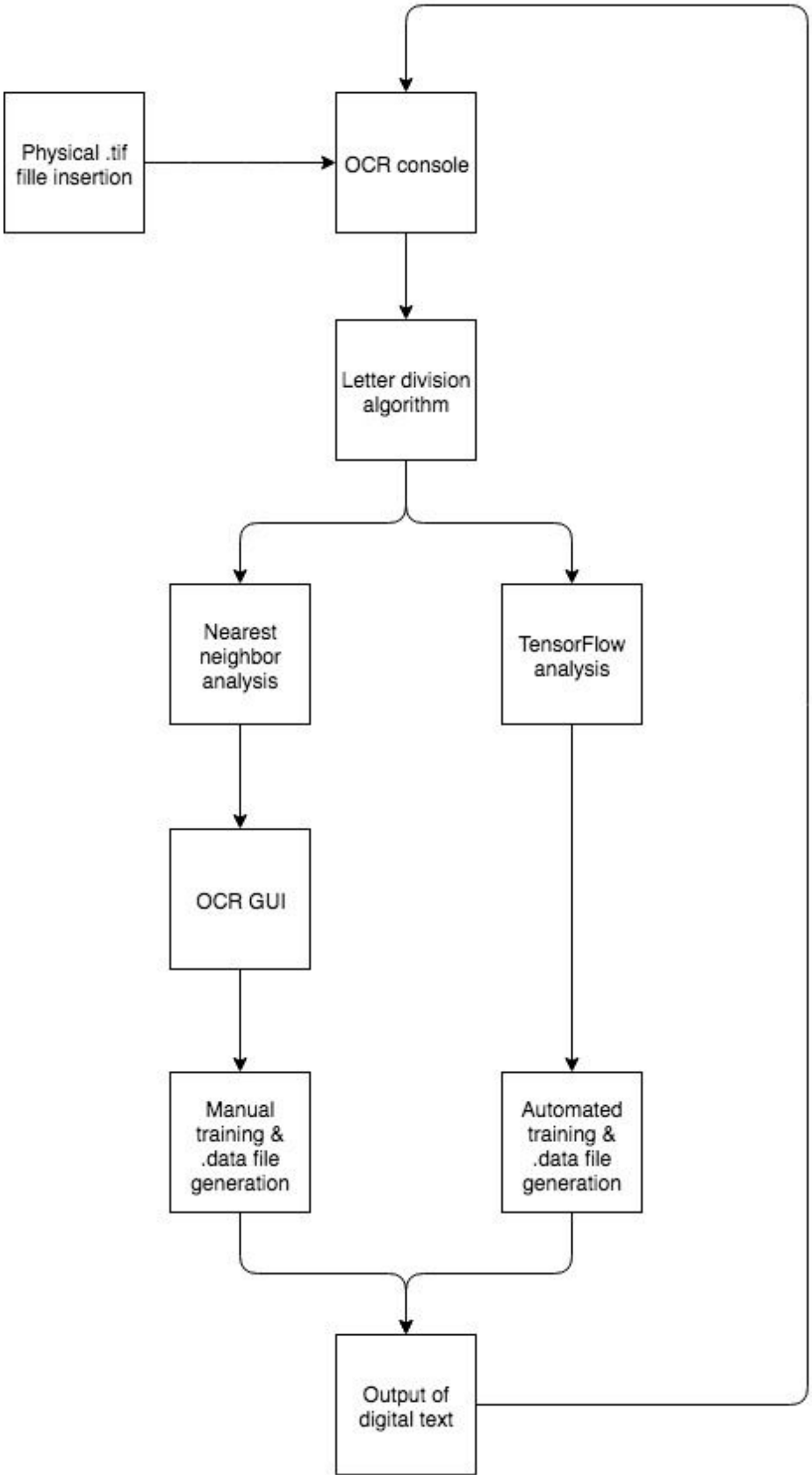


The existing OCR system allows an image file to be processed and stored as digital text. The system divides the text in the image file into letters, or glyphs, and examines them. Using k-d trees, the system is able to determine with a certain degree of accuracy what glyph is being represented. When the system is unsure of the glyph depicted, the user is able to manually use the GUI and tell the system whether the suggested glyph is correct or not. This process makes the system more effective and accurate when determining similar glyphs in the future. Our proposed system will eliminate the need for manual entry. The deep learning algorithm we create will be able to better classify glyphs based on the accuracy of previous glyphs. The more it learns, the more effectively the system will be able to determine the exact glyph being represented.

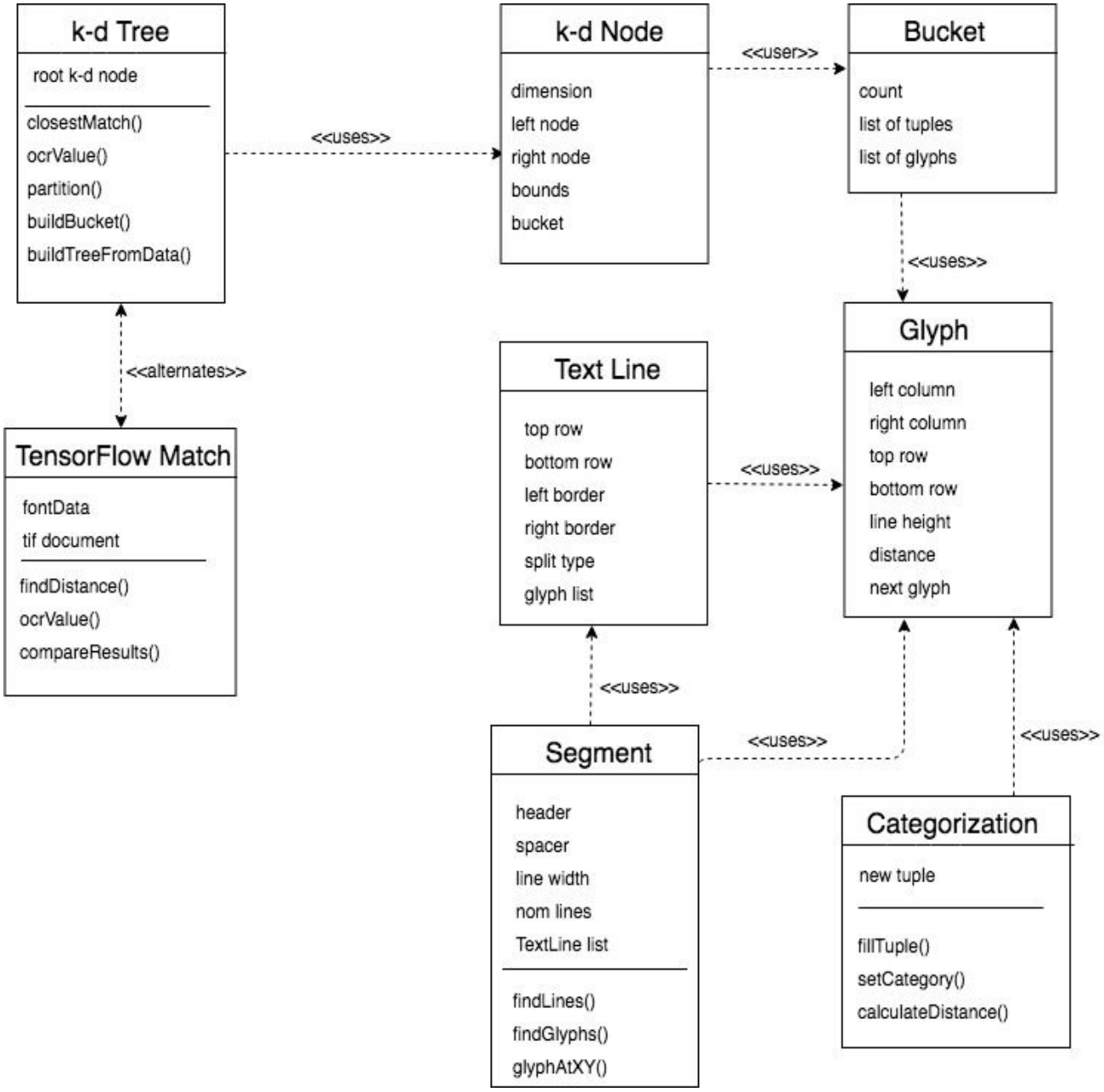
First and foremost, the project should utilize a deep learning algorithm to classify unknown characters through Professor Finkel's existing OCR program. The user should then be given an option to use the deep learning algorithm or the existing nearest neighbor search classification algorithm. This branching is shown in the diagram on the next page. It is also important that the developer be able to compare the two outputs and determine how closely the deep learning algorithm approximated the digital text in comparison with nearest neighbor search.

There are two ways to train the system: manual and automated. Our program should eliminate the need for manual entry of correct glyphs. Undoubtedly, the system will be unsure of the classification of some glyphs, but based on the degree of accuracy, the distance in the k-d tree, and the data it has learned from in past processes, it should be able to make an accurate speculation of which glyph is being represented. The more data that is run through the system, the more effective the OCR will be at classifying glyphs.

This high level architecture design diagram below shows how the user will be walked through both the nearest neighbor search algorithm and the deep learning TensorFlow algorithm. Both algorithms will implement the letter division algorithm, separating each glyph into a 27-D array that holds the data on that particular glyph in the k-d tree. The .data files are generated and hold this information as the system is trained - manually or automatically.


Physical .tif fille insertion
OCR console
Letter division algorithm
Nearest neighbor analysis
TensorFlow analysis
OCR GUI
Manual training & .data file generation
Automated training & .data file generation
Output of digital text

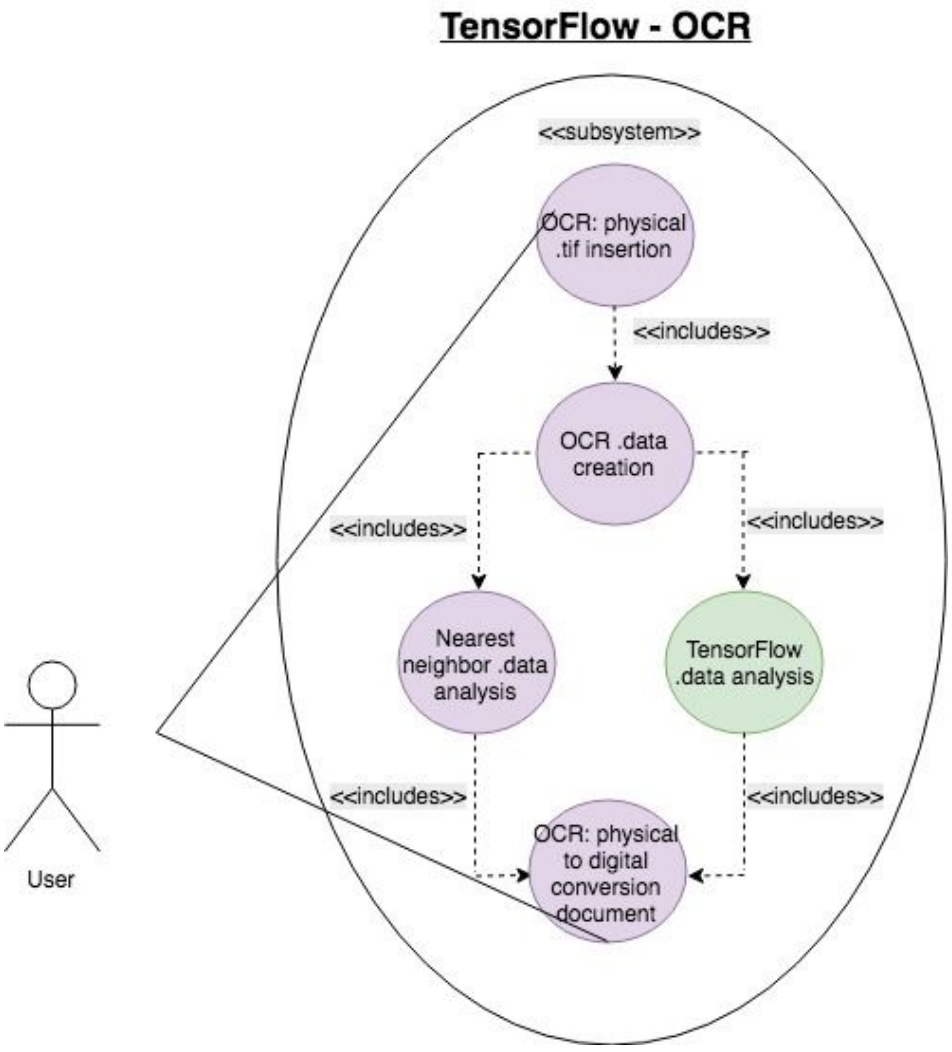
2. Detailed Design



The most basic level of representation within the OCR program is the Glyph struct. The Glyph represents a single character within the text using a 27-dimensional array of floats. Whenever a new character is being identified, glyphs can be grouped together in buckets of possible matches that are within a desired range of the new character. These possible matches are

stored within a 27-dimensional k-d tree. The closestMatch() function is the one that ultimately takes the tree of already recorded characters as an input and outputs the closest match to the new character. Our implementation of TensorFlow will be a alternate version for this version that uses deep learning instead of nearest neighbor search to find the closest match. Thus, our program will have to be an alternate version of the k-d tree class, that works identically to that class within the span of the program. Then, we can simply include an option within the main of whether the program should use the existing nearest neighbor or our deep learning implementation.

The user should be able to choose between using the interactive mode or the batch mode. In the interactive mode, a GUI is displayed and a nearest-neighbor search algorithm (created by Dr. Finkel) is implemented to find a match for the selected glyph. In batch mode, the program is run from the command line and the translation output is sent to a separate file. The batch mode will use the result from a deep learning algorithm, created by our group. When in batch mode, the user should be able to upload a tiff image, select a language, and be able to translate that into digital text. The image below describes what the user will see and experience when using this OCR program.



The main design pattern we've used is the Adapter structural pattern. The Adapter pattern focuses on repurposing a class with a different interface, allowing it to function using different calling methods. More precisely, we are repurposing Dr. Finkel's kd.c class to so that it calculates the correct character using a deep learning algorithm instead of nearest neighbor search. In order to do this, we must make sure that our own class work identically to his: accepting the same types of inputs and producing the same types outputs. Additionally, our Python-based code must run seamlessly with the existing C code, which we are planning on doing by wrapping Python code inside of .c files.

The function within the class in particular that we're working on changing is called the ocrValue() function, which itself calls the closestMatch() function. The function takes 5 inputs: the 27-d tree storing trained font data; a new, unknown character of that same font; a bucket of characters which are possible matches for the unknown character; an int pointer indicating the current closest match to the character within the bucket; a float indicating the closest possible distance between the new character and one of the existing ones for this to be considered the same match.

Our new OCR should be accepting a tuple input, just like the original ocrValue() function. This tuple represents the new character to be matched, and its key in the directory of the characters in the new text. And output should be identical also: returning the closest possible character in the tree to that new tuple. Additionally, if the distance between the tuple and the existing nearest match is more than the user-specified maximum value, OCRFAILS needs to be returned, signifying that the program needs to tell the user that there are no close matches for a pair.

3. Testing

For our project we will be using mostly the tensorflow libraries to produced the desired results when it comes to predicting various texts. Unlike the front-end of the OCR program our program will be found in the backend as a separate function. This means the user will have minimal interaction with our code and rather will request our program as a separate learning algorithm for their text conversion. When it comes to testing what will be required is to provide serviceable information that should be better than the current Nearest-Neighbour search. Some tests that will more than likely come up will be that of correctly deconstructing user input and then relaying the data to us for analysis so that our program. As such some inevitable test cases are as such:

Case No.	Test Case	Pass Conditions	Fail Conditions
1	Create a dataset using the original OCR program, a 'helloworld' .data file will be generated for	Correct data file is generated with the intended percentages being made for each detected character in text.	No data file is created or inaccurate information is created.

	sending to our tensorflow program.		
2	Flow of control.	Data is correctly pulled and deconstructed to prepare for training.	File is incorrectly parsed.
3	Converting original OCR program .data file to readable format for tensorflow.	Parsing is correctly distributed and is properly made for creating a model for analysis and training.	The program incorrectly creates the model for analysis and training.
4a	Data set implementation test.	Dimensions and data is correctly distributed in a way that matches tensorflow's model system.	Dimensions are incorrectly designed (sort of a bounds problem), false 26 dimensional arrays.
4b	Data set model functionality.	Tensorflow variables correctly utilize the data in such a way that training properly functions without error.	Program faults and kills the training process.
5a	Training set accurately distinguishes text examples.	Program is confident in distinguishing characters from one another.	Program fails to accurately distinguish between characters.
5b	Training set needs to not loop indefinitely so we need to limit the epoch (training sessions) correctly.	Program correctly stops when desired so that infinite loops do not occur.	Program loops indefinitely and because of this resources get taken up.
6	Saving training sets, this means we need to test for the ability to load and save previously training sessions on each	Because the user can choose multiple different texts of similar languages the program will need to successfully swap training sets at will	Training sets must be done every run time. Training sets aren't saved between separate uses.

	specific .tif file the user uploads.	correctly.	
7	Nearest Neighbor vs. Tensorflow	Program is able to confidently predict characters more accurately to nearest neighbor.	Program fails to provide more accurate results than the original designed method.
8	Class Structure Independence	Program is organized in such a way that each desired function is independent of outside functions.	Program relies heavily on functions designed in alternative classes making the classes heavily dependent on each other.

We will use each test cases from the figure above once we have the written code to test the its functionality. If the test conditions pass then we will consider the code to be successful and move on to test further functionality. If a test fails then we will have to allocate time to reconsider or redesign such code that at some point it will pass with the desired results utilizing the same original test method.

4. Review

1. Clarification of the high-level design - more in depth analysis.
2. Our sample English tif works with both the interactive and batch modes. However, there needs to be finer tuning in Interactive mode because it's currently having trouble with the width of some characters, like 'm' and 'w', and inaccurately separating them into two different glyphs.
3. One possible issue we may run into would be wrapping our Python code within the existing C program. Fortunately, TensorFlow provides libraries for both Python and C, which we can install on to the virtual machine.
4. Fixed some minor grammatical errors.
5. Added word count.

5. Metrics

System Complexity:

The maximum depth of the inheritance tree is currently 4 level. The kd.c class uses kd_node structs, which use bucket structs, which use Glyph structs. The kd.c class is the one within which our deep learning algorithm will tie in, so we expect for the depth to be identical when our project is completed.

Product Size:

User Stories:

- As a user, I should be able to see a green line over any character which may be incorrectly identified and a red line over any character which is distant from all possible matches. *(complete)*
- As a developer, I should be able to translate a sample document from English in order to ascertain the proper functionality of the current ocrValue function. *(complete)*
- As a user, I should be able to choose between running the deep learning and nearest neighbor classification algorithm. *(in progess)*
- As the product owner or software engineer, I should be able to compare the outputs of the existing nearest neighbor algorithm and the new deep learning algorithm. *(in progress)*

Product Effort:

Atanas: 12 hours
Blake: 12 hours
Leah: 12 hours

Defects:

Though not necessarily a defect, we have yet to figure out how we're planning on connecting the TensorFlow library with the existing C program. We're currently working with the Python TensorFlow library, so a priority is figuring out how to wrap the code within the existing C classes.

6. Web Page and Developer Notebook

We chose to create our developer notebook through github's own documentation section as well as our code development. Developer Notebook is located on our wiki page and our issue/milestones tab presented by github: https://github.com/BlakePrice/tensorflow-ocr/wiki. The ReadMe, Requirement Specs, Project Plan, Word Count, and of course all the code can be found on the master branch.

Word Count as of October 1:
Atanas: 1,998
Blake: 1,909
Leah: 2,180