Better Classifiers for OCR

TEAM
BLEAHLAKETANAS

Advisor: Professor Raphael Finkel
Word Editor: Google Docs
14 September 2018

Atanas Golev: Intro, Overview
Leah Woodworth: Dev and Target Envs, System Model, User Interaction, Functional Recs
Blake Price: Nonfunctional Recs, Feasibility, Conclusion, Appendices

Table of Contents

Introduction

Optical character recognition, or OCR, is the conversion of an image of text into digital text to facilitate editing, searching, and storing the data. The goal of our project is to create a deep-learning algorithm that can be used for OCR.

The most important feature of our program will be this deep-learning algorithm, which we plan to approach using the TensorFlow library. One constraint will be that our program will have to be embedded into Professor Finkel's already existing OCR program, which can already translate a *.tif* file into digital text from practically all languages and fonts. However, this constraint is also an advantage for us, because we'll be able to compare the efficacy of our own algorithm with that of the already existing nearest-neighbor algorithm.

This document is intended to summarize our plans the specifications for this project, to create a structure of guidelines for us to follow, and to clarify our ideas both for ourselves and for anyone interested in the project. It will cover the proposed development and target environments for our project, our system model, plans for user interaction, the project's functional and nonfunctional requirements, and finally, an argument for the feasibility of our plans.

Overview

OCR is important because the conversion of an image of text into digital text can facilitate editing, searching, and storing. For example, Dr. Finkel's OCR program has already been used to convert texts such as Sholem Aleykhem's *Tevye der milkhiker* into digital text, making it easier for academicians and readers in general to search through. We want to improve on this program by creating a new classification algorithm.

Our client is Professor Finkel, and the intended users are himself, us, and anyone else interested in being able to digitize text in a variety of languages.

Professor Finkel's OCR program has already been used to digitize texts and even dictionaries in a number of languages. It can be trained to recognize letters from any alphabet and any font. The current version of the program classifies each letter using a nearest neighbor algorithm. Initially, all the characters are separated from each other into blocks. Then, each character block is separated into 25 different squares. Each of the squares is given a value from 0 to 1 which represents the proportion of black to white space within the space. Each character also has 26th and 27th vectors: these respectively represent the height-to-width ratio of the character and its vertical distance from the center (used to differentiate between apostrophes and commas, for instance).

When a user runs the program, a GUI shows the user the original text and the translation side by side. This makes it possible for the user to check if all the translated characters are correct.

There are also edge cases: The GUI marks a green line above any character whose classification may be questionable, and marks a red line over any character that is completely unrecognizable. The user can then manually input that character into a textbox, using a keyboard of the appropriate language. The program stores these manually input known characters with their 27 dimensions in a k-d tree and uses a nearest neighbor search in order to classify new characters. Our goal is to create an alternate deep-learning algorithm that the program can also use for classification.

The most important feature of our program will be this deep-learning algorithm, which we plan to approach using the TensorFlow library, Google Brain's second generation system that provides both C and Python APIs. We will train our algorithm by running the existing *.data* files of the OCR program with their nearest neighbor search data. When we have constructed an effective predictive algorithm, we will then compare its results against the existing ones to test its quality. Once the new deep learning algorithm has been implemented, the user will be given the option to use either it or to use the existing nearest neighbor algorithm for the translation.

One constraint will be that our program will have to tie into the functionality of the existing OCR program. One way to do this would be to create an alternate *ocrValue* function that uses deep learning in a file similar to the the existing *kd.c* file. This may also involve making slight changes to any other files in which the *ocrValue* variable is used. Either way, this restriction means that our project will have to be done either in C or in Python which can be integrated into the existing C code.

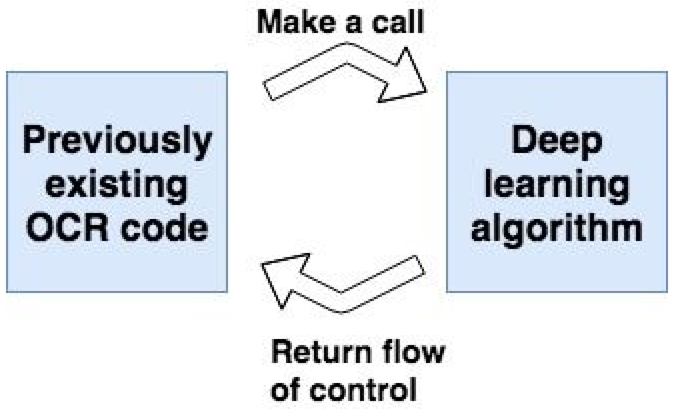
Development and Target Environments

For this project, we will be implementing an algorithm to perform deep learning in the OCR program. In the next few sections, we will examine the software needed for this deep learning algorithm, the previously existing OCR system, and the functionality of our proposed plan.

Deep learning is a subset of machine learning. Instead of the algorithms being "task oriented", the system is designed to "learn" using data, making the task more efficient and eliminating the need for the hard-coding necessary to make the system more effective. To implement our deep learning algorithm, we will be using a C library called TensorFlow, an open source software library used for machine learning. In addition to this library, we will be using a Linux virtual machine to build and run the system. This should allow the already existing OCR code to gracefully make a call to our new code, implementing the deep learning algorithm.

System Model

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.

User Interaction

In our program, the user will be able to choose between running the deep learning algorithm and the already existing nearest neighbor classification algorithm. In the output, a green line over the glyph will indicate that it may be incorrectly identified. A red line over the glyph means it is distant from all possible matches in the k-d tree. The user should also be able to compare the two outputs and determine how closely the deep learning algorithm approximates the digital text in comparison with nearest neighbor search. Ultimately, we want the system to not only be compatible with the existing OCR program, but to be able to effectively identify every glyph, and through repeated processing, become more accurate in its classification.

Functional Requirements

For our system to function properly, there are several requirements that must be implemented. The system must use a deep learning algorithm that allows for more effective future classification of glyphs. The new algorithm must also be integrated well into the existing OCR system, allowing for a smooth transition between old and new functions. If the deep learning algorithm is effective, we should satisfy the requirement that manual entry of correct glyphs is eliminated. These functional requirements are pivotal in our implementation. Nonfunctional requirements are similarly important in the development of our system, and will be outline in the next section.

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. 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. From this information, we will be able to determine how accurately the deep learning algorithm is working. According to our plan, the more the system learns, the more effectively it will be able to determine the exact glyph being represented, to some degree of accuracy.

Another critical requirement is that our program needs to be integrated seamlessly into the existing one, which can be done by either editing the *ocrValue* function within the *kd.c* file, or creating our own alternate *kd.c* file. We may also need to manipulate other files and functions within which the *ocrValue* variable occurs. Because the existing OCR system is on a Linux machine, we will be doing our implementation, testing, and debugging on a Linux virtual machine to ensure that our deep learning algorithm is able to accurately communicate with the existing system.

Our program should also eliminate the need for manual entry of correct glyphs. This process lets the system know if its assumption is correct or incorrect, allowing for more accuracy and efficiency of classifications in the future. 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.

Testing the functionality of our new algorithm can be tested against the existing *.tif* translations of the OCR program. We will be able to immediately recognize whether our own algorithm is comparable in accuracy to the existing one by simply comparing the two translation outputs using the Unix *diff* function. This testing can be done using the existing *Makefile,* which currently contains recipes for translating from a couple dozen image files in a variety of languages; we would simply need to make copies of those recipes that use our own *kd.c* file instead of the original. Over time, as the new system is given more data, the classification of glyphs should become more accurate. As stated, this can be tested by using the *diff* function.

Nonfunctional Requirements

As our program will be a deep learning technique, using TensorFlow libraries, there are some alternative criteria that must be made. In order for out program to function correctly, we must have compatibility, efficiency, memory, and reliability requirements. The language we choose must be compatible with the TensorFlow libraries and our code must be efficient enough for the response time to be generally quicker than that of a linear speed while using the large data sets. Because we are creating a program that utilizes massive data sets, another requirement would be that of memory usage, so we must adjust our program to effectively use the resources wisely. When it comes to reliability of our program, it must also be functional so that it provides accurate results.

The first and most important requirement is choosing the correct language for the TensorFlow libraries. With the original OCR program being independent from our own, we have the choices of utilizing the languages C, C++, or Python. Which language our program is written in doesn't affect transitioning data to the original OCR program as the two won't directly communicate, rather the OCR program will trade *.data* files containing the insight for our program to function. The choice of programming language instead falls down to the ease and compatibility of which each have with the TensorFlow libraries.

The second criteria would be that of reliability. Obviously, for our program to function efficiently, we must first have it reliably produce accurate results. This means that during this phase of our project, our job should be to mitigate the amount of errors so that there is a smooth experience for the end-user. The goal would be to make our implementation robust rather than have a multitude of exception cases. This, in the long run, will make future edits and re-use of code much more feasible as well as giving the end-user a clear and concise experience when running our code.

TensorFlow uses a lot of memory, where the training procedure takes place. When running our program and using the large *.data*, files a physical limit can be reached. So our goal should be to adjust our program so that it uses memory in such a way that does not hinder our performance nor overuses our resources. For instance, if we are calculating a data set that carries only 32-bit integers, then it would be best to not save calculated data as larger data types. Doing so will allow us to not be limited by physical restraints while providing accurate results.

Feasibility

Our job in this project is to research and successfully build a working model of TensorFlow in action. After that, the next job is to make it adaptable to the original OCR program. These two points are the most important steps in having a finished product by the end of our deadline. When it comes to dividing the work, the top priority is to create the TensorFlow deep learning technique and to have an understanding of it at its deepest level. This achievement should allow us to easily transition to the desired results intended with the original OCR program to complete the project.

A simple version of our program will consist of creating a finished working implementation of the TensorFlow libraries. This finished program will take simple *.data* files from English text and be able to train itself to accurately make out the difference for various characters found within the text. This version will not communicate with the original OCR program and instead will only represent a backend that is incompatible with other languages.

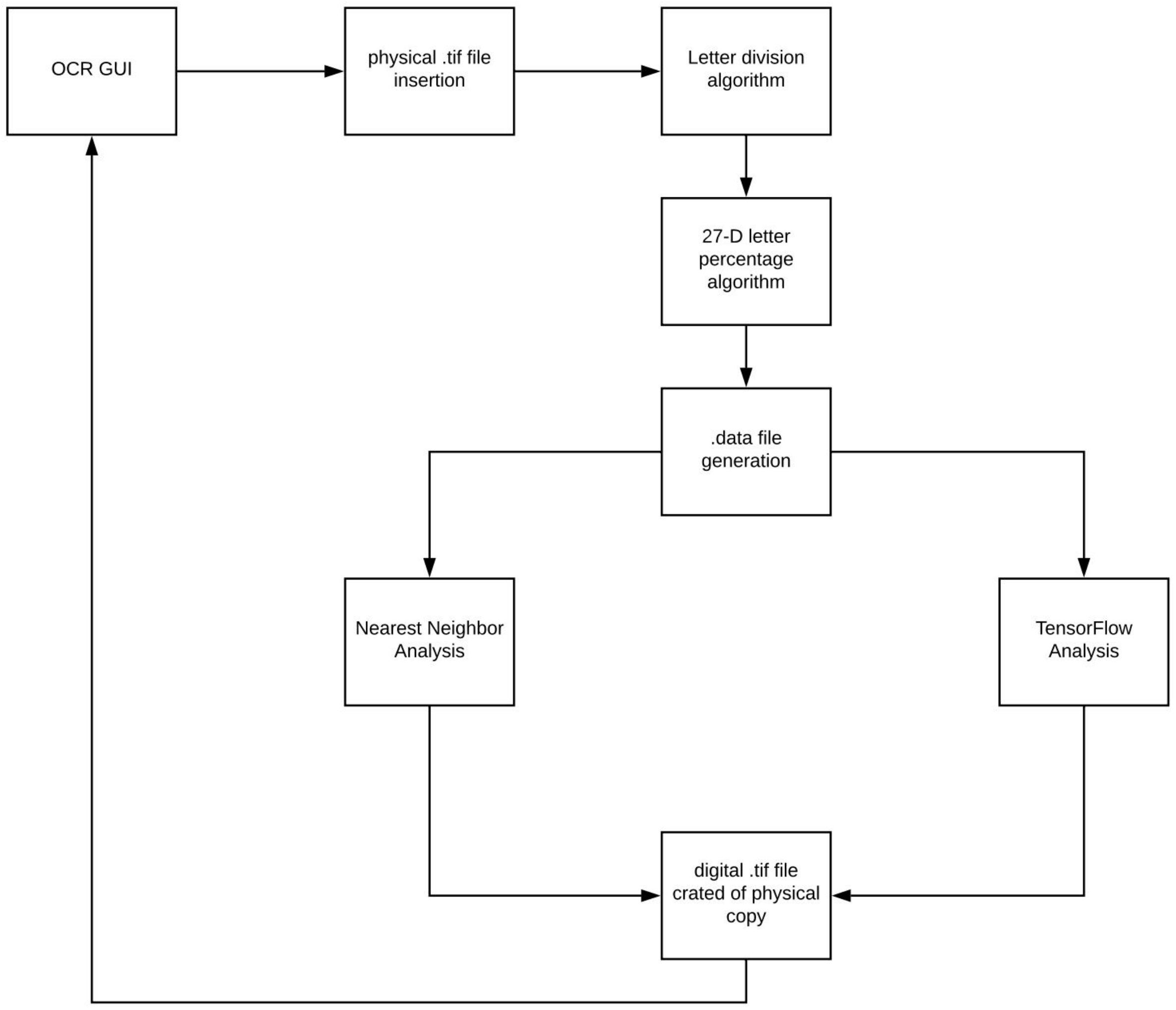
A more complicated version of our program will be able to follow the requirements of the original project plan and be able to accomplish various learning requirements. The program will be able to take various *.data* files from other languages, rather than a single one, and be able to dynamically adjust in a way so that each language is learned separately from one another, but in the same way. In addition, this finished implementation will also be able to directly link with the original OCR program so that a end-user may carry the option of selecting our TensorFlow implementation to learn their various texts (saved as *.tif* files).

Conclusion

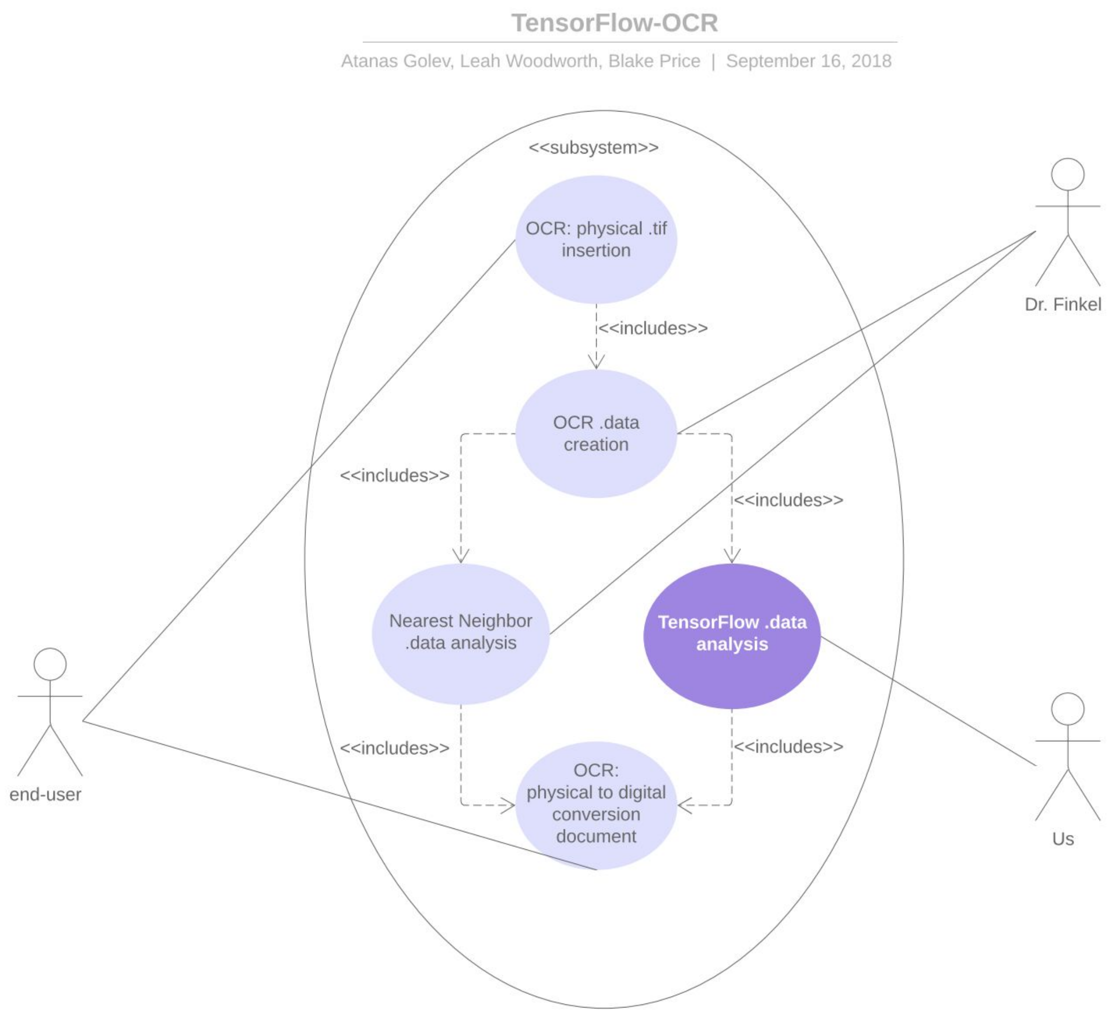
Over the years, digital documentation surpassed the capabilities of physical documentation with the continued development of technology. Optical character recognition, or OCR, is a tool for continuing this process of converting the physical to digital. However the algorithms used in the past aren’t the limit to what this technology can accomplish. Utilizing the TensorFlow libraries, our goal is to successfully form a program that can provide OCR with a more accurate deconstruction of the physical medium as it converts to digital. With the implementation of our program using TensorFlow libraries, we hope to show just how effective deep-learning techniques are in image analysis.

Appendices

System block diagram:


OCR GUI
physical .tif file insertion
Letter division algorithm
27-D letter percentage algorithm
.data file generation
Nearest Neighbor Analysis
TensorFlow Analysis
digital .tif file crated of physical copy

Use-case diagram:


TensorFlow-OCR
Atanas Golev, Leah Woodworth, Blake Price | September 16, 2018
<<subsystem>>
OCR: physical .tif insertion
<<includes>>
OCR .data creation
<<includes>>
<<includes>>
Nearest Neighbor .data analysis
TensorFlow .data analysis
<<includes>>
<<includes>>
OCR: physical to digital conversion document
end-user
Dr. Finkel
Us

Website for Documentation:
https://github.com/BlakePrice/tensorflow-ocr