Oregon State University
College of Engineering

CS Capstone Progress Report

May 6, 2018

CDK Data Stream AI

Prepared for

CDK Global

Chris Smith

Prepared by

Group 65

SigFind

Inhyuk Lee

Jacob Geddings

Juan Mugica

Abstract

This report is the cumulative summary of our groups progress throughout the spring term. We will be discussing the progress made for the term and how we handled them as well as touching upon client interaction and group decisions moving forward. This report will also talk about the three primary components of our project and why we de-constructed our project this way. At the end there will also be a week-by-week breakdown of events.

CONTENTS

1 Introduction 2

2 Project Purposes and Goals 2

3 Recap of Last Term and Progress 2

4 Projected Goals of Spring Term 2

5 ImageMagick 3

6 OCR Tesseract 3

6.1 Signature Locating Program . . . 3

6.2 Winter Term Progress and Problems . . . 4

6.3 Spring Term Progress . . . 4

6.4 Steps Moving Forward . . . 4

7 Convolutional Network 5

8 Weekly Summary of Spring 6

8.1 Week 1 . . . 6

8.1.1 Plans . . . 6

8.1.2 Goals . . . 6

8.1.3 Problems . . . 6

8.2 Week 2 . . . 6

8.2.1 Plans . . . 6

8.2.2 Goals . . . 6

8.2.3 Problems . . . 7

8.3 Week 3 . . . 7

8.3.1 Plans . . . 7

8.3.2 Goals . . . 7

8.3.3 Problems . . . 7

8.4 Week 4 . . . 7

8.4.1 Plans . . . 7

8.4.2 Goals . . . 7

8.4.3 Problems . . . 7

8.5 Week 5 . . . 8

8.5.1 Plans . . . 8

8.5.2 Goals . . . 8

8.5.3 Problems . . . 8

1 Introduction

As for distribution of workload for this progress report: Geddings completed sections Introduction, Project Purposes and Goals, Recap of Last Term, Projected Goals of Spring Term, and worked with Lee on Weekly Summary. Lee completed the OCR Tesseract section and its subsections regarding the merging of OpenCV line detection. Juan completed the section regarding the Convolutional Network.

2 Project Purposes and Goals

Our project, CDK Data Stream AI, is concerned with the construction of a program that can arbitrarily detect if a document has a signature line and if it has been signed. To accomplish this primary task the program must make use of an open source AI platform, function as a black box, and be modular enough that parts can be added or removed from the program. Accompanying this primary goal is a series of stretch goals that, should our primary concern be easy to implement, become our focus for the remainder of the coding section. These stretch goals include license validation, data security, image categorizing, and vehicle image processing. Should these be accomplished, additional interests have been stated in regards to automatic construction of consumer portfolios based on submitted forms.

A major concern for our client is that we do not get caught up on peripheral issues or stretch goals before the core task is completed. Weve been explicitly requested not to worry about container software, cloud operation, or user interface in particular. Another issue of which we must remain mindful is data sensitivity; any documents or forms were permitted to use for testing purposes cannot be made public. They are sanitized for us to legally view, but the forms are not open to the public. Lastly, CDK Global is aware that this project may not be feasible and in lieu of a completed product, they will accept a report indicating why we were unable to complete the project.

3 Recap of Last Term and Progress

All documentation and recordings for the Winter term have been completed and successfully turned in. The primary focus of Winter term was the coding and actual implementation of our project which was also completed in a timely manner. With the closing of Winter term, we only had debugging left to do for our main objective and decided as a group that stretch goals will not be a primary concern moving forward. This means that our code was able to successfully convert PDFs into individual PNG images, pass them onto OCR Tesseract to isolate potential signature lines, and pass these signature lines off towards OpenCV to determine if theyve been sufficiently signed.

As for the individual components of our project theyve each developed in ways we hadnt anticipated when we started. Three distinct modules formed out of this project: image conversion, line detection, and signature verification. These three parts are handled by ImageMagick, OCR Tesseract, and OpenCV respectively. ImageMagick is being used in its command line form while being wrapped in a Bash script. OCR Tesseract and OpenCV both are using C++ for implementation. With all parts working together the end result is an approximate 85 percent accuracy on judgement calls for signature lines. Work has continued in development beyond Winter term however, bug squashing and an expansion of OpenCVs functionality has resulted in a more robust design.

4 Projected Goals of Spring Term

Spring term is the final stretch for our project and this course as a whole. While we didnt manage to develop the various stretch goals we wanted to see come to life we are confident we met the clients original target. Moving forward our priority is preparation for the upcoming Expo as well as searching for any more lingering bugs within our code. As will be mentioned in more detail below, OCR in particular has on more than one occasion surprised us with sudden bugs during testing. We are also going through and adding more comments into our code. As it stands we are nearly complete with the commenting of internal code and will then shift focus to the GitHub Repositories README sections. Lastly, we are making another pass on our installation instructions after learning that various parts of our instructions no longer worked due to version updates on several pieces of our project.

5 ImageMagick

For the file conversion software, we opted into using ImageMagick for our PDF to PNG needs. Ultimately, on its own this portion of the project was straightforward and shares a close bond with the OCR Tesseract portion. As a result, ImageMagick will be discussed briefly on its own for this section and in the following OCR section there will be more discussion on how OCR handles its output. This open-source program was immediately compatible with our school servers and is running on all three of our machines. While this program is exhaustive in all the functions it can carry out we only needed it for its conversion capabilities. We required the ability to convert a given PDF into a upscaled PNG image to have passed off to OCR Tesseract. As it currently stands, the program is both running and accurately producing upscaled images to then pass off to OCR. One problem we had was when attempting to merge ImageMagick with OCR Tesseract more directly. We had considered the use of Magick++ which would provide an in-line C++ variant that we could work with but after performance issues, lack of function calls, and ultimate loss in modularity we opted to keep this portion of the project as a standalone.

6 OCR Tesseract

6.1 Signature Locating Program

To identify whether a given image is a signature or not the software must be able to locate where the sign line is located. Our signature locating program uses the following two techniques to address this problem. One is tesseract optical character recognition, and second is OpenCV line detection technique.

Commonly, signature line consists of two parts which are text and line. To locate the sign line, software must be able to locate both text and line, and software uses Tesseract optical character recognition to locate the text part and OpenCV line detecting technique to locate the line part.

Tesseract optical character recognition (OCR) is an open source software that can convert images to machine readable characters. It is used to search the word signature so software can locate where the possible sign line is. First, software converts a given formal document to characters and searches for the signature phrase within it.

After locating all the signature locations in the document, it finds signature line for corresponding to each text part. Signature locating program uses OpenCV line detection method to locate all horizontal lines in the document, and it finds the closest line from the text part. When the program finds both text and line pieces, it saves the location of the sign line and sends it to signature detecting program (convolutional neural network).

6.2 Winter Term Progress and Problems

During winter term, our group were able to implement whole signature locating program. Our signature locating part was able to search CUSTERMER SIGNATURE from the document and locate it. Also, it was able to find closest horizontal line from the searched text and it could pass the location of the sign line to signature detecting program.

However, there was two major bugs in our program. First bug was in line detecting part, when signature locating program deals with scanned document, sometimes it does not detect the whole line of signature line. Since scanned document has a lot of noise compared to an electronic document, our program had difficulties detecting full length of the line. This was because lines in the scanned document can be uneven resulting in the line finding function detecting the single sign line as multiple separate lines.

The other bug was found in setting sign line boundaries. Signature locating software sets the sign line boundaries based on the closest horizontal line location from searched text. However, on some documents, the program set completely random sign line boundaries. It was not even closed to either searched text or closest horizontal line.

6.3 Spring Term Progress

For this term, our team mainly worked on debugging because our group finished implementing minimum requirement of software features and we found major bugs in our software. On signature locating program, our group focused on fixing the two bugs that were mentioned in the previous section, and we were able to resolve it.

On line detecting part, the major problem was detecting a single line as multiple separated lines on scanned document. To solve this problem, we came up with the idea that combining all the overlapping lines, and this solved the detecting full length of the line. Now, our signature locating program has a significantly reduced chance of cutting signature off.

The other bug was a bit more difficult to solve since we did not even know what was causing the problem. Our group started looking for the exact reason of the random behavior, and we found out that the problem was on tesseract OCR part. As is previously mentioned, signature locating program set the sign line boundaries based on the location of text and line. However, program was not locating exact location of searched word. Instead of locating the searched word, program was locating the whole paragraph that contains the searched word, so sometimes program was setting completely wrong sign line boundaries. To solve this problem, one more loop was added. After segmenting the pages into paragraph, program segment each paragraph into words. Then, it searches for the signature. Program now locate the exact location of the sign line and set correct sign line boundaries.

6.4 Steps Moving Forward

There are a few more problems in signature locating program that needs to be solved and two main problem will be focused on. First problem is searching method, this problem was mentioned on winter term progress report, but it was not fixed yet. When signature is written over the searching word, tesseract OCR would not recognize the characters. Also, the other problem we need to solve is detecting different type of sign line. Our current version only detects the sign line that has the signature line on or above the text part. We need to update our software to detect sign line such as sign line is on right to the text part.

7 Convolutional Network

In order to assess whether a certain region contains a signature or not, our team decided to utilize a convolutional neural network. Convolutional neural networks are a specific type of neural network which take thousands of inputs relating to specific objects, and start learning specific features contained within these. Convolutional neural architectures are specifically designed to cater to object detection, which made them a perfect match for our given task of recognizing signatures.

The first thing our team had to do was familiarize ourselves with what exactly a convolutional neural network was. Convolutional neural networks take an image as an input, and return a percentage estimation of what objects are contained within that image. They do this through a series of convolutional blocks, which are typically comprised of four layers: a convolutional layer, a pooling layer, a ReLu (rectified linear units) layer, and a softmax regression layer. The convolutional layer applies a learned feature filter to the image. This feature filter will generate higher values when applied to an area that is similar to the feature learned while training. The pooling layer looks at smaller regions of around 2-4 pixels and selects the highest value, discarding the others. Neural networks are usually fully connected, but connecting every pixel in an image to every neuron in an another layer, is what many define as computational explosion. The convolutional layer plus the pooling layer allow the network to be much less connected while still being able to achieve satisfactory performance. The ReLu layer makes any negative value into a 0. Many times after training a network will develop negative synapses, also called weights. These negative synapses can be turned into 0s deeming that area of that specific filter irrelevant. By doing this there is no loss in performance but there is a gain in computational productivity. The softmax regression layer applies a function to the value calculated through the other layers, and computes a percentage estimation related to the labels the objects were given during training. Convolutional neural networks like many other neural networks learn through back propagation. The idea is to craft a cost function evaluating how wrong or how far away we are from a correct answer. One this relation is modeled it is evaluated towards a minima and the weights in the network adjusted appropriately depending on whether certain areas activating generated a correct answer or an incorrect answer.

Once we were familiar with the architecture of convolutional neural networks the next step was initializing our own. There are many templates out there for CNNs that cater to a wide ranging complexity of tasks. In order to evaluate signatures efficiently a convolutional neural network does not need to be very complex. The neural architecture we chose consisted of one convolutional layer, one pooling layer, one ReLu layer and 2 softmax regression layers. Originally this convolutional took inputs of type ubyte, since it was modeled to learn to read handwritten digits, and used the MNIST

database to do so. After correctly implementing it and testing it with this subset, the next step was back engineer its input system to be fit our subset. Also this network was designed to be trained and tested without being able to load specific kernels after running the program. In order to fix this, since our network must be able to use trained kernels even after the program is done, we fashioned a kernel loading module and integrated it into the already existing program structure.

In the current version of our program, the network is able to be trained, save the results of the training, and then be ran with a single picture returning an assessment of whether a certain image is a signature or not. Given that we have a rather small subset of signature to train with, it is not yet up the standard of correctness that we would like it to be. It currently functions at 52 percent to 56 percent accuracy with a training set of 300 signatures, and we would like to raise those numbers to 70 percent to 75 percent within this next week. Our final goal is for the network to have over 90 percent accuracy, which will require a subset of 2000 - 2500 signatures.

8 Weekly Summary of Spring

8.1 Week 1

8.1.1 Plans

For our first week back the plan was similar to Winter term, re-establish communication with everyone and start figuring out the ideal meeting times we'd be using moving forward. This included determining our meeting times within our group as well as with our TA and client.

8.1.2 Goals

Our group was able to setup weekly meeting times for ourselves as well as with our TA. Times for inter-group meetings were set to Monday, Wednesday, and Friday from 1400 to 1500. Meeting with TA was set to be Tuesday from 1200 to 1250.

8.1.3 Problems

The only major problem we ran into this week was being unable to setup a meeting time with our client yet. It was also noted that our group meeting times may not work for the full duration of the term like we've done in the past and may need to change them.

8.2 Week 2

8.2.1 Plans

General debugging was the name of the game for this week, we didn't have sufficient testing done to be confident that errors are not present within the code. As a result, we planned to create more test cases and to explore more document styles in an effort to stress the system.

8.2.2 Goals

Test generation was the only major goal this week but we also successfully established a meeting time with our client.

8.2.3 Problems

During this week we encountered the potential for line detection bugs through our more rigorous testing attempts. Initial interpretation of the bug was that the line for the signature was being prematurely cut off. This raised concerns about the possibility that it could randomly cut off a signature line before the signature appears, rendering an otherwise acceptable form as invalid in the eyes of OpenCV.

8.3 Week 3

8.3.1 Plans

A repeat of the previous week in game plan, this focus on debugging would hopefully reveal all bugs within our code before the freeze deadline. As an auxiliary concept, having OpenCV assist in line detection is being explored.

8.3.2 Goals

Additional signatures have been generated as well as development into OpenCV assisting OCR via signature colorization took place this week.

8.3.3 Problems

A potential concern with our testing design was the lack of variety in documents. We needed to test more bizarre layouts that a signature line might be present on. Up until this point we have remained relatively traditional in format.

8.4 Week 4

8.4.1 Plans

Bug was discovered on page segmentation portion of our project. Focus was to iron out this issue and search for additional problems that may exist within the code. We also made plans for final poster revisions as well as ensuring our expo waivers had been completed for submission.

8.4.2 Goals

Poster was submitted, waivers submitted, and the page segmentation bug was resolved this week.

8.4.3 Problems

We need to merge our newest updates and additions into our final version to ensure they all still function together.

8.5 Week 5

8.5.1 Plans

Updates towards main version to include the newest additions was the focus for this week. Progress report became new priority for the remainder of the week.

8.5.2 Goals

Full version was successfully updated with new pieces as well as fixes to existing code.

8.5.3 Problems

One new component is resulting in occasional errors when detecting horizontal non-signature lines. Another bug was discovered during the merge when a signature takes too small a portion of the overall signature line it is observed as empty.