Oregon State University
College of Engineering

CS Capstone Technology Review

November 21, 2017

CDK Data Stream AI

Prepared for

CDK Global

Chris Smith

Prepared by

Group 65

SigFind

Juan Mugica

Abstract

Our team has been assigned to assist in the development of AI for application to CDKs existing Data Streams. These streams deal mainly with the classification of documents as well as pictures and information contained within them. This document outlines the various technologies we will be looking at as candidates to perform the assigned task. Each section will detail a specific technology, three possible candidates, and compare and contrast highlighting positive and negative aspects.

CONTENTS

1 THE ALGORITHM

1.1 Overview

The main goal of this project is to develop an algorithm that can discern whether a document contains signatures, needs to be signed, or does not contain sign-able spaces; based on the analysis of a large number of already existing documents. Given that the task is to utilize an already existing pool of samples to discern key patterns relating to our three criteria while keeping scalability in mind, this problem is best catered to by a neural network algorithm. Neural networks have the unique property of discerning patterns they were never explicitly trained to discern. This property makes them extremely unique when exploring image recognition problems, since these are based on recognizing a wide number of very specific features no one person could ever possibly specifically outline.

1.2 Criteria

When looking at neural nets tasked with image processing, were looking at thousands of pixel related inputs that must be correlated in order to make sense of a certain picture. To land on a particular neural network paradigm we will be looking at the following categories. Efficiency of data propagation: how quickly can the net make sense of a given data set. Learning proficiency: how quickly is the neural network projected to draw relationships between given inputs in comparison to other neural networks. Memory efficiency: what level of memory usage is required to efficiently run this network in comparison to other neural networks.

1.3 FeedForward Neural Networks

Feed forward neural networks are artificial neural networks where connections between units do not form a cycle. In essence, we are talking about neural networks in which data only flows forward, that means it never reassesses data as it passes through, rather it relies on existing architecture to make a prediction based on its current state. These neural nets learn by analyzing the correctness of their estimate against known estimates, which classifies their learning as a form of gradient descent, a first order iterative optimization algorithm for finding the minimum of a function[1]. Feedforward neural networks can provide very fast, resource mindful solutions for simple input data sets, e.g. : guessing a student's overall grade while only being given hours of homework and hours studied. Feedforward neural nets can become flustered when the input data set becomes more complex and thus are to be utilized for scaled down versions of complex problems.[2]

1.4 Convolutional Neural Networks

Convolutional neural networks utilize a variety of different methods in order to discern key patterns about a given data set. The most important method a convolutional neural net utilizes is called, quite fittingly, convolution. Convolving consists of picking out subsets of a certain data set and merging them into one meaningful value. This is done throughout the entirety of the data set ensuring that data subsets have a certain percentage of overlap. These values are then remapped into their own data set creating a feature map representative of the original data set. By repeating this

process, our neural net can start discerning more and more particular features about a data set. A convolutional neural network learns by backpropagation, short for ”backward propagation of errors” [3] , it is uniquely suited to discern patterns within an image, but it must be trained, which in turn means it could be under-trained or over-trained at the point of assessment. Convolutional neural nets are extremely resource heavy and tend to require dedicated GPUs to run efficiently.[4]

1.5 Genetic algorithms

While most neural networks utilize some kind of learning function in order reassess the weights of it?s connections, this can often be a lengthy and resource heavy process. Another approach to learning is the genetic algorithm approach, where we exploit evolutionary concepts in order to create better and smarter nets at each iteration. Genetic neural network algorithms, like all neural network algorithms, start with a neural net with randomized weights that is fully incapable of completing the task that it was given, or rather, a collection of them. A single neural net never learns anything in a genetic algorithm, at its time of birth it was either destined to succeed and breed, or to fail. Among all these neural nets, which are all randomized differently, there will be a subset deemed the most ?fit?. Fitness is the way we assess how well a neural net performs its purpose within a genetic algorithm. The usefulness of the ?breeding? technique is that it is straightforward and simple, and thus creates learning from randomization rather than mathematical assessment. Genetic algorithm neural nets are resource mindful, but can get stuck in certain iterations and/or take a very long (uncertain) amount of time to reach peak performance.[5]

1.6 Discussion

We explored three different neural network paradigms that each highlighted specific components of neural network capabilities. The first was the straightforward feedforward network. A network that learns by optimizing a cost function (cost = —correct answer - wrong answer—), and is only concerned with feeding information through its neurons. It can perform simple tasks very quickly, but its simplistic nature can find itself flustered by complex problem spaces. We explored the convolutional neural network, a network suited for discerning patterns within a data set by iterating recursively over key feature maps, allowing it to discern more complex patterns at each iteration. These neural networks are suited for image detection and are the industry standard in terms of image recognition. While extremely efficient in performing its task, these are resource heavy and rely on powerful GPUs to work correctly. Lastly, we explored the genetic algorithm learning approach, where we were not concerned in teaching any single network to perform our task, but rather only concerned with ?breeding? the networks that are most successful at performing it. This approach is resource mindful, but can get stuck in certain iterations and never learn to perform our task satisfactorily or take an unreasonable amount of time to do so.

1.7 Conclusion

While the feedforward approach is efficient, it does not seem to be powerful enough to tackle the problem space that we will be dealing with. The convolutional network is basically the obvious choice but it is fair to take genetic mutation

and learning into consideration if our simulation encounters points where backpropagation performs underwhelmingly.

1.8 References

[1] ”Wikipedia Gradient Descent” Internet: https://en.wikipedia.org/wiki/Gradient_descent, [Nov.15,2017].

[2] ”CS Stanford, Neural Networks” Internet: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Architecture/feedforward.html, [Nov.15,2017].

[3] ”Brilliant : Backpropagation” Internet: https://brilliant.org/wiki/backpropagation/, [Nov.15,2017].

[4] ”Adit Deshpande: A Begginer’s Guide to Understanding Convolutional Neural Networks” Internet: https://adeshpande3.github.i Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/, [Nov.15,2017].

[5] ”Matt Harvey: Let’s evolve a neural network with a genetic algorithm” Internet: https://blog.coast.ai/lets-evolve-a-neural-network-with-a-genetic-algorithm-code-included-8809bece164, [Nov.15,2017].

2 Container Software

2.1 Overview

One of the most important features outlined throughout our project is the ability for it to be used from any machine. The project is to act as a standalone application that is easy to launch and operate regardless of the machine we are utilizing. In order to do so it must be deployed utilizing containerizing software, which addresses this problem at its core.

2.2 Criteria

One of the most important features outlined throughout our project is the ability for it to be used from any machine. The project is to act as a standalone application that is easy to launch and operate regardless of the machine we?re utilizing. In order to do so it must be deployed utilizing containerizing software, which addresses this problem at its core.

2.3 LXC

LXC is specifically designed for Linux users and aims to emulate the Linux standard installation without the need for a separate kernel. LCX?s API supports a number of languages for deployment including Python 3, Lua, ruby, and Go. LXC does not support being deployed in c++ which could create extra constraints if the native code of our application runs on c++. LXC support, being a Linux based application, depends on stable Linux releases and the efforts of their creators to cater to it with specific distributions. Documentation for its API is existent but often difficult to find, and almost always requiring disambiguation for specific tasks. This is mainly due to problems being Linux distribution specific.

2.4 LXD

LXD is also designed for Linux distributions and offers an experience similar to virtual machines but utilizes Linux containers instead. LXD is an image based containerizing software available for a wide range of Linux distributions that supports all the languages LXC does. LXD main pros are it is intuitive and simple API, its security driven design, and image based deployments. Unlike LXC whom relies on singular developers to tend to it, LXD is developed and distributed by Canonical LTD. Canonical LTD offers support and documentation for LXD which is lacking for LXC. LXD builds on top of LXC and acts as a friendlier version of the LXC framework.

2.5 Docker

Docker is the current industry preferred choice when it comes to container software. Docker is faster than other container options, is extremely well documented and has its own public repositories to obtain the latest installations. The Docker Hub hosts thousands of container images as well as support for the download of ready to use apps, something that tends to be hard, if not impossible to find on other repository websites such as GitHub. Even though Docker is advertised as supporting Windows and Mac OSX, it utilizes virtual machines on these which in essence defeats the point of the container approach. Docker is written in the Go programming language, something which the team must familiarize itself before utilizing it [1].

2.6 Discussion

Each of our three choices contains its own set of pros and setbacks. LXD pertains to more bare-bones approach where we utilize the basics of the containerizing paradigm at our own complete discretion. This could save memory, but due to its poor documentation and support, will need extra time to get accustomed to in order for the team to become proficient in its usage. LXD is well documented and image based, both of which will help save time and effort when containerizing the application. Even though there is extensive support for LXD, it builds LXC, and it could potentially fall into the same undocumented setbacks and LXC. Docker is widely regarded as the best container software across the industry and was mentioned by our client as a choice when the project was first discussed. It is well documented and supported, but boasts cross platform implementations even though it utilizes virtual machines to do so, which can be done to utilize LXC and LXD. Docker also will need that the team familiarize itself with the Go programming language [2].

2.7 Conclusion

While LXC could be a very powerful resource efficient approach, utilizing LXD will provide most if not all LXC features in more time sensible manner. While LXD is well documented and industry standard ready, Docker seems to be widely regarded as superior due to the aforementioned features. While not having utilized either they might seem like equals, its popularity across many developer opinions warrants that Docker be our main choice, with LXD being only pertinent if certain required features prove untimely, or inefficient, on Docker [3].

2.8 References

[1] ”Linux Containers, LXC” Internet: https://linuxcontainers.org/lxc/introduction/, [Nov. 18, 2017]

[2] ”Linux Containers, LXD” Internet: https://linuxcontainers.org/lxd/introduction/, [Nov. 19, 2017]

[3] ”3 Pros and 3 cons of Working with Docker Containers” Internet: https://sweetcode.io/3-pros-3-cons-working-docker-containers/, [Nov. 20, 2017]

3 Development Platform

3.1 Overview

In order for a neural network algorithm to run it must run continuously and at a high end processing power. It takes many iterations of a neural network before it is optimized to perform the task that it is given. For this specific task we will explore three different continuous run-time environment options that can host our application, our home pcs, the OSU flip server, other resources / paid services.

3.2 Criteria

The chosen technology must allow a constant run-time environment of at least 24 hours at a time. This number is rather arbitrary but optimal so that instances can be checked in the mornings and left unsupervised throughout the day. It must be able to run Linux and must have enough computing power to efficiently run our algorithm uninterrupted.

3.3 Home-PC

Developing the algorithm on our home computers allows us ultimate accessibility especially when executing certain permissions necessary for the proper functionality of our various frameworks. It also allows unrestricted access to installing any third party software we deem appropriate to test and maintain our application. Utilizing our home pcs is free of charge, but home pcs might not have the capability to efficiently run our algorithm, and will require constant attendance to ensure the continuous run-time environment necessary to achieve optimal learning.

3.4 OSU flip server

The OSU flip server guarantees a constant run-time environment monitored by OSU computing facilities staff. OSUs computing resources are vast and allow our algorithm to run efficiently and uninterrupted with minor restrictions on memory usage. They are free of charge but are restricting when it comes to installing permissions.

3.5 Other resources / paid services

Our last option would be to find a paid service that allows us to code free of restrictions and in a continuous run-time environment. These services would combine the pros of both OSU?s flip server and our home computers, allowing us full control of our environment as well as a supervised server unlikely to go down at any critical point. Utilizing one of these services would come at a cost that would have to be paid by the members of this group, putting an economic constraint on the project.

3.6 Discussion

Each one of these options is fully capable and viable in terms of our existing criteria. The choice will likely come down to a number of factors for each of our options. For home pcs we must figure out if each member's option has the computing power to run our algorithm. This basically comes down to a powerful gaming oriented GPU. For paid services it will be decided on whether the group can accommodate the expense of renting / utilizing a pay as you go server. If neither of these options is viable, the OSU flip server must be utilized and its restrictions addressed to ensure the proper functionality of our various frameworks.

3.7 Conclusion

Given that the OSU flip server has both the computing capability, as well as constant availability, it is the most likely candidate for the development of our application. In the unlikely event that a critical function cannot be executed due to a lack of permissions non addressable as students, one of the other choices must be picked and our strategy reassessed.