Proceedings of the Canadian Undergraduate Conference on Artificial Intelligence

March 6 – March 8, 2020
Toronto, Ontario

Copyright Note

The papers presented in this document make up the proceedings of the event described on the title page and foreword, and reflect the opinions of the authors.

Editorial production by Jeremy Kulchyk.

Cover art production by Ricky Zhang.

Review Process for Best Paper Awards

Of the 24 paper submissions, 10 papers were considered for Best Paper Award, as selected by the executive team of QMIND. These 10 papers were sent out to three professors and two graduate research students, all in the field of AI, to rank their top 5 papers. The papers were ranked on the basis of quality of work, novelty of the solution, and quality of the paper.

Thank you to Ali Etemad, Amy Wu, Tamas Ungi, Amber Simpson, Karen Batch, and Kaelan Lupton for your time and contributions to the paper ranking process for Best Paper Award.

The three highest ranked papers, in no particular order, that recieved Best Paper Award are:

Happy Transformer
Helping Hand: The ASL Translator
Robotic Traversal and Pothole Avoidance

Congratulations to these teams for their impressive work. These papers are also distinguished in the proceedings.

Table of Contents

Table of Contents

Sponsors Acknowledgement

First and foremost, we would like to take a moment to thank all of our sponsors. Without you, this event truly would not have been possible. You provided us with speakers, panelists, workshop coordinators, and booth representatives. It was an honour to showcase your support and have you at the conference to share your insights and experiences. In turn, you helped delivered an unparalleled experience to our 215 delegates.

We'd like to give a special thank you to our Title Sponsors: The Smith School of Business Master of Management in Artificial Intelligence (Smith MMAI) of and The Faculty of Engineering and Applied Science (FEAS), both of Queen's University. We are forever grateful for your support. Second, we'd like to thank our Gold Sponsors: The Centre for Advanced Computing (CAC), Distributed Compute Labs (DCL), The Canadian Institute for Advanced Research (CIFAR), Suncor, Pelmorex, Round 13, TD Canada Trust, The Dunin-Deshpande Queen's Innovation Centre (DDQIC), and RBC Royal Bank. Lastly, thank you to our Silver Sponsors: Canvass Analytics, KPMG, Sagard Holdings, Klick Health, Vector Institute, Schulich School of Business Master of Management in Artificial Intelligence, and Oliver Wyman.

Thank you all and we hope to see you again next year!

Conference Organization

Conference Chairs

Rachel Bouwer, Queen's University, Canada
Maddy Liblong, Queen's University, Canada

Managing Directors of QMIND

Levi Stringer, Queen's University, Canada
Maxwell Berkowitz, Queen's University, Canada
Hatem Aldawaghreh, Queen's University, Canada

Outreach Team

Mitch Mathieu, Queen's University, Canada
Cameron Mackinnon, Queen's University, Canada

Delegate Manager

Ani Verma, Queen's University, Canada

Logistic Coordinator

Denise Micu, Queen's University, Canada

Marketing Coordinators

Samantha Dunn, Queen's University, Canada
Cathy Yan, Queen's University, Canada

Directors of Design & Editors

Stefano Roque, Queen's University, Canada
Shawn Carere, Queen's University, Canada
Willem Atack, Queen's University, Canada
Sahil Grewal, Queen's University, Canada

Director of Research and Chief Editor

Jeremy Kulchyk, Queen's University, Canada

Foreword – QMIND

This collection of publications is the product of a year's work from nearly 200 undergraduate and graduate students at QMIND - Queen's University's AI hub. QMIND's mission is to actively challenge the status quo around what students are capable of achieving in the field of AI. QMIND's members are makers, innovators, and disruptors, who are already solving industry-level problems at the undergraduate level. QMIND's goal is to continue the growth of its community and provide a space for members to explore the world of AI. QMIND works directly with academia, industry, media, and other student organizations across Canada.

QMIND's many external partnerships span the gamut of organizations looking to leverage AI. Research labs at Queen's look to QMIND to help tackle numerous areas of study, and members ideate new areas of research to apply their skills. On an industry front, QMIND has established itself as a reputable consulting organization, having delivered high quality solutions to paying clients in industries across banking, insurance, transportation, retail, and marketing. QMIND is also committed to contributing to our surrounding communities, and frequently engages in pro-bono projects such as med-tech development for local hospitals. The product of these projects is an AI talent pipeline composed of industry-trained students all centralized under the QMIND name. Every year, companies choose QMIND as their source for new AI talent through office tours and on-campus networking events. Our external engagement extends to AI societies at universities across Canada to foster a national network of students who share QMIND's vision.

Within QMIND, members have the opportunity to develop many skillsets through a suite of educational initiatives that greatly compliment their in-class learning. Members who are new to the space of AI are provided with an educational onboarding that is tailored to different skill levels and taught by experienced members. Throughout the year, members have access to an online course developed by QMIND to enrich their self-directed learning. QMIND has also taken a community-based learning approach by connecting members with others to help overcome technical challenges. In an effort to remove roadblocks to AI research at an undergraduate level, QMIND has invested in robots and GPUs to facilitate our autonomous driving stream, as well as partnering with Queen's professors and graduate students to provide mentorship. Through the Professional Development program, members are provided with career mentorship, LinkedIn and resume workshops, industry networking events, and technical communication workshops. Additionally, the Women in QMIND (WIQ) consists of a mentorship program and supportive sub-communities, encouraging diversity in the AI space.

Foreword - CUCAI

Cumulatively, these initiatives bolster the portfolios of QMIND members and enable leadership experiences that help define their careers. These accomplishments are reflected in this collection of publications.

From March 6th-8th 2020, the second annual Canadian Undergraduate Conference on Artificial Intelligence (CUCAI) was held at The Westin Harbour Castle in downtown Toronto. Following a highly successful debut in 2019, the event featured 300+ delegates and industry representatives hailing from across the country. The weekend consisted of top-notch speakers, hands-on workshops, thought-provoking panel discussions and professional-caliber student projects all geared towards the advancement, legislation, and education of AI. Students were able to learn more about industry, companies had the opportunity to recruit the next wave of tech talent, and the AI community took one more step towards making the world a better place.

Over the course of the weekend, delegates had the opportunity to hear from Pavel Abdur-Rahman, a Partner and Head of Data & Trusted AI at IBM, Shary Mudassir, Co-Head of Global Equities Execution at RBC Capital Markets, and Sheila McIlraith, Canada CIFAR AI Chair and Professor of Computer Science at University of Toronto. To culminate the weekend, Geoffery Hinton delivered the keynote speech. Hinton is viewed as a leading figure in the deep learning community and is referred to by some as the "Godfather of Deep Learning". In addition to hearing from leading experts, CUCAI's Industry Showcase allowed students to interact directly with companies. This weekend was made possible by many generous sponsors, including our Title Sponsors, Queen's Faculty of Applied Science and Queen's Smith School of Business. The full list of sponsors can be found on the Sponsors Acknowledgement page.

CUCAI's Student Showcase was a booth-style event in which QMIND teams presented the work they had accomplished over their year-long projects. QMIND members shared their work with delegates and had their designs reviewed by industry experts in a professional setting. The calibre of these projects is demonstrated by the breadth and depth of the articles found in these proceedings. QMIND and CUCAI encourage interested parties to reach out and participate in future iterations of this innovative work.

Analytics

City of Kingston Heating

Erin Peterson[1], Val Kobilaski[2], Christian Martyn[3],

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada

1 email: erin.peterson@queensu.ca
2 e-mail: val.kobilaski @queensu.ca
3 email: 18cam5@queensu.ca

Abstract: *The problem we addressed stemmed from an environmental passion, an interest in current smart thermostats, and the lack of commercial availability for these products. The problem was a tackled through the process of data collection, data manipulation, model creation, and testing. The results included a prediction on a scale of zero to one of what the heating should be set at, at any given hour, in order to preemptively heat the room based on the upcoming external temperature. In the future, it is hoped that the model be integrated with the City of Kingston's Facility Maintenance System.*

---

1. INTRODUCTION

1.1 Motivation

Within the current ecological climate, reducing energy costs is an absolute must in order to achieve a regressing carbon emission. One of the greatest factors that contributes to energy consumption is maintaining a building's temperature, in fact in Europe, it's estimated that approximately 50% of energy produced goes towards heating buildings [1]. Because of this, building designers have begun investing more towards thermo insulation in new contemporary buildings.

The problem persists in buildings that were constructed prior to the understanding that thermal insulation. This is elevated by the fact that buildings over 40 years old with cultural significance can be deemed a cultural historical site and can no longer be outfitted with modern insolation [2]. Additionally, for many buildings

constructed in the 20th century it's simply not economical to outfit the building with modern insolation solutions. Leaving the building designers and the City of Kingston in a dilemma as the City of Kingston is one of the earliest sites of European settlement in Canada and contains over 135 historical sites and buildings.

One possible solution is to make standard heating control more efficient and economical. Many of the current heating control solutions don't consider current weather conditions. External temperature as well as high wind speeds have a significant impact on a building's internal temperature, and this is only further compounded in buildings with poor insulation already [3]. Designing a heating solution which considers outside temperatures would help greatly in reducing heating costs within the City of Kingston.

1.2 Related Works

The smart home revolution has prompted a recent development in smart thermostats. Companies like Google, EcoBee and Wiser Air have all come up with solutions that claim to be more energy efficient than standard thermostats. Google's Nest and Eco Bee's smart thermostats however are tailored to adjusting temperature based on your personal activity. They connect with peripheral smart devices in your

house and moderate the temperature based on your personal activity, for instance it may communicate with your smartphone in order to lower the temperature when you are away [4]. Wiser Air's thermostat does read temperature forecasts and adjust the thermostat directly, but this is a static solution that doesn't necessarily apply to buildings with unique heating properties. In order to accommodate these unique buildings, we will employ machine learning in order to create unique and dynamic heating solutions which can read future temperature forecasts and moderate the building's temperature in a way that's unique and tailored to the building it's installed in.

1.3 Problem Definition

The outstanding problem, with modern day thermostat solutions is that they aren't designed to accommodate larger buildings with outdated thermal isolation. This can invariably lead to problems when employing these smart devices as they fail to properly determine the effect of weather on the building's internal temperature. As well, most of these buildings use reactive thermostats that results in temperatures that are often fluctuating to uncomfortable temperatures while being inefficient in energy consumption. This means the temperature is able to drop significantly below the building's standards and the thermostat would react by turning the heat to full blast to compensate, which is not only expensive, but it would take a long time.

Our proposed solution is to create a neural network system that learns based on historical internal temperature data as well as historical weather data and compare them to the time and date to determine how weather affects the building in question at specific times. The system once trained will take as input, the future 24-hour forecast for the region and produce an estimated activity for the thermostat that minimizes heating costs, while still maintaining a steady internal temperature.

2. METHODOLOGY

2.1 Data Collection

It was necessary to collect data regarding both the external temperatures and the internal temperatures of the system is to be implemented in, the Grand Theatre. The theatre data was provided by the City of Kingston, and consisted of time, date, and temperature. The source chosen was the Government of Canada's historical weather data, and the weather station closest to the Grand Theatre was utilized. We then web scraped the data from this site.

2.2 Data Manipulation

Both sets of data were reduced to only the date, time, and temperature at that time. They were then put into arrays containing the current internal temperature, and the following 24 external temperatures.

	Year	Month	Day	Time	Temp
2	2019	11	4	16:00	16.9
3	2019	11	4	17:00	16.9
4	2019	11	4	18:00	16.8
5	2019	11	4	19:00	16.8
6	2019	11	4	20:00	16.9
7	2019	11	4	21:00	16.9
8	2019	11	4	22:00	16.9
9	2019	11	4	23:00	16.9
10	2019	11	5	00:00	16.9
11	2019	11	5	1:00	17.3
12	2019	11	5	2:00	17.0
13	2019	11	5	3:00	16.7
14	2019	11	5	4:00	16.4
15	2019	11	5	5:00	16.3
16	2019	11	5	6:00	16.1
17	2019	11	5	7:00	15.9
18	2019	11	5	8:00	15.8

Figure 1 - Visual of the manipulated theatre data.

2.3 Model

A model was created to provide weights to each hour regarding how much the external temperature would affect the room at that hour. It also takes into account the time of day and the day of the week to account for common times people are in the building or doors are open etc. For instance, the model would be able to recognize that from 9 am to 5 pm on Monday to Friday the temperature output would vary from normal and it would adapt to compensate for the changes. In simple terms, the neural network would be able to determine how the difference in thermal bodies at different times of the day would affect the required temperature output. Using the information given, the model would predict the ideal heat output as opposed to reacting to the internal temperature of the building.

2.4 Testing

Testing was done on the model with variable number of epochs and learning rates, and the results were compared in order to find the numbers that would produce the lowest error rate. However, the model is constantly receiving feedback on its performance and

making adjustments to improve and achieve a temperature as close to the desired temperature as possible.

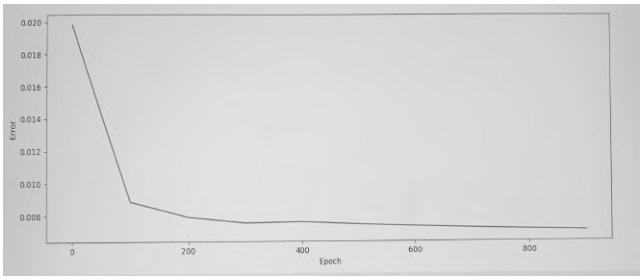


Figure 2 - Error Rate vs. Epoch when the learning rate is 2 and the number of epochs is 5000.

2.5 Reasoning

Multiple approaches to solve this problem were discussed, and this solution was chosen because it allowed for the evaluation of the given data in order to provide a number on a scale of zero to one, zero being off and one being full heat, that would indicate the thermostats necessary setting. This result also allowed for the ability to add on to the solution, such as including utility data as an input.

3. RESULTS AND DISCUSSION

Results from the project indicated the optimal heating power on a scale of zero to one at any given time in the day in order to keep the temperature in the room the requested 17 degrees Celsius.

Our methods did work as expected with the model was trained on the data we had, but it would have been better if it could continue training on real time data. If it was given the opportunity to do this the model would have continued to learn and the results would have become more accurate.

As well, a model that would be used to predict the utilities cost for the month was also in progress. Therefore, if this model was used in the system, this neural network could look at the average heat output of the month and compare that to utilities costs and heat output of previous months to make an accurate prediction of the utilities costs of that month. The model performed well, but we believe that it could have been expanded on in the future.

4. CONCLUSIONS AND FUTURE WORK

Overall, the attempted solution of a commercial predictive heating software was successful. If we are to continue to work on this project, the most important aspect to expand on would be the integration with the City of Kingston's Facilities Management System. This is also a challenge that remains, because there is a difficulty of integrating to an existing system due to security measures that are in place. The City of Kingston has a contract in place for its Facilities Management System which makes incorporating our solution into their system a difficult task. We are still working with the City of Kingston.

Another aspect that could be expanded on is implementing the City of Kingston's utility data as an input. This would enhance the software, giving it the ability to reduce the City of Kingston's utility consumption, and therefore reducing the cost to heat and cool the buildings.

REFERENCES

[1] Dylewski, Robert, and Janusz Adamczyk. "Economic and ecological indicators for thermal insulating building investments." Energy and Buildings 54 (2012): 88-95.

[2] "Main Criteria And Guidelines". Pc.Gc.Ca, 2020, https://www.pc.gc.ca/en/culture/clmhc-hsmbc/ncp-pcn/evaluation. Accessed 25 Mar 2020.

[3] Wojdyga, Krzysztof. "An influence of weather conditions on heat demand in district heating systems." Energy and Buildings 40.11 (2008): 2009-2014.

[4] Saha, Ankita, Murat Kuzlu, and Manisa Pipattanasomporn. "Demonstration of a home energy management system with smart thermostat control." 2013 IEEE PES Innovative Smart Grid Technologies Conference (ISGT). IEEE, 2013.

Hockey Rebound Prediction

Andrew Simonds[1], Mike McColm[2], Kyle Air[3], Jake Zeldin[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada

1 e-mail: andrew.simonds@queensu.ca
2 e-mail: mike.mccolm28@gmail.com
3 e-mail: 16kaa11@queensu.ca
4 e-mail: 17jjz@queensu.ca

Abstract: *With goalies being so skilled in the NHL, one of the best ways to score a goal comes from a second chance on a rebound into a high danger area of the ice. Our team has developed a tool to accurately predict where a rebound will be directed after a shot. The tool will be used to train players to shoot in certain locations to produce the highest scoring chance following the shot. The team trained a deep neural network, with help from TensorFlow libraries, using sample data consisting of shot location on ice, and shot location on net to then predict the rebound angle in 30-degree bins surrounding the net. The final product consists of a GUI which allows a user to choose where the puck is being shot from and the location on net and output a spray chart of probabilities of where the resulting rebound will be.*

---

1. INTRODUCTION

1.1 Motivation

While certain sports have been infatuated with data analytics, the hockey analytics awakening is still in its infancy [1]. Companies are beginning to emerge with the ability to extract vast amounts of live data from a hockey game, such as player and puck location on the ice. Although National Hockey League (NHL) franchises have yet to adopt data analytics as a major form of scouting and coaching, it is evident that these practices will soon be implemented.

In 2016, NHL coach John Chayka became the youngest general manager in the NHL's history, and he also happens to be the founder of Stathletes, a hockey analytics company [2]. He is one of the main driving forces in proving the validity of data analytics in professional hockey.

One of the specific problems in the game today is the lack of knowledge regarding rebounds, and tendencies associated with certain goalies. This is what our team within QMIND aims to explore further.

1.2 Related Works

Similar projects have been completed within the realm of hockey analytics, such as the investigation into how the pace of play in a game affects how well a team will perform. The research paper, "Playing Fast Not Loose: Evaluating team-level pace of play in ice hockey using spatio-temporal possession data", dives into this problem by assessing various pace metrics and analyses of players and teams. Their findings suggest that team-level pace is beneficial, but only to a certain point [3]. Some of the main issues within their project involved the randomness of the sport, which will prove to be a similar challenge for our project. The group working on this project plans to expand their findings into similarly structured sports such as soccer, basketball, and rugby.

1.3 Problem Definition

In a similar manner to the team mentioned in Section 1.2, our team plans to tackle a difficult to grasp aspect of hockey, in the form of rebound prediction. Through our general passion for the sport and research before starting the project, we came to a consensus that a fantastic problem to focus on would be creating a way

to take all of the events leading to a rebound event, and format insights from these events in a useful fashion to be used by players and coaches. Our main problem we decided on was to take in a shot location on the ice, and shot position on the goalie, and output accurate predictions for where the rebound angle would be. We believe that with this information, players will be able to exploit certain goalie tendencies, or goalies will be able to acknowledge and work to fix some of their weaknesses.

2. METHODOLOGY

2.1 Preprocessing

As one of the main issues seen in similar hockey analytics projects was the randomness of the game, we tried to reduce this element by focusing on clean shots on goal to analyze rebounds for. This took out shots on goal involving deflections or other unique shots. The randomness still remains in the project, but due is further touched upon in the discussion of our project.

Through the partnership with Iceberg Sports Analytics, we were provided with a very large dataset of shot attempts through a series of American Hockey League (AHL) games, specifically focused on the Bridgeport Sound Tigers, the AHL affiliate of the New York Islanders. Each of these data points included a number of features, including shooter and goaltender coordinates, shot type, player information, and a number of other characteristics. Our first step was to filter the data down to a more manageable size. With such a large dataset, we had the luxury of being able to remove datapoints with missing or incomplete information. Categories unrelated to the target (the output rebound angle) were discarded to simplify the process. We also used binning on continuous data (i.e. splitting the offensive zone into twelve zones, the net into six zones, and the rebound bins into nine zones of 30° angles), as seen in Figure 1.

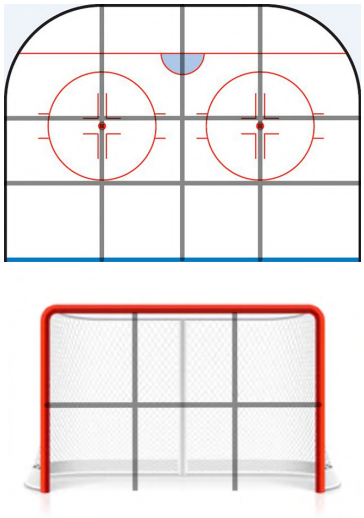


Figure 1: A view of the team's Graphical User Interface displaying the binning performed on both the ice surface and the net.

The final result of the preprocessing stage was a dataset where every datapoint had complete information in the categories that we as a team had deemed most predictive: the shooter's coordinates on the ice, and the shot target location on the goaltender (i.e. high glove, five-hole, etc.), as well as the target feature of output rebound angle.

2.2 Model Creation and Optimization

Once the dataset was refined, we began creating models. After familiarizing ourselves with the different prediction techniques available and working through a trial-and-error process, we decided to proceed with a Deep Neural Network predictor using technology from Python's TensorFlow library. The first iteration of the design returned accuracy rates of between 15-20%. To increase this accuracy, we changed some of the model parameters, including layer sizes and the learning rate. Additionally, we doubled the size of the rebound output bins from their original values of 15% to their final values of 30%. As a result of these changes, we observed drastic increases in their accuracy, to the point where the final model was able to correctly predict the approximate rebound between 45-60% of the time. Once the model was finalized, we also created a Graphical User Interface (GUI) to display the results of the model and allow user interaction.

2.3 Model Evaluation

When evaluating the model, we primarily used logical reasoning and visual inspection to determine how successful the model was. When considering the situation and the unpredictability of hockey rebounds,

we were happy with these accuracy rates of around 50%. If the model was guessing at random, the accuracy rates would be around 9% due to the 11 possible bins. As all members of the team were familiar with hockey and typical rebound placement, we were able to confirm that the model's predictions made sense through analysis of output images taken from the team's GUI (shown below in Figure 2).

Figure 2: Display of rebound angle likelihood from the team's GUI.

3. RESULTS AND DISCUSSION

As mentioned earlier, our only real result to validate was the accuracy range, which was between 45-60%. Because of the entropy of the situation, the wide range was expected, and can be attributed to the randomness of the partitioning between training and test data. When compared to the initial projections that we made at the start of this project, we believe our final results outperformed these initial benchmarks. From this, it's evident that our methods did work as expected as all decisions made throughout the process led to these final results. Over the year, we learned a lot, both in our work with new artificial intelligence technologies and techniques as well as our new exposure to the world of sports analytics. We believe that our results and accuracy have reached a level that could make them valuable to hockey teams looking to analyze and optimize their performance and success.

4. CONCLUSIONS AND FUTURE WORK

In conclusion, we were happy with the results and progress made over the year. Our stages of initial research/resource acquisition, data preprocessing, and model creation/refining all helped us build unique skills along the path to creating a working product. If this project were to be continued in the future, several steps could be taken. The first of these would be to make further improvements to the model. This could include acquiring more data from different sources, or perhaps including different predictors to create one ensemble prediction network, which typically can achieve higher results than any other single predictor. Another step would be to present this data to actual teams (i.e. youth hockey) and ask them to try building strategies around it and report the results to confirm that the theoretical results are applicable to a real-world situation. The final future step for our project would be to acquire a mass amount save data from a specific goalie to train the model on to accurately analyze tendencies that the goalie has. With the increased focus on data analytics, including the NHL's plans to accurately track the players and puck at all times [4], we as a team believe that this field of hockey analytics has a very promising future.

REFERENCES

[1] Wheeldon J. *The Future of Hockey Analytics.* TheHockeyWriters.com. September 25, 2017. Accessed on: March 24, 2020. [Online]. Available: https://thehockeywriters.com/future-of-hockey-analytics/

[2] Basen N. *How Analytics Will Help Win the Next Stanley Cup*. TheWalrus.ca. November 13, 2019. Accessed on: March 25, 2020. [Online]. Available: https://thewalrus.ca/how-analytics-will-help-win-the-next-stanley-cup/

[3] Yu D., Boucher C., Bornn L., Javan M. *Playing Fast Not Loose: Evaluating team-level pace of play in ice hockey using spatio-temporal possession data.* SportLogiq.com. March 2, 2019. Accessed on: March 24, 2020. [Online]. Available: https://sportlogiq.com/en/news-views/sportlogiq-s-research-papers-at-sloan-sports-analytics-conference

[4] Cotsonika, N. *Puck, Player Tracking in final testing stage before Stanley Cup Playoffs.* NHL.com. March 3, 2020. Accessed on: March 24, 2020. [Online]. Available: https://www.nhl.com/news/puck-player-tracking-technology-unveiled-during-2020-postseason/c-315806398

Cyber Security

Generative Adversarial Network for Antivirus Evasion

Will Macdonald[1], Will Coffell[2], Connor Chappell[3], Cameron Morrison[4], Ryan Saweczko[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: 16wjgm@queensu.ca
2 e-mail: 16wjac@queensu.ca
3 e-mail: connor.chappell@queensu.ca
4 e-mail: 16cjm20@queensu.ca
5 e-mail: 18rjs5@queensu.ca

Abstract: *The number of malware attacks and their capabilities have been increasing exponentially as the world relies more on technology. Due to this, the current heuristic detection based anti-malware solutions are becoming less effective. The team aimed to create a more effective solution to this using a machine learning based approach implementing a Generative Adversarial Network. From the EMBER dataset, vectors of features describing both benign and malicious files were input to the network for model training. Both the Generator and Discriminator trained against one another, exhibiting the converging behavior expected from a Generative Adversarial Network making the Generated samples practically indistinguishable from benign samples, proving the system is viable given a large enough batch of files. Next steps involve reconstructing these labeled files to further confirm functionality, after Generator modification, against current state-of-the-art anti-malware solutions.*

---

1. INTRODUCTION

1.1 Motivation

With the use of computers in day to day life growing exponentially over the last decade, research shows that the deployment of new malware has also followed the same trend, leading to financial losses and the exposure of private data [1]. With the rate of malware production ever increasing, the current solutions for malware detection are falling short in performance. Many current solutions rely on a heuristic feature identification approach in which the anti-malware system parses the files for code fragments, file hashes and properties that are present in known malicious files [2]. This trend in malware production led researchers and industry leaders to begin implementing machine learning algorithms to replace the current state-of-the-art solutions.

This paper will explore the implementation of a Generative Adversarial Network (GAN) as a proof of concept for training two machine learning models; the Generator – alters malicious files to evade detection from anti-malware software; the Discriminator - classifies files as either malicious or benign. Both models could have application in industry; the Discriminator would assume the role of an anti-malware solution, and the Generator can provide useful data for training new detection models on malware files that are significantly more difficult to detect than their respective versions unaltered by the generator.

1.2 Related Works

During preliminary research, a paper titled "Generative Adversarial Malware Examples for Black-Box Attacks Based on GAN" [3] gave insight into a proposed GAN titled MalGAN - released as an open source research project on GitHub [4] - which aimed to conduct malware attacks on a Black-Box detection system. The goal of MalGAN was to develop a model for creating false negatives as opposed to the typical cybersecurity approach of mitigating false positives in detection.

The MalGAN algorithm implemented a different structure to the typical GAN; using three models (Generator, Black-Box Detector and Substitute Detector) as opposed to the typical two models' structure (Generator and Discriminator). The Black-Box Detector is in place to predict which form of anti-malware system is being attacked. The methods to accomplish this are explored in depth in the research paper [3]. The goal of the Substitute detector in this architecture is similar to that of the Discriminator in the two model architecture where it classifies samples as malicious or benign; though it differs

from the Discriminator such that it attempts to fit to the Black-Box detection system (i.e. uses the system detected by the Black-Box detector to predict the samples), then uses this to classify the adversarial examples in the same manner the anti-malware system it's attacking would.

1.3 Problem Definition

The initial goal was to develop a way of classifying whether a file is malicious or benign. Based on the approach taken in the related work MalGAN, the concept of maximizing the number of false negatives that the Generator could force the Discriminator to produce played a large part in the definition of the design. The team, however, differed slightly from MalGAN by implementing the typical two model structure of a GAN, allowing the Discriminator act as its own malicious file detection system as opposed to using the Black-Box detection system to model a pre-existing anti-malware system. The reasoning behind this decision was to have two models that could independently function once fully trained as opposed to MalGAN, which is designed to attack detection systems.

2. METHODOLOGY

Upon conducting research into malicious datasets, the paper "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models" [5], and its related open source repository [6] were discovered, that provided the necessary labeled data needed for the planned architecture.

EMBER is composed of pre-extracted features from 1.1 million portable executable (PE) files. The dataset is divided into 400,000 malicious file features, 400,000 benign file features, and 300,000 unlabeled file features. The pre-extracted features are represented in an array of length 2381 in which each index represents its own feature from the file. These extracted features are explained more in depth in the paper [5].

Using this dataset, an iterative approach was taken toward the construction of the GAN; first starting with data visualization and preprocessing. Using libraries such as NumPy, Pandas, and TensorFlow to create a visualization of the data by which the team was able to better understand the contents of EMBER. These libraries also allowed for the modification of the data from its current state in EMBER into tensors of correct dimensions and shapes to input into the architecture.

Upon completion of the data processing phase, the next iteration was to develop a baseline model of the GAN that would later be improved upon. An optimization phase followed the initial development. Training checkpoints were implemented, in which the weights of the model could be saved and reloaded at a different time. This iteration also implemented a more complex training process that involved the concept of model balancing [7]; relying on freezing the trainable parameters of one model while the other is training and vise-versa at a ratio of one-to-one. Model balancing was then improved by implementing dynamic freezing/unfreezing mechanic based on the loss scores averaged over the past 3 epochs of training (i.e. if the average loss score was not improving, that model would unfreeze and train to improve its loss scores). The last optimization made was the implementation of dynamic batch sizes, where the batch size per epoch would increase as training progressed, ensuring that the more trained the model becomes, the more data samples it is exposed to in later epochs.

In the next iteration, aim was shifted to testing the GAN against current state-of-the-art applications for malware detection. To do so, reverse engineering of the features extracted from the original PE files was required to recreate the file with altered features from the GAN. This step required access to the original PE, something that EMBER did not provide. This issue required the creation of a dataset of PE files collected and tested through VirusShare [8] and VirusTotal [9]. Upon creation of the dataset, the team was able to implement EMBER functionality to extract the features used in EMBER's dataset from the dataset of PE files. These were passed through the GAN to receive adversarial examples of features to which the team had the original PE. The final iteration was planned to use these adversarial examples in pair with the library LIEF to recreate these features into a PE file.

3. RESULTS AND DISCUSSION

Evaluating the performance of GANs differs significantly from evaluating performance of other machine learning models, as GANs rely solely on loss functions until a point of convergence is reached [10]; at which point the outputs of the generator become indistinguishable from benign files. In other words, there is no objective function aside from loss functions that can be used to assess the progress of the model throughout training.

In the ideal case where the two models are trained equally and neither take precedence over one another, the accuracy of predictions made on adversarial examples by the Discriminator would be 50%; since once the point of convergence is reached, the discriminator will no longer be able to differentiate between a benign example and an adversarial example.

As seen in *Figure 1* during training, the GAN could reach the convergence state in which predictions were 50 ± 3% in favour of predicting adversarial examples as benign.

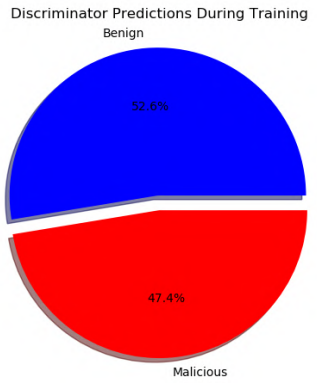


Figure 1 - Predictions made on adversarial examples depicting convergence of the models in training.

Finally, as depicted in *Figure 2* we can see behaviour of the loss functions for the Generator and Discriminator during the same training instance related to *Figure 1*.

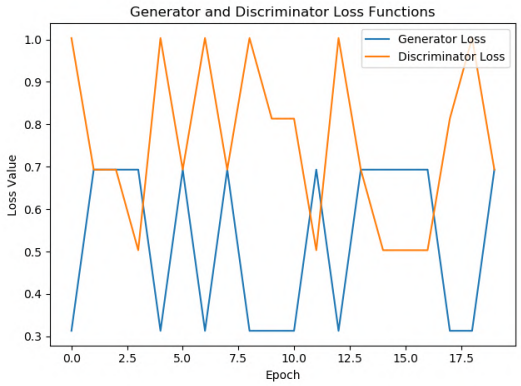


Figure 2 - Graph of loss functions throughout the duration of training. Depicting the inverse relation between the performance of the models.

The behaviour of these two Loss functions is interpreted as the relation between the models in the GAN. As the Discriminators loss score decreases, the Generators loss score is the inverse, making the Generators loss increase.

4. CONCLUSIONS AND FUTURE WORK

After brainstorming different model structures that would allow for a malware detection neural network, it was decided the best approach was to create a GAN. The motivation behind this choice was due to the flexibility of the structure regarding malicious file input. Since a GAN has two separate neural networks training adversarially, the Discriminator will experience a wide range of inputs from the generator, resulting in a more formidable final model. The EMBER dataset was used as it provided the ideal format and size for neural network training. Once decided on EMBER, models of the Generator, Discriminator, and their respective optimization functions were developed.

After functionality, testing began. Issues with how the models were simultaneously updating their weights, required a model balancing method to be implemented. Once the system was working, a set of 100,000 malicious files was used to train the models, after which the GAN achieved the desired behaviour of convergence.

Justifying the viability of the proposed solution in practice is something the team will continue to work on. Research into creating original PE files has been completed, however the difficulty lies in reconstructing these files. The generator only accepts tensors as it's input, so these files' major features are extracted into tensor form before use. Once the generator has altered the features, it must be converted back into a working PE. Otherwise it's unclear if the generator's modification of the file rendered it functionally benign. Due to the nature of constructing PE files let alone reconstructing a malicious PE file from features, very little scientific research is made publicly available. Making the process significantly more difficult and out of the time frame for this proof of concept.

REFERENCES

[1] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware Detection using Machine Learning and Deep Learning," *ArXiv190402441 Cs*, vol. 11297, pp. 402–411, 2018, doi: 10.1007/978-3-030-04780-1_28.

[2] Kaspersky Company, "Machine Learning Methods for Malware Detection," AO Kaspersky Lab, Moscow, 2020. [Online]. Available: https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[3] W. Hu and Y. Tan, "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN," *ArXiv170205983 Cs*, Feb. 2017, Accessed: 24-Mar-2020. [Online]. Available: http://arxiv.org/abs/1702.05983.

[4] Y. Lai, *yanminglai/Malware-GAN*. 2020.

[5] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *ArXiv180404637 Cs*, Apr. 2018, Accessed: 24-Mar-2020. [Online]. Available: http://arxiv.org/abs/1804.04637.

[6] *endgameinc/ember*. ENDGAME, 2020.

[7] J. Hui, "GAN — Ways to improve GAN performance," *Medium*, 10-Mar-2020. https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b (accessed Mar. 24, 2020).

[8] "VirusShare.com." https://virusshare.com/ (accessed Mar. 24, 2020).

[9] "VirusTotal." https://www.virustotal.com/gui/home/upload (accessed Mar. 24, 2020).

[10] J. Brownlee, "How to Evaluate Generative Adversarial Networks," *Machine Learning Mastery*, 25-Aug-2019. https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/ (accessed Mar. 24, 2020).

Recurrent Neural Network Password Cracking

Alastair Lewis[1], Lewis Hillard[2], Patrick Lister[3]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: alastair.lewis@queensu.ca
2 e-mail: 18ljh@queensu.ca
3 e-mail: patrickrlister@gmail.com

Abstract: *Password Cracking is a crucial field in Offensive Security, most techniques used for password cracking are inefficient and are not able to crack a significant number of hashes in a reasonable amount of time. Our project looks to employ a Recurrent Neural Network trained on common passwords to produce a set of possible passwords to use to crack as many hashes as possible. Cracked passwords are then added into the training set allowing for the Recurrent Neural Network to produce higher quality passwords overtime. We aim to be able to achieve a significant performance advantage over typical brute forcing methodologies.*

---

1. INTRODUCTION

1.1 Motivation

Offensive Security has become one of the fastest growing areas in the Computing industry. This area of computing involves the use of hacking techniques in an ethical manner in order to search and identify security flaws in a computer system so that when a hacker with malicious intent tries to exploit the system, the owner can be prepared. Data Security has quickly developed into an industry that is crucial to the success of businesses, government, healthcare and personal use. As Data Security has become more prevalent, so has the number of hackers that wish to steal data and exploit it for personal gain. In response to this, new careers have become available to combat hackers, namely penetration testers. A large area in penetration testing is password cracking, used to gain access to data that is meant to be protected and secured. Our motivation was to see if we could create an approach to password cracking superior to conventional tactics by implementing a Neural Network to assist in the cracks.

1.2 Related Works

Artificial Intelligence has been used for countless applications in the security industry from detecting anomalous network traffic to identifying malicious files and programs. At the time of the project's inception, no published sources had attempted to create a workflow quite like the one we had planned. We were able to consult several journals and articles that had attempted a subset of what our system could do. We found several guides on how to use a Recurrent Neural Network for text generation [1] as well as guides on trying to break hashing algorithms using Artificial Intelligence [2]. Both sources were extremely helpful in getting our project off the ground.

1.3 Problem Definition

When a password is used to authenticate a user, it is immediately converted to a unique, one-to-one mapping of a series of ascii characters known as a hash. Hashes are useful because it is impossible to obtain the password from the hash alone as the conversion process only works in one direction, ensuring that the plain text version of the password is never seen by anyone but the user. Cracking a list of millions of hashes using traditional brute forcing methods is incredibly inefficient and time consuming.

To improve upon this, we will train a Recurrent Neural Network with a set of real user passwords and have it generate a set of predicted passwords which we can then use to try and crack a subset of the list of hashes and extract their plaintext passwords. By adding the plain text passwords into the training set, we are able to continuously add more training data into the Neural Network and have it create higher quality password predictions over time, improving the rate at which passwords are able to be cracked. Ideally, this should allow for the user to be able to crack the list of hashes much faster than traditional brute forcing tactics.

2. METHODOLOGY

To create the system, we started by acquiring lists of passwords. The first set of passwords was obtained by searching the internet for lists of "commonly used passwords". We were able to find results that had obvious passwords. As seen in the figure below.

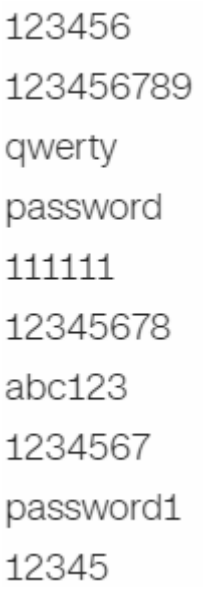


Figure 1: Examples of commonly used passwords

We used these passwords as the initial training set for the Recurrent Neural Network in order to have it be able to produce strings that resembled potential user passwords. Some examples of the passwords that were initially generated with a training set of 5000 commonly used passwords were the following:

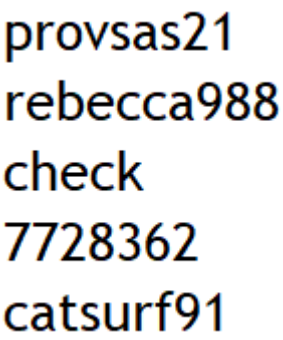


Figure 2: Generated Passwords with training set of 5000 Passwords

We also acquired lists of passwords that are used by real users. Leaked password lists such as the RockYou list are available for public download on the internet *by* the public and allowed us to deploy our system on hashes that are used by real people, making the system more applicable in real world scenarios. Some examples of passwords on the RockYou list are the following:

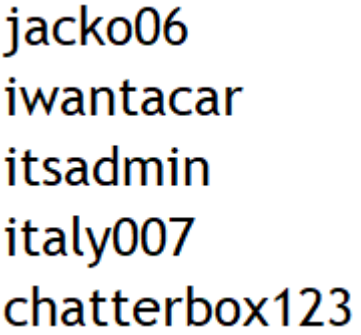


Figure 3: Real User Passwords from the RockYou List

After 10 epochs worth of training, we would have the Neural Network generate a set of 10000 passwords and use it to try and crack as many of the hashes as possible. Upon a hash being cracked, it would be removed from the hash list and its plaintext password would be added into the training set. The cycle would then repeat until either all hashes were cracked or no more hashes were able to be cracked by the Network for 3 cycles in a row.

3. RESULTS AND DISCUSSION

What we found upon testing the system was that while it did its intended function, it required too much computing power to be able to crack a significant number of the hashes in a realistic amount of time. Running on a relatively powerful Desktop with a GPU, we were only able to crack on average 5 passwords per training cycle. Each training cycle and subsequent cracking session took around 3 minutes to complete.

This meant that cracking a list of several million hashes was just unrealistic. Performance issues aside, the passwords being generated by the Recurrent Neural Network were visibly becoming better and better every training cycle. By the 10th training cycle, the passwords being generated were like the examples below:

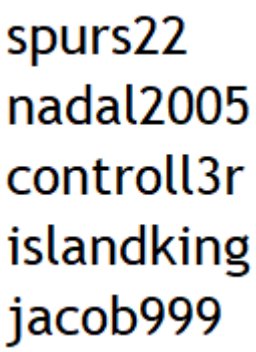


Figure 4: Generated Passwords after 10 Training Cycles

As we had hoped, this approach was able to crack passwords faster than traditional brute forcing. When we compared the rate of cracks between our system and brute forcing, we found that we were cracking 30 times more passwords in the same amount of time. This is because brute forcing a 10-character password that uses the entire ASCII table for possible characters has 1.09×10^{19} possibilities to try to crack which is essentially an impossible approach when trying to crack a list of the magnitude of several million hashes. An interesting application that we found with our system was the ability to pair it with the password cracking method of "Rainbow Tables" which is essentially a mapping of hashes to known plaintext passwords. Using our system, we were able to add uncommon password-hash mappings to the table that most tables don't contain. This improves the success rate and efficiency of using the Rainbow Tables for password cracks.

4. CONCLUSIONS AND FUTURE WORK

In conclusion, the project was an interesting exploration of the field of password cracking and how Artificial Intelligence can be embedded into a system to provide it with an advantage over traditional methods from both an efficiency stand point as well as generating data that regular programs wouldn't be able to produce. While we weren't able to achieve the performance marks we had hoped, we were able to complete our initial goal of improving over brute forcing methods. Given more time, we would spend more time trying to optimize the system by adjusting several parameters, namely; training set size and content, number of epochs to train, size of generated attack set, size of hash list to crack and Neural Network parameters. Hopefully this would improve our efficiency and allow us to crack more passwords in less time. We would also explore embedding our system into an actual Rainbow Table framework to see if our hypothesis of combining the two techniques would get the performance boost we anticipate. This project was an excellent way to explore the sectors of Offensive Security as well as Artificial Intelligence.

REFERENCES

[1] Gobeil, Yan. *Generating Pokémon names using RNNs*. 3 July 2019.

[2] James, Febin John. *How Artificial Intelligence Can Be Used For Password Guessing*. 19 September 2017.

Distributed Computing

QMIND Distributed Computing Team

Troy Giorshev[1], Duncan Mays[2], Dal Farra[3]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada

1 e-mail: troygiorshev@gmail.com
2 e-mail: DuncanMays@Outlook.com
3 e-mail: nickdf4@gmail.com

Abstract: *The QMIND Distributed Computing team, explored the use of distributed computing in the context of machine learning. As opposed to training a machine learning model on one computer, we were able to build a system to train the model across multiple computers, in an asynchronous and parallel manner. We built the system from the ground up, allowing it to be robust yet flexible. During CUCAI we deployed the system on a cluster of Raspberry Pi single board computers, with an associated visualization, successfully training a model.*

---

1. INTRODUCTION

1.1 Motivation

There are three main business benefits of Distributed Computing: Speed, Efficiency, and Privacy.

First, the benefit of speed is straightforward: a single computer can only be so fast. Past a certain point it is required that we have multiple computers working together.

Second, the benefit of efficiency comes from the fact that many computers are left idle for long periods of time, usually overnight. As opposed to these computing resources being left idle, they can be put to work at useful tasks. Distributed Computing allows for increased productivity for no additional capital expenditure.

Last, distributed computing gives the option of privacy in a way unparalleled by other techniques. This is best illustrated with a concrete example. Say you are looking to train a natural language processing model on text messages. Without distributed computing, you would need to ask people to send you their private text messages, from which you would compile a dataset, and you would train your model. However, we would expect that many people would refuse to let others read their private text messages. With distributed computing, the model could be trained right on the user's device, and with only the improved model being sent back. At no point does a central server ever access the private messages. With this increased privacy comes access to a wealth of new sources of data.

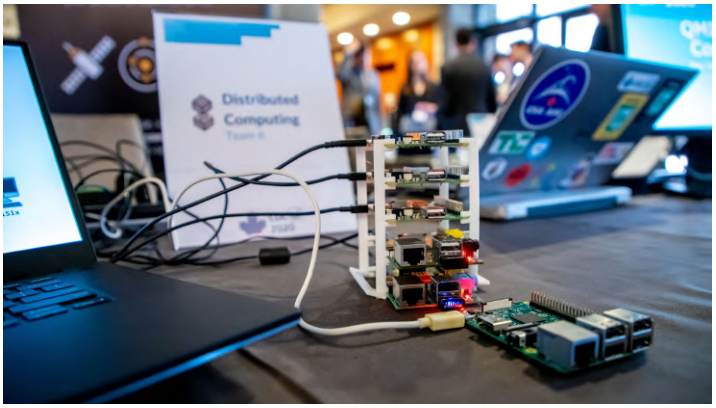


Figure 1: Raspberry Pi cluster, as presented at CUCAI 2020.

1.2 Related Works

There are many motivations for doing distributed training, and so there are many algorithms that optimize for a specific objective. Two complementary examples of this are Downpour [1] and Federated Learning [2], both were developed by Google to train neural networks on multiple, distributed, machines.

Downpour was designed for performance, to use the resources of multiple machines to train a neural network as fast as possible. Despite remaining a cutting-edge algorithm since 2012, Downpour is strikingly simple. Using a manager/worker configuration (one manager and many workers), Downpour consists of workers calculating gradients of a neural network with respect to local data, and then periodically sending those gradients to the manager, which will subtract them from the neural net stored in its local memory. The workers will also periodically pull parameter updates from the manager, syncing the parameters of the neural net in their memory to the neural net on the manager. Note the contrast between this scheme and a master/slave configuration. In a master/slave configuration, work is driven by the master and given out to the slave devices. The master must always be aware of the status of the slave devices and actively manage their workloads and connectivity. Downpour is worker-driven, workers push and pull gradient updates and parameters on their own schedule. This gives the algorithm robustness, should a worker lose connectivity or stop functioning for any reason, the algorithm will continue uninterrupted.

Federated Learning was designed for very different requirements. Its focus is not so much performance, but privacy. A large motivation for Federated Learning was the need to create text-prediction models on mobile phones. It is illegal to collect texting data from users' devices, and so Google could not create very good predictive models of their user's texting patterns. This changed with the realization that averaging the parameters of two equally trained neural nets will create a network with lower loss than either of them. Due to their convex loss surfaces, averaging neural nets is an effective way to combine the knowledge each has learned from training data. Federated Learning exploits this to preserve users' privacy. A model of a user's texting patterns is created on that user's phone, and then that model, instead of the texting data, is transmitted to Google's servers, where it will be averaged with the models from other users' phones and then redistributed. In this way, Google (and other telephone companies) can iteratively improve their text-prediction models, while not collecting private data.

In general, all distributed training algorithms are varied solutions to something called the knowledge-sharing problem. The knowledge-sharing problem is the problem of learning and encoding knowledge about a certain source of information so that other agents can acquire and use that knowledge, without going through the process of training on data. Knowledge-sharing is a common engineering problem across numerous fields of study, from communication to machine learning to mathematical control.

2. METHODOLOGY

In our setup, we have a single manager and multiple workers. The single manager holds the current "best idea" of the model. The goal is to update the model on the manager such that it improves as quickly as possible. Each of the workers has its own copy of the model, which they train with their own chunk of the data. Periodically during training, the workers will pull the "best idea" model from the manager and overwrite their own. As well, periodically during training the workers will push their model to the manager. The question is then the following: how best should the manager integrate the workers' models into its own "best idea" model? In general, this "integration" is called "Model Aggregation". A naïve approach is a simple averaging: every time a worker pushes a model the manager averages the weights and biases of the worker model with its own model. A slightly more sophisticated approach could be a method of weighted averaging, possibly weighted to the number of data points that each of the models has seen. One of the future goals of this project is to explore and compare different model aggregation methods.

3. RESULTS AND DISCUSSION

As stated before, the goal of our work was to explore distributed computing in the context of machine learning. Toward this goal, we built a distributed computing system, following the architecture laid out by Google's Downpour paper. This system was built from the ground up in Python, starting at the level of TCP Sockets. The advantage in doing this, as opposed to using an existing system, was twofold. First, it allowed us to understand more deeply the inner workings of existing distributed computing systems. Building something yourself gives a unique perspective into the challenges and advantages it holds. Second, it allowed us to make the system extremely flexible. Wherever our interests took us, we were confident that we could adapt what we had built to fit out needs.

We deviated from Downpour in one notable way. In Downpour, the workers send gradients up to the Master. In our system, the workers send the parameters of the model (weights and biases) instead.

4. CONCLUSIONS AND FUTURE WORK

Future work for this project can be laid out in two broad categories: optimizing Downpour and experimenting with different model aggregation techniques.

In Downpour, workers push and pull at two independent but preset intervals (measured in, say, number of batches of training data processed). However, the Downpour paper doesn't specify what these intervals should be. We look to explore what gives optimal training times. We suspect that some relationship between the individual worker's compute power and the compute power of the overall cluster will give optimal results. Additionally, we will look to explore if varying the push and pull intervals as training goes on would be a better approach.

In exploring different model aggregation techniques, as suggested above, we will try a variety of methods, comparing the results to our existing naive approach. We hope to find methods which balance performance with stability.

REFERENCES

[1] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng, Large Scale Distributed Deep Networks, Google, 2012.

[2] Jakub Konecny, H. Brendan McMahan, Daniel Ramage, Federated Optimization: Distributed Optimization Beyond the Datacenter, Google, 2015.

Finance

Asset Price Forecasting

Ethan Bonnardeaux[1], Yale Freedman[2], Kyle Oppenheimer[3], Ronan Almeida[4], Ben McKerry[5], Daniel Goldstein[6]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: 18ejb@queensu.ca
2 e-mail: yale.freedman@queensu.ca
3 e-mail: 15kao2@queensu.ca
4 e-mail: ronan.almeida@queensu.ca
5 e-mail: 17bdm1@queensu.ca
6 e-mail: daniel.goldstein@queensu.ca

***Abstract**: Over the past decade, machine learning and statistical modelling have become essential elements of business success. Our team was contracted by Two Rings Media to develop asset-forecasting models and data analysis tools to further understand non-intuitive trends within the construction and automotive industries. We developed a stacked machine learning ensemble using various models to capture sales seasonality and a data visualization web application. Our machine learning ensemble on average, varied from the selling price by $12,332 in a dataset with a standard deviation of $29,024. In addition, the model is extensible to include factors beyond seasonality. We are looking to expand our solution to different classes of assets and incorporate an initial asset cost in the training data to minimize model loss.*

1. INTRODUCTION

1.1 Motivation

The demand for asset forecasting models and business intelligence insights has been growing rapidly as many corporations have realized their effect on operations. Two Rings Media (TRM) is a technology and media firm that focuses on developing digital solutions and strategies for clients such as BMO, General Electric and Home Hardware. Our team partnered with TRM to develop software that improves their current Data Mining capabilities by utilizing Machine Learning practices.

1.2 Related Works

Bohdan M. Pavlyshenko is an accredited Machine Learning researcher who wrote a paper on *Machine-Learning Models for Sales Time Series Forecasting* [1]. This paper explores the idea of conducting an analysis on different time series and gradient boosting algorithms to determine the most effective model stacking architecture given different datasets.

1.3 Problem Definition

The client had already built a price optimization tool to forecast the sales price of tractor trailers based on depreciation of value using a linear regression model. However, this model does not include factors such as seasonality and location of sale, despite this data being available in the dataset. The team was therefore tasked with creating a tool to either visualize or create predictions with the data at hand to give the client some insight about a given tractor trailer that would not have been available with the previous linear regression model.

2. METHODOLOGY

2.1 Introduction

Due to the nature of our project, the first half of the contract timeline was spent researching different potential solutions and building proof-of-concept models to present to the client. Upon the proposal of these solutions, the client decided that we should pursue the development of a Data Visualization Web Application and continue research on machine learning solutions regarding seasonal asset price forecasting.

2.2 Ensemble Learning Model

To tackle the asset-forecasting regression problem, we decided to build a stacked machine learning ensemble by developing three different models (Gradient Boosting, Neural Network, Sarimax) and feeding their outputs into a second Neural Network.

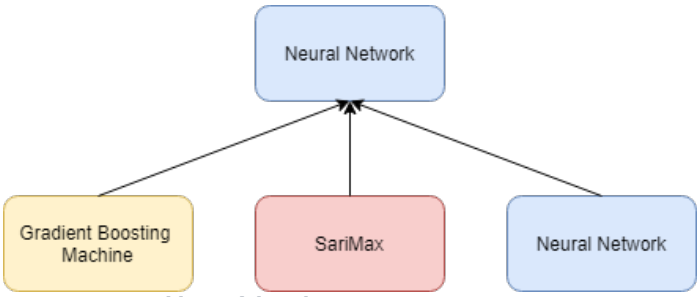


Figure 1: Ensemble Model Architecture

The architecture was designed to put a large bias on the seasonality of the asset using the SariMax model. This final Neural Network is composed of two 200 neuron hidden layers with ReLU activation separated by a dropout layer with a rate of 0.2, all of which feeds into the final output layer with a linear activation function. This was implemented with the purpose of combining the different model outputs while optimizing results.

2.3 Encoding

To utilize the categorical variables within the dataset, different encoding methods were implemented. The specific makes of tractors were grouped and converted into a numeric value via target encoding with formula.

$$Encoded\ Group = (n * x' + m * w)/(n + m)$$

The Location of Sale and Month of Sale were both encoded using Exclusive Feature Bundling (Fischer's Method) in conjunction with the Gradient Boosting model training to create optimal partitions within the respective classes and reduce the over-fitting. [2]

$$Partition\ Class = min\ (\sum_{i=1}^{n} w_i(a_i - a_i'))$$

2.4 Gradient Boosting and Neural Network

To achieve a strongly reduced training loss, the Gradient Boosting Regression algorithm was selected using Root Mean Squared Logarithmic Error as the evaluation metric. It was only boosted for 50 iterations concerning attempts to reduce the over-fitting of the data by restricting the length of the decision tree. The first Neural Network had 3 hidden layers with a dropout layer between the first and second, the activation functions utilized were ReLU and LeakyReLU. The model was trained using Mean Squared Error as the evaluation metric and AdaGrad as the optimizer.

2.5 SariMax Model

Due to inconsistent amounts of data for specific years, volume of sales per month did not accurately reflect demand. In order to represent demand for tractor trailers in each month, we decided to use the average sales price instead as a substitute to volume of sales. The model was trained with the yearly parameters {p=3, d=1, q=0} and seasonal parameters {p=5, d=0, q=0, s=12}.

2.6 Web Application

The Web Application was to be integrated within the TRM code base. This presented us with the challenge of re-creating their production environment. Our system design was created using a Linux-based virtual machine which runs a Docker SQL server container that interacts with the Django Application.

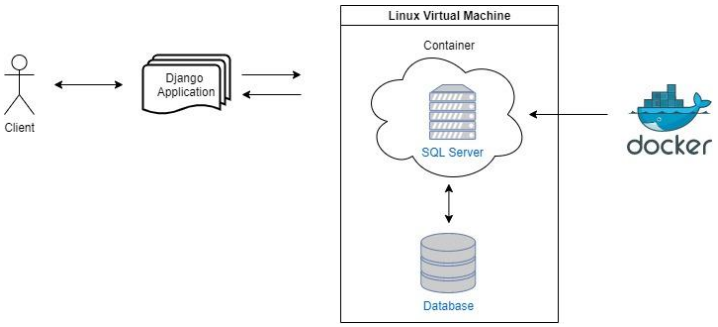


Figure 2: Web Application System Design

The front-end of the application allows the user to filter the data and visualize the exact subset of data through its features. The Histogram and Map features display location and seasonal based trends allowing user to quickly analyze the data and make decisions.

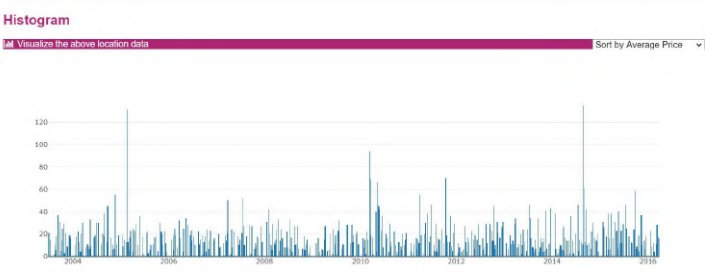


Figure 3: Histogram Application Feature

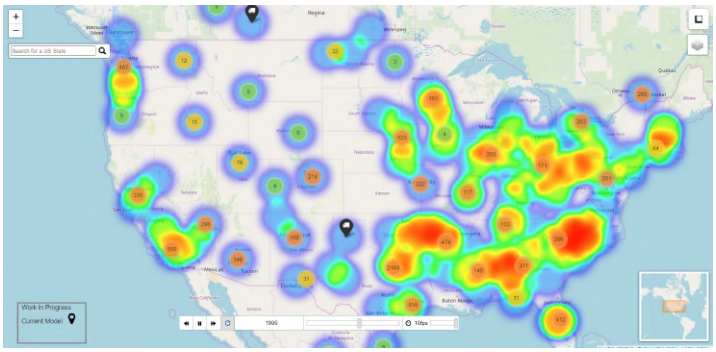


Figure 4: Map Visualization Application Feature

3. RESULTS AND DISCUSSION

The Ensemble model was able to achieve a validation Root Mean Squared Error [RMSE] of 12332.0 on the dataset with asset prices ranging between [1000, 150000] therefore the RMSE was 8.28% of the range.

Models	***Root Mean Squared Error***
Proof-Of-Concept	23185.7
Preliminary Neural Network	16921.4
Ensemble Model	12332.0

Table 1: Results of our models

The Gradient Boosting model achieved a RMSE of 12461.9 but did not incorporate a training-testing data split ensuring the Ensemble Model could randomize its data prior to training. Therefore, its RMSE is not directly comparable to the Ensemble Model. The model was trained with a testing validation split of 40% for generalization purposes. However, the validation loss did not converge with the training loss. A possible cause could be that Tractor prices had a large standard deviation (29024) and the large variety of makes (150 total). The model was designed to have each asset's initial price as an input, but we were unable to access the data within the contract timeline. From this project we can conclude that Ensemble Model was a successful technique and produces far better results than stand-alone techniques

4. CONCLUSIONS AND FUTURE WORK

The machine learning model is effectively able to pick up on trends in the dataset that were not accounted for in the client's previous depreciation based linear regression model. Training on the linear regression predictions will give the ensemble a picture of both the depreciated value of a given asset, as well as any seasonal and location-based indicators that could sway this value, leading to a far improved final model. We are also looking into adding another preliminary SariMax model that is trained on the time series data for the specific make of a tractor being predicted. Upon completion, the team will look to broaden the scope beyond that of tractor trailers. Many other construction vehicles and tools share the features of the tractor trailer identified in both the web tool and the machine learning model. Furthermore, they will follow the same construction industry trends that the current model is accounting for. Therefore, the team will look to broaden both the web tool, and the machine learning model to the construction industry as a whole, with the added challenge of incorporating any specific differences that come with various areas of construction into both the models and the web tool.

REFERENCES

[1] B. M. Pavlyshenko, "Machine Learning Models for Sales Time Series Forecasting," 2010.

[2] A. Othmen, "Know about Categorical Encoding, even New Ones," Medium, 2019.

Financial Planning with Machine Learning

Elan Bibas[1], Andrew Fryer[2], Elikem Hermon[3], Ekim Karabey[4], Sylvain Plouvier[5]

QMIND –AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: 17emb8@queensu.ca
2 e-mail: andrew.fryer@queensu.ca
3 e-mail: 17ekh@queensu.ca
4 e-mail: 18ebk@queensu.ca
5 e-mail: s.plouvier@queensu.ca

Abstract: *Motivated to understand potential applications of Machine Learning algorithms in Financial Planning and Analysis industry, we sourced data from publicly traded companies, cleaned it with basic statistics, and fit several models to it. The linear regression model performed best. We will be continuing research with more complex models relating more factors.*

---

1. INTRODUCTION

1.1 Motivation

Many large companies have Financial Planning and Analysis (FP&A) teams. Their goal is to accurately predict the expenses and revenues the company will see in upcoming years. These predictions are very valuable to the company executives for making business decisions.

In the age of information, these companies have very detailed records about their financials. Since predictions based on past data can be accurate, but not correct, machine learning (ML) methodologies are a good fit.

This project is focused on Income Statement, Balance Sheet, and stock price data for the top 100 S&P 500 companies in the utilities industry because this data is readily accessible. Models will be fit to the data with the hope of determining whether these models are useful for FP&A.

1.2 Related Works

Given the lucrative nature of the finance industry, it is no surprise that all avenues of data manipulation, visualization, and forecasting are being explored and refined. This is exemplified through the rise in high-speed algorithmic trading and comprehensive financial forecasting models.

There are several popular ML models that attempt to achieve this. Imhgchoi's future price correlation model predicts the price correlation of two assets for future time periods.[1] This is particularly useful in maintaining a balanced portfolio.

Long Short Term Memory (LTSM) recurrent neural networks (RNN) are used in predicting the stock price correlation coefficient of two individual stocks. RNN's are used for their ability to comprehend and manage temporal dependencies. Furthermore, the use of LSTM cells increases its long-term predictive capacity.

1.3 Problem Definition

The scope of the project is to learn about timeseries ML models, understand financial data from the S&P 500 utilities companies, and to determine the feasibility of using ML models for FP&A.

In particular, this project investigates simpler models than those used in related projects. Since linear models are currently used in industry by finance teams, we hope that contrasting the performance of linear models with other simpler models will provide insight into

how finance teams can adopt this emerging technology without learning about complex machine learning concepts.

2. METHODOLOGY

2.1 Data Collection and Cleaning

A csv file containing the stock symbols, names as well as industries for companies in the S&P 500 was retrieved from Wikipedia. The Financial Modelling Prep API was then used to obtain stock prices, income statements, balance sheet and cash flow reports for companies in the utilities industry. The stock prices were averaged around the date of the quarterly published financial statement of each company to further refine the data. This data was then exported as a json file for use in subsequent stages of the project.

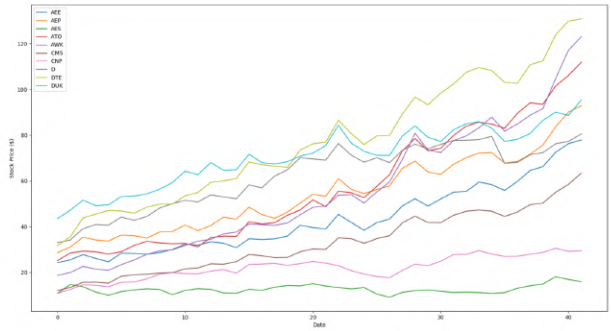


Figure 2: Stock prices for ten companies in the S&P 500 dataset

2.2 Model Consideration

We considered various time series ML models such as ARIMA, linear regression and k-nearest neighbors (KNN). KNN was eventually abandoned as it is better suited for classification and less for time series forecasting.

2.3 Procedure

The scikit-learn library was used to provide access to linear regression and ARIMA models. These models were used to predict stock prices, revenue and expenses for the companies present in the exported json file. After retrieving the relevant information from the file, we proceeded to split the data into 70% for training the model and 30% for testing. The training data was then used to fit the model. Once the model was trained, we predicted values for the testing set of date values. We then used matplotlib to plot the predicted values against the actual test data.

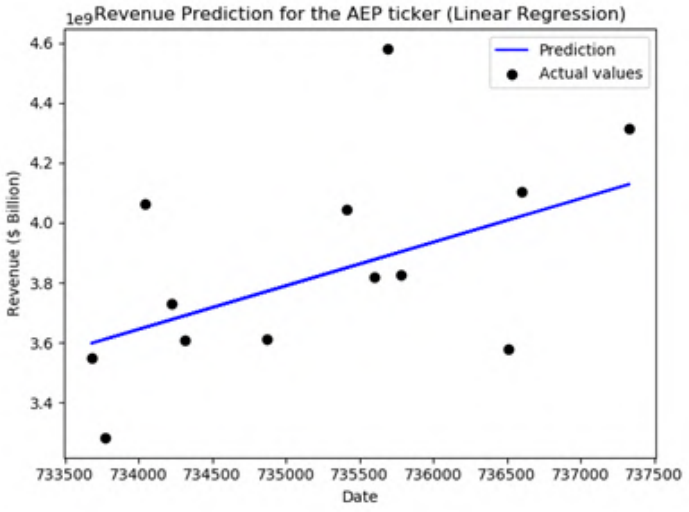
2.4 Evaluation

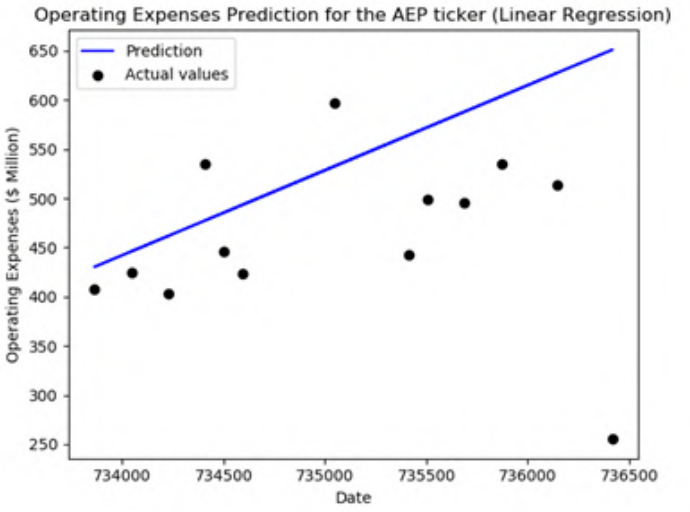
For the linear regression model, its accuracy was quantitatively measured by observing the mean squared error as well as the variance score. For the ARIMA model, the accuracy of the prediction was quantitatively evaluated by considering the mean squared error produced. The predictions for both models were qualitatively evaluated by comparing the plot of the predicted values to that of the actual values.

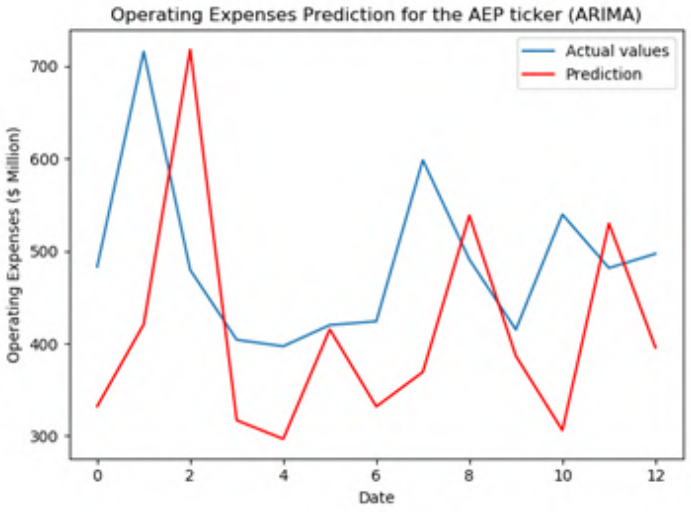
3. RESULTS AND DISCUSSION

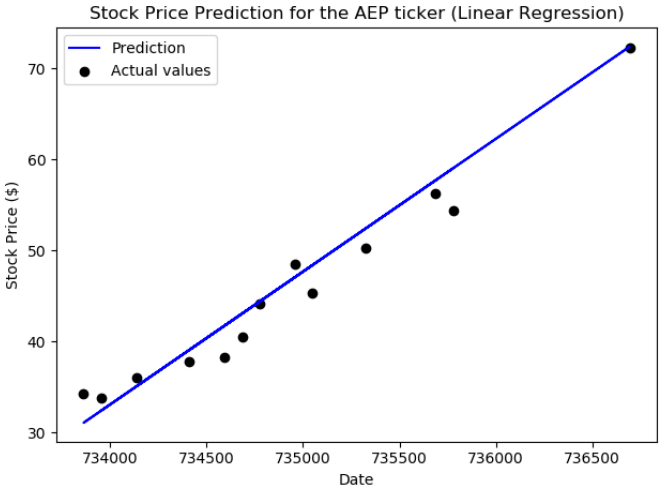
It was expected that the ARIMA model would perform better than the linear regression model because stock data can be very volatile, with peaks and dips, and so a line of best fit will not have a strong correlation to the data. Furthermore, the ARIMA model has the potential to have a much stronger correlation to stock data with its ability to have peaks and dips. However, in practice, we observed that the ARIMA becomes overfit and does not perform as well as the linear model.

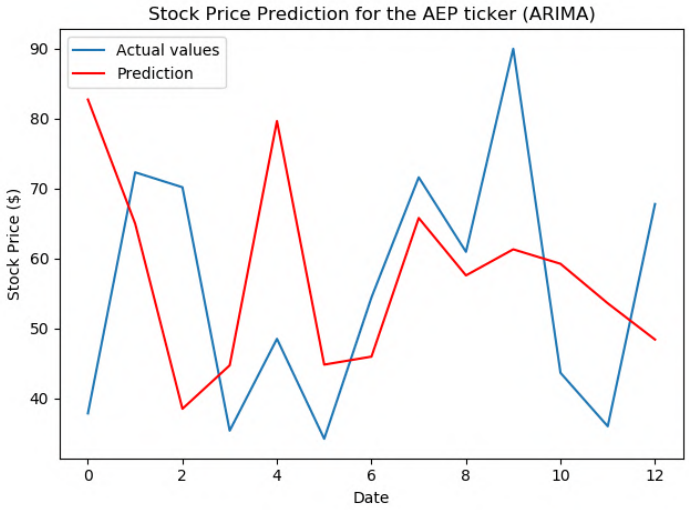
Please see the following graphs of the linear regression model's predicted vs actual values for revenue, operating expenses, and stock price.



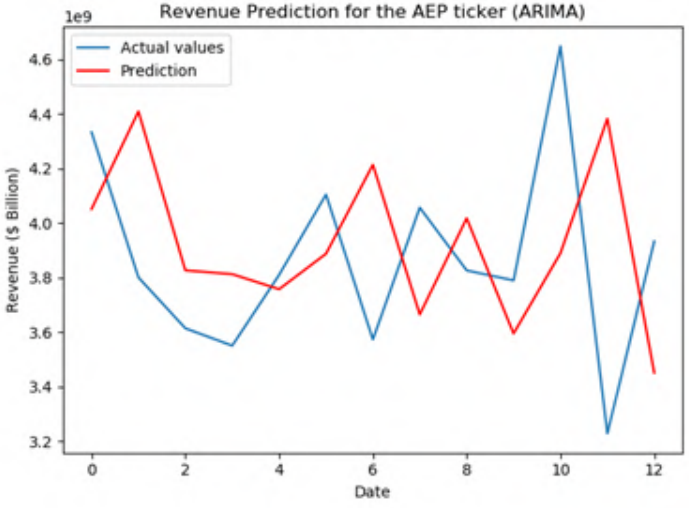








Please see the following graphs of the ARIMA model's predicted vs actual values for revenue, operating expenses, and stock price.



Both linear and ARIMA models were trained and tested 10 times, and the average root mean square errors for AEP's stock price were 3.965 and 19.103 respectively. This means that the ARIMA model actually did not perform as well as the linear model.

4. CONCLUSIONS AND FUTURE WORK

Our results showed both linear regression and ARIMA models to be fruitful, though the former yielded better results. In the future, we plan on refining our linear regression model using a k-folds cross validation. Still, a challenge we have to overcome are the periodic non-linear trends between cash flow and other metrics. To find a better fitting model, TensorFlow Estimator may be used.

REFERENCES

[1] Imhgchoi, "Correlation Prediction with ARIMA-LSTM Hybrid Model", GitHub, October 2018

Health

Akira Patient Prediction

Willem Atack[1], Jacob Johnston[2], Angus McInnes[3], Alexander Nechyporenko[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada

1 e-mail: willem.atack@queensu.ca
2 e-mail: jacob.johnston@queensu.ca
3 e-mail: 17asm11@queensu.ca
4 e-mail: 15ann2@queensu.ca

Abstract: *This project is partnered with a virtual health care startup called Akira which staffs nurse practitioners to virtually advise patients. An important part of their business model is deciding on the number of nurse practitioners that are staffed throughout the day. Using past data, the team determined that machine learning and statistical forecasting could be an appropriate solution to forecast how many patient inquiries will be raised a week in advance. A SARIMA model and a recurrent neural network were applied to this problem, and the team received an error of approximately 3.4 patient inquires. The team found that the SARIMA model was not accurate enough, and that a deep learning model would be needed to draw detailed insights. The RNN did outperform the SARIMA model, however the model is still incomplete with lots of optimization to be done.*

1. INTRODUCTION

1.1 Motivation

This project was partnered with a virtual health startup called Akira. Akira provides virtual care to patients using their platform via the use of messages, calls and video calling. The company makes money by having users sign up for their service, and their main costs involve staffing nurse practitioners and doctors across the country who are readily available to support clients. The purpose of this project was to forecast the number of patient inquires that Akira will receive up to a week in advance. The idea of this is that better knowledge surrounding the expected inquiries will enable better decision-making regarding staffing decisions. Over-staffing causes the company to lose money by paying for more labor than is really required. Under-staffing could cause long wait times for patient inquires and thus reduce the quality of Akira's service. Thus, optimizing the number of health-care professionals staffed at any given time is extremely important of their business model, and our team determined that statistical forecasting and machine learning could provide a solution.

1.2 Related Works

The team read several articles, tutorials and papers regarding time series forecasting to begin this project. In particular, the team paid attention to sales forecasting as this problem is essentially with the problem that the team was dealing with for our project. The team soon identified ARIMA, SARIMA, and recurrent neural networks to be the most popular and effective models for forecasting [1]. These models were researched extensively, particularly within the context of forecasting univariate data. The team also read papers discussing patient number predictions for hospital emergency rooms, where it was suggested that external variables such as weather and date could affect forecasted values [2].

1.3 Problem Definition

The exact problem at hand is to predict the expected number of patient inquires exactly one week in advance of when the prediction was made, based on previous patient inquiry volume data, and external variable such as weather and precipitation.

Time Stamp	Inquiry Count	Precipitation	Avg Temp
9/4/2019 16:00	23	0.34	12

Figure 1. Sample training example

2. METHODOLOGY

2.1 Data Preprocessing

The team was presented with a dataset including every action that Akira users made for a year. This data was first condensed into unique user inquiries, and then parsed to aggregate the number of inquiries during each hour of that year. In addition, the location of the inquiry was kept as a third feature, such that the location of the user could be used to determine the weather in the user's area. Then, this feature could be replaced by the user location's weather, which included the average temperature on the day of the inquiry and the amount of precipitation. A sample training example is shown in Figure 1. The tea used this dataset to fir to both a SARIMA model, and a recurrent neural network.

2.2 SARIMA Model

ARIMA is short for 'Auto-Regressive Integrated Moving Average', while SARIMA indicates a 'Seasonal ARIMA' model. In essence, ARIMA modelling uses the residuals from previous data points to forecast future values. The model does this by comparing predicted values to their actual values using a split data set. Optimization can be done by adjusting the parameters of the overall ARIMA equation, shown below [3].

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

The AR (auto-regressive) parameter, usually denoted by a 'p', controls how many lag periods are used by the model. Models utilizing only auto-regression closely resemble a linear regression. The MA (moving average) parameter, denoted by a 'q', indicates the number of lag errors used. A moving average helps reduce random noise. The differencing parameter, denoted by a 'd', refers to the order of differencing used [3]. Differencing renders the model stationary, keeping the mean and variance constant and facilitating predictions. A fourth parameter is included for SARIMA models, denoted by a 'm', referring to the number of periods per season [4]. An example of a SARIMA forecasting output is shown in Figure 2.

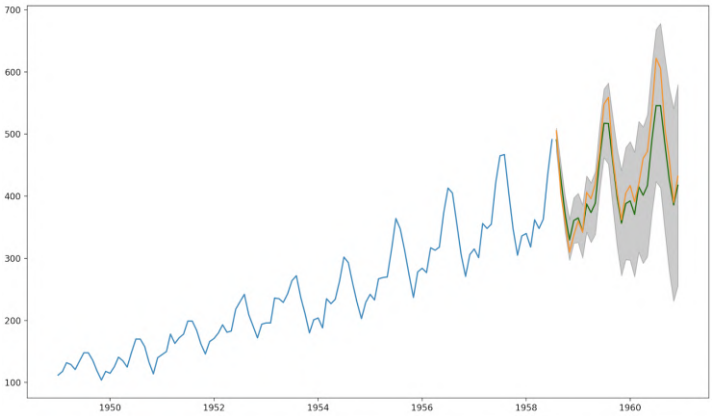


Figure 2: Sample SARIMA output for a split dataset; orange is predicted values, green is actual data, grey is error bounds.

General trends become apparent, while predictions lack accuracy, which is why the team thought it could improve the results using a deep network.

2.3 Recurrent Neural Network

Before feeding data into the neural network, the team windowed the dataset into windows of 168 consecutive time series values (one week of data), where the last value corresponds to the tag value, and the first 167 values were the features. Once this was done, the team fit 80% of the data to a simple neural network. Once this was completed, we changed the layers from Dense to Recurrent. This is because recurrent layers enable the network to consider how the features affect each other, making this model type ideal for sequence models. Once this was done, some simple optimization regarding number of epochs and learning rate was done, enabling a final model.

3. RESULTS AND DISCUSSION

3.1 SARIMA Model

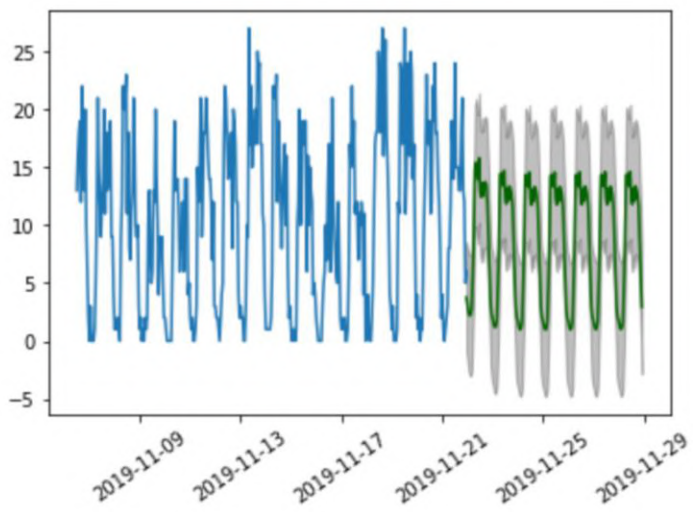


Figure 3: The output of the SARIMA model highlighting the predicted number of incidents, hour by hour, to be anticipated a week in advance

Based on the resultant forecast, the model was able to predict the number of incidents per hour with normally distributed errors and a minimum average error (MAE) of 3.70831956. To determine if these results are satisfactory, further work with Akira is required to quantify what degree of accuracy is necessary in incident prediction. Ideally, the model should yield results within error less than the number of incidents a doctor or nurse can address within an hour. If this is the case, then the model results will be deemed sufficient as AKIRA can accurately staff healthcare professionals within one person. As seen in Figure 3, it should be noted that the SARIMA forecasts did not reach the extremes of the data, and thus gave large error bounds and

3.2 Neural Network

As expected, the neural network did slightly improve upon the ARIMA model, giving a final MAE of around 3.4 patients every hour. However, the team never managed to fully integrate the external variable into our model, so we feel as though this is just the tip of the iceberg in terms of accuracy, as much more analysis could be done to improve this model. This is magnified by the fact that, unlike the SARIMA model, the RNN's forecasts were able to reach the extremes of the actual data.

4. CONCLUSIONS AND FUTURE WORK

To conclude, the team created two types of models to forecast a week in advance to aid Akira's operation team with staffing to reduce costs and optimize quality of service. As of now, our results are likely insufficient for Akira. The SARIMA model captured the general trend but lacks accuracy. The neural network has the potential to perform accurate forecasts but is incomplete. To finish this model, the team would like to do a great deal of optimization and analysis. This includes finding the ideal learning rate, neural network architecture, lags considered, number of epochs, and integrating in the external weather data. We feel there is potential to solve the problem by taking the above steps. In addition, the team would like to add a feature capturing the length and complexity of each patient inquiry. This analysis would enable the team to inform the operations team to make better decisions by explicitly predicting how many nurse practitioners should be staffed at any given hour.

REFERENCES

[1] "SARIMA models," DUKE,2004.[Online]. Available: https://people.duke.edu/~mau/411arim.htm#pdq. [Accessed 2020].

[2] "Time Series Forecasting," TowardsData Science, 2018.[Online]. Available: https://towardsdatascience.com/time-series-forecasting-arima-models-7f221e9eee06.[Accessed 2020].

[3] S. Prabhakaran,"Recurrent Neural Networks," Machine Learning Plus, 2019.[Online]. Available: https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/. [Accessed 2020].

[4] J. S. Armstrong,"Sales Forecasting," Penn Libraries, 1999.

Forecasting Internal Medicine Bed Occupancy for Kingston General Hospital

Antonio Braulio[1], Huang David [2], Bakhtyari Ariana [3], Pasha Tabassum[4], Gauthier Christophe[5], Boxer Shiyan[6]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: 19bfac@queensu.ca
3 e-mail: 17ab70@queensu.ca
4 e-mail tabassum.pasha@queensu.ca
5 e-mail: chris.gauthier@queensu.ca
6 e-mail: shiyan.boxer@queensu.ca

Abstract: *Over a period of 8 months we worked with the Kingston Health Sciences Center to help them solve problems associated with the bed occupancy in the Internal Medicine department of Kingston General Hospital (KGH). Like most other hospitals in Ontario, KGH is also operating at capacity and sometimes over capacity during surges. The largest associated cost with surges staff overtime, in particular, nursing staff. To help them decrease this overtime cost we developed and tested a Multilayer Perceptron Neural Network, and an ARIMA model, that would allow the hospital to predict the Internal Medicine bed occupancy. With our neural network we were able to forecast 455 days into the future, with the RMSE of 7.63 and a MAXE of 20.78. The ARIMA model can predict for the next day, with an MSE of 33.249.*

---

1. INTRODUCTION

1.1 Motivation

Almost every hospital in Ontario is currently running overcapacity, as there are too many patients and not enough hospital beds. In 2013 Ontario had the lowest number of beds per capita in Canada (2.4 hospital beds for every 1,000 residents) [1]. Patients are waiting for hours in the emergency department and being treated in hallways is no longer uncommon [2]. Two main causes of this issue are population growth and patients designated as Alternate Level of Care (ALC). An ALC patient is someone who occupies a bed in a hospital and does not require the intensity of resources/services provided in an acute care setting [2]. To make long term improvements, a significant amount of capital will need to be invested in expanding hospitals, long-term care spaces, and other forms of continuing care. In the short run, we can leverage AI/ML to help hospitals operate more efficiently and respond to surges in a more cost-efficient manner, which under current circumstances, put an immense amount of stress on hospital staff and resources when they occur.

1.2 Related Works

In a research paper by E.Kutafina et al. called "Recursive neural networks in hospital bed occupancy forecasting", they tackled this problem by constructing a model based on a set of recursive neural networks, which is optimized for a 60-day forecast during the summer season (May–September). They used historical admission and release data combined with external factors such as public and school holidays. They were able to achieve an MAE of 12.1 beds and an average mean absolute percentage error (MAPE) of 6.24.

1.3 Problem Definition

There are two main problems associated with a high bed occupancy, and they relate to resource management. The first one is ensuring that hospitals have the appropriate number and mix of staff to deal with surges and dips is crucial in cutting down costs. The most significant cost associated with surges is nursing overtime, which is 1.5X the hourly base pay in Ontario. By predicting surges ahead of time, hospitals can schedule in necessary medical personnel in advance to avoid overtime costs. Secondly, it is difficult to plan staff vacations due to uncertainty around patient influxes and needs. Knowing the bed occupancy ahead of time will allow staff to schedule vacations during a low volume period.

2. METHODOLOGY

2.1 Neural Networks

The (raw) data we obtained from KGH consists approximately 24 thousand timestamps of admissions and discharges from the Internal Medicine Unit. Each admission and discharge can be linked through a unique identifier as seen in figure 1.

Study ID	Resident_Service.NAME	ADMISSION_DATE_.TIME	DISCHARGE_DATE_TIME
458495	Internal Medicine B	04-01-2014 02:15	04-03-2014 16:30
1212197	Internal Medicine D	04-01-2014 04:21	04-17-2014 22:37
507244	Internal Medicine D	04-01-2014 04:30	04-03-2014 17:05
202030	Internal Medicine E	04-01-2014 04:30	04-04-2014 17:12
…	…	…	…
1186894	Internal Medicine C	03-31-2019 23:38	04-25-2019 12:28

Figure 1

We binned the data in 24-hour bins so that we could determine daily admissions and discharges. The range of dates comprises more than 4 years of information. The bed occupancy, y_n,at a given day, n,was computed as

$$y_n = y_{n-1} + a_n - d_n,$$

where a_n and d_n are the number of admissions and discharges on the same day, respectively.

For the Neural Networks model we included in the daily data, the day of the week, encoded as $\sin\frac{2\pi n}{6}$, where $n = 0$ is Sunday, 1, Monday and so forth. This is done so that the days of the week are represented by a continuous periodic variable and avoid the discontinuity from 6 to 0 (Saturday to Sunday). We also included maximum, minimum and average temperature and the "feels like" temperature obtained from the AccuWeather API. Additionally, we encoded the holiday calendar into the daily data with 1 for holiday (national or provincial) and 0 otherwise.

The preprocessing of this extended data entailed decomposing the bed occupancy values into slow and fast varying components. We chose to calculate the 3-day average -what we call the smooth component, and then subtracted this from the bed occupancy number, which is the fast component. The names are descriptive of the fact that taking the average of any fast-oscillating time-series such as the one in Fig. 2 will effectively remove local oscillations leaving a smooth average 'trend'. If we then subtract this smooth component from the original data only the fast oscillations will remain.

The Neural Network, a multilayer perceptron with 2 hidden layers trains and predicts independently the slow and fast components. The slow varying time series of the bed occupancy are fed to the neural network with a lag of 5 days, meaning that $\{y_{slow,n-i}\}$ with i from 1 to 5 are the predictors for $y_{slow,n}$. This is essentially an autocorrelation approach similar to an ARIMA model. The fast varying component is also fed with a lag but the extended values are also used, meaning that $\{y_{fast,n-i},\ extended\ data_{n-i}\}$ are the predictors for $y_{fast,n}$. The data was divided (chronologically) in 70% and 30% for training and testing respectively.

2.2 ARIMA

Naturally, bed occupancy records are presented as time-series datasets. As such, we chose to analyze these data using an autoregressive integrated moving average (ARIMA) model.

ARIMA models are a flexible and straightforward method for analyzing time-series data such as ours. The model takes three parameters p,d,q where p is the order of the AR (autoregressive) term or lag order, d is the differencing order, and q is the order of the MA (moving average) term respectively. The AR term refers to the fact that ARIMA models use previous observations (lags) to make predictions. To determine the lag order, we observe how many elements in the time series have a significant positive correlation, indicated by the number of points that lie above the significance limit line on a partial autocorrelation function plot. For our model, we chose a p-value of 5, and used a differencing order to transform time series data into stationary data. To determine the degree of differencing, we used an Augmented Dickey-Fuller test that assumes the time series is non-stationary as its null hypothesis. Therefore, given a p-value > 0.05, we accept the null hypothesis and determine the differencing order to be 1. The MA term refers to the number of lagged forecast errors to be included in the model. To determine how many MA terms to include, we use an autocorrelation function plot and observe which lag points lie above the line of significance. For our model, we used a q value of 0 [3].

3. RESULTS AND DISCUSSION

3.1 Neural Networks

Figure 2 below, displays the Neural Network forecast. The model presented to be 85% accurate for 93% of the data. The root mean squared error was 7.63, meaning that our error was equivalent to 7 beds, or around one nursing staff. Also, the MAXE, maximum error, was 20.78 beds. Out of the forecasted 455 days, the error is 20 beds, or around 3 nurses. The error is not significant, typically hospitals usually have around 7 nurses in each department.

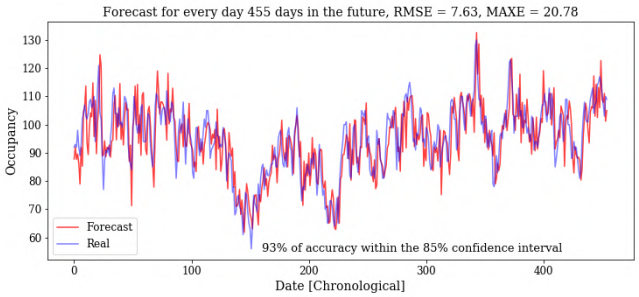


Figure 2

The model created allows KGH to forecast bed occupancy for the upcoming year. Predicting bed occupancy reduces expenses by lowering staff overtime. In addition, this will reduce wait times for the hospital's emergency room. The method proved to be exceptional, predicting the proceeding 455 days.

Future improvements to the model would include reducing error, comparing methods, changing parameters, and adding layers to the multilayer perceptron. Reducing the error consists of testing our model with different combinations of hidden layers and predictors to see which performs best.

3.2 ARIMA

With our final version of the ARIMA model, binning data daily, we were able to reach an MSE value of 33.249, a low value indicating our model is accurate. Unfortunately, the ARIMA model uses day-by-day predictions, meaning in order to predict the following day, the model needed to be trained and have the data of the previous day. Although this method allowed for a high accuracy level, it did not solve our issue, as the hospital was looking for a long-term prediction. We updated the model with weekly binning in order to make the model more useful for our situation. After running the model, it returned with an MSE value of 2407 as seen in figure 4, making it unusable for our problem.

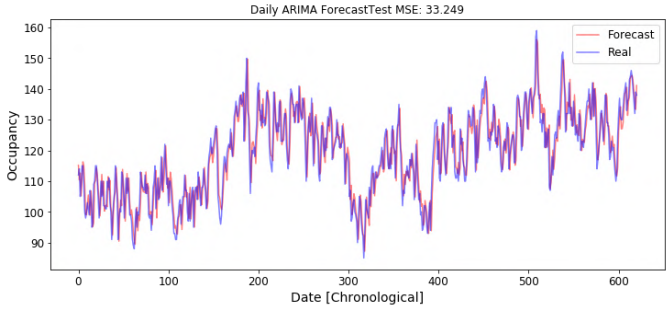


Figure 3

As for future iterations of this model, our goal would be to decrease the MSE. This can be done by increasing the lag value, and the computing power used.

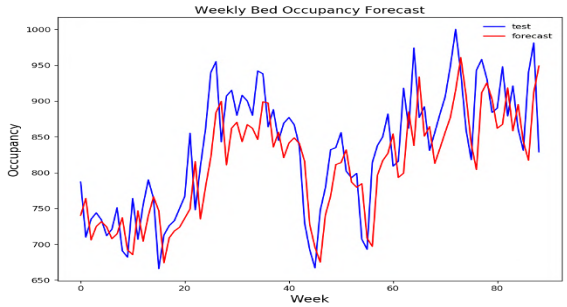


Figure 4

4. CONCLUSIONS AND FUTURE WORK

During this 8-month project, the team built ARIMA and Neural Network models to forecast bed occupancy at KGH. Using data to predict bed occupancy will assist hospitals in staffing more efficiently, reducing both expenses as well as wait times in emergency rooms

The next steps include improving the models by reducing error, changing parameters, adding layers to the multilayer perceptron, and understanding the impact of each predictor on the Neural Network. Ideally, the model we choose should accurately and consistently predict the occupancy for three months or more in advance. Forecasting occupancy for other hospital units would be valuable, especially, in volatile departments such as the ICU. The hospital can also observe more significant trends from the predictions allowing organization structure improvements.

To develop a robust model and allow for wide-scale adoption of these forecasting methods, acquiring data from other hospitals of a similar size and structure is necessary. This is so that we can test our model on different data sets, as well as understand the significance of different predictors. However, challenges imposed by the government regarding privacy for healthcare and factors such as management buy in, red-tape, and funding restrictions contribute to the slow implementation of AI in healthcare. To progress despite these restrictions, there needs to be a push from hospitals towards addressing operational inefficiencies using AI/ML.

REFERENCES

[1] Stephen Archer, "Ontario's Alternate Level of Care (ALC) Problem is Killing Acute Care Hospitals: Time to cut the Gordian knot". Department of Medicine Queen's, July 2016

[2] J.M. Sutherland, R. Trafford Crump, "Alternate level of Care: Canada's Hospital Beds, the Evidence and Options", PubMed, August 2013

[3] Robert Nau (2019). ARIMA models for time series forecasting. Statistical forecasting: notes on regression and time series analysis. Fuqua School of Business, Duke University

Machine Vision

FitVision

Alex Mason[1], Justin Paoli[2], Jaden Edhlund-Roy[3], Dean Sacoransky[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: 15ambm@queensu.ca
2 e-mail: 17jnp2@queensu.ca
3 e-mail: 17jer10@queensu.ca
4 e-mail: 17drs3@queensu.ca

Abstract: *FitVision is a mobile application that aims to help casual and experienced gym-goers refine and perfect their form for the squat. The application is designed to detect and estimate the users pose through a phone camera and classify their current pose as a squat or not. The app can count proper reps and replay the set to the user when they are finished. Through consistent use of the app users will develop clean and consistent squat form. The machine learning element of the project involves the use of the PoseNet pose estimation model. This allows us to track the users pose in real time and, using the data from the model, make a classification about whether they are performing a squat correctly or not. FitVision acts as a proof of concept for the use of pose estimation models in the field of physical fitness and exercise.*

---

1. INTRODUCTION

1.1 Motivation

The squat is a vitally important compound exercise. It involves bending the knees while carrying a weight on the shoulders. It is important for the development of the hip, knee and back. It is one of the most important part of training programs for weightlifters and powerlifters and is directly linked to the performance of lifters in competition [1]. While powerlifters and weightlifters have coaches and staff to support and guide them, many casual gym-going people have no access to such things. It is therefore difficult to improve and perfect squat form without proper instruction. Improperly performing squats can lead to permanent injury and a multitude of other problems. For this reason, the idea behind the FitVision app was born. The app will help gym users of all experience levels observe their squat form and receive visual feedback on the correctness of their form. It will also count reps and allow users to re-watch their latest sets.

1.2 Related Works

This problem has not been dealt with before using an app design like that laid out in this report. The most relevant material to this problem is advanced pose estimation models developed by various sources. FitVision settled on using PoseNet [2], a pose estimation model developed using TensorFlow.js, capable of accurately detecting the pose of a single user in a frame.

1.3 Problem Definition

Many casual gym-goers struggle to learn proper squat form due to a lack of instruction. Using online resources has limited effectiveness as there is no substitute for squatting in a gym. This can lead to frustration and worse, injury. Many gym-goers avoid squats altogether because they are intimidated and unsure of proper squat form. This represents a market for developing a solution to help casual gym-goers improve and perfect their squat form in an intuitive and easily understandable way.

2. METHODOLOGY

2.1 Choosing a Model

The design of the app was divided into a few distinct phases. Before beginning development, the team needed to choose a pose estimation model. The pose estimation model is the foundation of the app, and therefore we wanted to choose the one best suited to our app. The decision was narrowed down to PoseNet and OpenPose. Ultimately, we decided on PoseNet because it was developed in JavaScript, a language more familiar to the team and better suited for app design.

2.2 Learning the Model

Next, we needed to learn how to use and interact with the model. This means learning how to give it input and how to process the models output. The PoseNet model takes as input an image and produces a json object with the model key points. The key points include, for example, the left and right shoulder, elbow, wrist, hip, knee and foot. Each key point has two properties. First is the pixel location of the key point on the image. The second is the confidence score of the key point. In other words, how confident the model is that the key point is at that particular location. If, for example, the user's wrist was not in the image, the confidence score might be 0.05. If the user was clearly visible against a high contrast background and all their limbs were distinct, the confidence score might by 0.95.

Once we learned the model we began testing with static images. This meant feeding various images of a squat, mid-repetition, to the model and observing its output. From testing we discerned what angles were most important and generally the limitations of the model. One of our main discoveries was that the model works best when viewing the user from the front. This produced the most stable and consistent pose.

2.3 Building the Classifier

The next phase of development was building the classifier and getting it to work consistently on static images. Before working on a video feed, it was important to ensure that the classifier was working on unmoving images. The benefit of the squat in this application is that there is a distinct point where the user correctly "completes" the squat. That is, when the user reaches the bottom of a squat, assuming the proper depth, we can say the squat has been completed successfully. We did not consider the case where the user is unable to standup, as this is obviously not a complete squat.

The development of the classifier was a lot of trial and error. Fundamental squat principles dictate that depth should be *at least* 90 degrees, or parallel. That is, the angle formed at the knee should be approximately 90 degrees or less when at full depth. The classifier was continually adjusted until it successfully classified at least 10 different images of squats, with each image depicting a correct squat. It was also applied to improper squats to ensure false positives were avoided.

After developing the static-image classifier we turned our attention to developing the video classifier. This was achieved by passing frames from the phones video feed to a canvas object that could then be passed to the PoseNet model. At every frame of the video the PoseNet model would estimate the pose of the user, use the squat classifier, and indicate on the screen whether a squat was detected. We visualized this by changing the pose color to green when a squat is detected but to blue otherwise.

3. RESULTS AND DISCUSSION

The fundamental task of the app was to detect and classify a user's squat. This task was achieved however the app is not exceedingly robust. The app is susceptible to poor lighting, bad angles, and multiple users in the frame. These are faults inherent in the PoseNet model, which suggests that the app could be made more robust if a different model was used. With this being said, we were able to classify squats with relative consistency. Other features include the ability to count successful reps and replay the last set to the user so they can observe their squats. Screen captures of the app in action can be seen in Figure 1 and Figure 2. In these figures we can see that when the user is at a standing position the pose "skeleton" is drawn in blue. When the user reaches the correct squat position their pose becomes green, indicating the successful completion of a single squat. This functionality allows the user to confirm when they perform a proper squat and the replay feature allows them to review their most recent set and observe any inconsistencies or problem areas of their squat form.

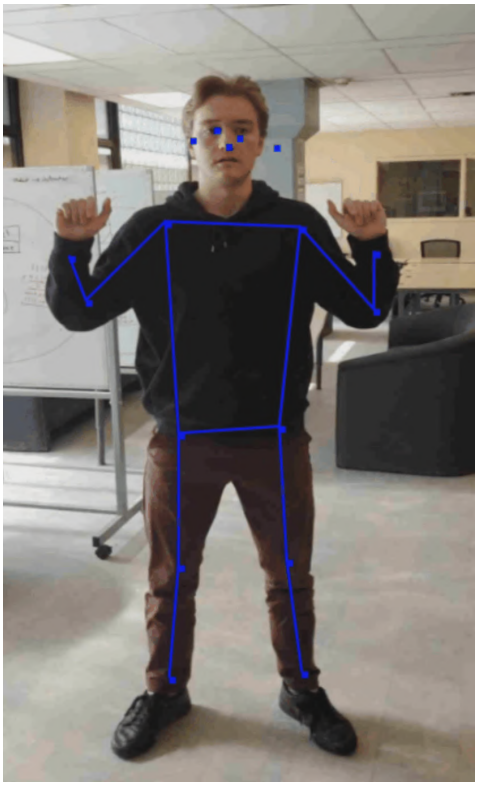


Figure 1: No Squat detected, blue pose

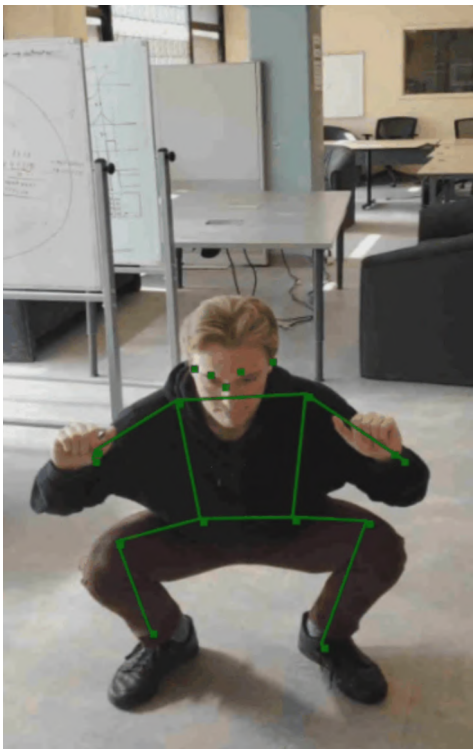


Figure 2: Squat dected, green pose

4. CONCLUSIONS AND FUTURE WORK

The project was successful in achieving its intended purpose. It can be said, however, that there are many possible extensions of the app and, given more time, could be developed. The most obvious extension includes adding additional exercises. More exercises would give the app a broader purpose and allow users to work on perfecting their form for exercises other than just the squat. Other common but important exercises that might be added include the Deadlift, Bench Press, Clean, and Lunge, among others. The app could have also benefitted from a prettier user interface; however, the effect of this change would be purely visual and offer no real functional improvements. Lastly, as mentioned briefly in results and discussion, the app could benefit from a more accurate and robust pose estimation model. This might allow for more detailed feedback on the users squats such as the angle of their back, how their weight was distributed through their feet, how fast or slow their squat was and many more minute but valuable details.

Ultimately, the app does not present a comprehensive solution to learning to squat. It does not replace an experienced coach or trainer. It does, however, offer a proof of concept look at the benefits of applying pose estimation models to physical fitness and health problems. With a free, open-source model found online the team was able to create a useful tool for helping casual gym-goers learn and improve their squat form.

REFERENCES

[1] P. WRETENBERG, Y. FENG and U. P. ARBORELIUS, "High- and low-bar squatting techniques during weight-training," *Medicine and Science in Sports & Exercise,* 1996.

[2] TensorFlow, "Pose Detection in the Browser: PoseNet Model," 2019. [Online]. Available: https://github.com/tensorflow/tfjs-models/tree/master/posenet.

Formula 1 Analytics

John David Anthony[1], Eric Yuyitung[2], Connor Sparling[3], Brendon Cop[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: johndavid.anthony1@gmail.com
2 e-mail: 17edy@queensu.ca
3 e-mail: connor.sparling@queensu.ca
4 e-mail: 17bsc@queensu.ca

Abstract: *Broadcasting companies are continually looking for ways to increase viewer engagement. Can a broadcast be tailored to the wants of viewers based on observing and comparing viewer engagement levels? To answer this question, we created 3 classification models to tag footage of the 2019 Formula 1 racing season. The race was tagged to determine if the footage was a replay, what camera angle is currently shown, and what teams are on screen. We were able to tag replays with 100% accuracy; however, unbalanced data presented an issue for the camera angle and team on screen classifier. We were able to classify teams with a sufficient amount of data correctly 76% of the time.*

1. INTRODUCTION

1.1 Motivation

In the age of video streaming services such as Netflix, viewers can choose to watch content at a time that is suitable for them. Sporting events, however, are mainly viewed live and because of their long duration, maintaining the viewer's attention is a challenge.

The value of sports broadcasting rights continues to increase year after year. As a result, broadcasters are looking for ways to increase both viewership and engagement from casual and hardcore fans. By increasing fan engagement, broadcasters can attract new viewers and gain the opportunity to display advertisements to further strengthen their bottom line.

The question that now remains is how can sports companies alter their broadcasts to make it more exciting for the viewer? In order to make a sports broadcast more engaging, the broadcaster needs to know what the viewer wants to see.

1.2 Related Works

To increase the engagement of the viewer, there needs to be a way to measure their engagement. Instead of eye trackers, J. Hernandez et al. [1] used RGB cameras to predict the viewer's engagement level. However, challenges arise when the viewer's engagement could not be fully determined from the image captured from the camera. The viewer may be looking away from the sports broadcast and discussing the sport with a friend. In this scenario the system would predict this as a low level of engagement.

Pattern matching is a technique used to identify if a template is present in an image. The work of Y. Hel-Or et al. [2] explores how to accomplish this task while maintaining efficiency and effectiveness.

1.3 Problem Definition

We have decided to analyze footage from the 2019 Formula 1 season to determine what impacts viewer engagement. Deloitte Canada has provided their EmotionPlus platform that predicts facial sentiment from a standard camera. In addition, Deloitte has provided raw data of over 40 viewers. The provided

data included the level of Valence for the viewers which is the viewer's general happiness while watching the race. The data also includes the viewer's level of engagement. This is calculated through eye and face movement as well as their expressed emotions.

To determine what effects the viewer's engagement level, we first need to classify the frames of the race. The first property to be tracked is if a frame is a replay or not. Secondly, what camera angle is currently being used. Finally, what teams are on screen.

Three classifiers need to be created that will take a frame from a race as input and will output their predicted label.

These labels will then be used against the emotion data previously collected to identify patterns and insights.

2. METHODOLOGY

To begin with, the data needed to be manually tagged to train the classifiers. Three different races were filmed at 50 frames per second. 1 frame was chosen every 50 frames to be manually tagged, as we did not want to create the world's largest repository of Formula 1 pictures. The individual frames were then tagged with its replay status, camera angle, and the teams on screen in bounding boxes.

Figure 1: Replay Template

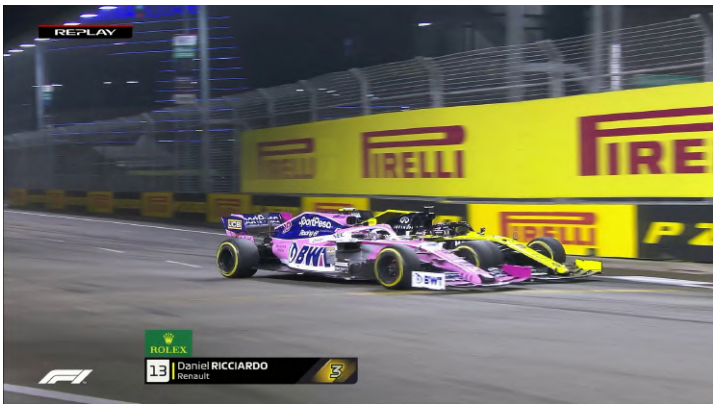


Figure 2: Replay Frame

In Formula 1 broadcasts, there is a replay flag present in every replay sequence. This is incredibly helpful as it reduces the magnitude of the first classifier. By using the pattern matching method explained previously, the template (Fig. 1) is transferred across the frame (Fig. 2) seeking the best match. If a sufficient match is found, the frame is tagged as a replay.

The camera angle classifier uses a pre-trained neural network Inception-V3 (IV3). IV3 is a convolutional neural network used for analyzing images. The last 3 layers of the network were retrained with the tagged data from the races. The camera angles were divided into 5 categories: driver view, pit view, spectator view, track view, and other view. The majority of frames fell within the track view.

The team on screen classifier uses a simplified variant of YOLOV3 – a pre-trained object detection network. Originally, YOLOV3 was planned to generate cropped images of all cars in the frame. The images would then be fed through a function to generate their colour histogram. This histogram would then be used to predict the car's team based on their colour. This was not needed as YOLOV3 displayed exceptional promise in differentiating cars based on their colour and decals.

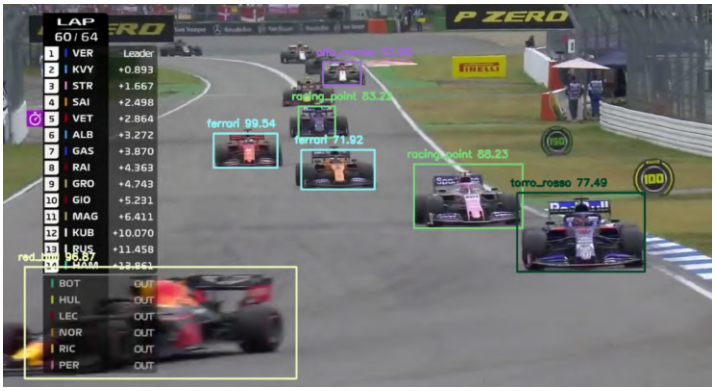


Figure 3: Tagged Teams

3. RESULTS AND DISCUSSION

The replay classifier was able to achieve 99.99% accuracy as a result of the trivial nature of the problem. The classifier's only misclassifications arose when there was a crowd in the area where the replay flag usually is. This caused sporadic replay classifications for only 1 frame. As there will never be a 1 frame replay, this classification can be ignored bringing the total accuracy to 100%.

The camera angle classifier was not able to get accuracy as high as the replay classifier however it was

a more difficult problem. In the current state, the classifier is biased towards predicting a track view. This is the result of unbalanced data, as the majority of the race is viewed on the track.
The team on screen classifier had similar issues to the camera angle classifier. The training and testing sets were unbalanced. When only including the 4 teams with the most data (Ferrari, Racing Point, Renault, and Mercedes) the average precision was 76%.

Additionally, COVID-19 had a negative impact on the ability for the team to continue development as well as we hoped.

4. CONCLUSIONS AND FUTURE WORK

The work completed thus far shows that broadcast footage can be tagged based on various features.

Moving forward, to improve the impact of the system we would like to expand the replay tagging system. Throughout the race, various events occur on the screen such as the fastest lap, warning flags, the team on the radio, etc. These events have an image on the screen similar to the replay flag, thus we can detect these events occurring and further analyze their impact on the viewer's engagement.

Secondly, further work needs to be done on the camera angle classifier and team on screen classifier. We will use multiple strategies to tackle the issue of unbalanced data. By oversampling the minority classes, we allow the network to learn more about the imbalanced classes. Additionally, cost sensitive learning is an effective way to train the network where the cost of classification mistakes is higher on the minority classes.

Finally, we believe that more emotional data would allow for more in-depth insight into viewer engagement. Especially emotional insight for geographically separated viewers. This would allow for additional awareness into what engages different regions and create the opportunity for more tailored broadcasts.

REFERENCES

[1] J. Hernandez, Zicheng Liu, G. Hulten, D. DeBarr, K. Krum and Z. Zhang, "Measuring the engagement level of TV viewers," 2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), Shanghai, 2013, pp. 1-7.

[2] Y. Hel-Or and H. Hel-Or, "Real-time pattern matching using projection kernels," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 9, pp. 1430-1445, Sept. 2005.

Google Street View Risk Assessment

Matt Wright[1], Adib Chisty[2], Jonah Gramaise[3], Bhavan Suthakaran[4], Sanil Andapally[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: matt.wright@queensu.ca
2 e-mail: adibchisty@gmail.com
3 e-mail: jonahgramaise@gmail.com
4 e-mail: bhavahnsuthakaran@gmail.com
5 e-mail: saniladapally@gmail.com

Abstract: *The goal of this project is to apply machine learning solutions to automate the expensive and time-consuming task of assessing individual property risk of houses. Using theft crime rates, robbery crime rates, and fire frequencies in conjunction with Google Street Views of houses, we experimented with encoding then standardizing the datasets. Using k-means to cluster the data, the result indicated regions of risk. The fire, theft, and robbery rates effectively correlated towards assessing risk, while the effect of the image data values on the final clusters was less clear. Moving forward, we would like to refine our image processing so that we can derive more meaningful information from our images. In addition, we want to find more personal data to differentiate between neighbouring houses.*

---

1. INTRODUCTION

1.1 Motivation

Currently, real estate risk assessment is performed by insurance company agents contacting and booking a time to meet the property owner to inspect the property in person. The gathered information is then used in a proprietary formula created by the insurance company to calculate the risk associated with insuring the house. Visiting individual properties can be extremely inefficient and costly across a large customer base especially when property owners avoid insurance agents.

Our goal is to develop an agile machine learning solution in which a preliminary risk assessment is to be completed based on public data to automate the in-person inspection phase. To test this hypothesis, the city of Toronto was chosen as a testbench.

1.2 Related Works

Unsupervised image classification has been explored by various institutes before. Research out of the University of Wollongong describes feature extraction as an important step in image modelling for classification [1]. Further, autoencoders can be used for either feature extraction or dimensionality reduction when analyzing images [2]. A similar project was explored by university students from Warsaw and Stanford in which they predicted car accident risk for a resident based off Google Street view image of the individual's house [3].

1.3 Problem Definition

The formulae insurance firms use to calculate their risk assessments is the intellectual property of the company and is not publicly available. However, it is widely known that when calculating real estate premiums crime rates, proximity to fire stations, police stations, and neighborhood information are all considered in addition to the state of the house.

This project will explore the viability of using Toronto's open data on crime information and google street view imagery to distinguish between houses and place them in a risk category using unsupervised learning.

2. METHODOLOGY

We found a list of all residential addresses in Toronto and then collected any useful data that has a corresponding address, postal code, longitude and latitude, or GEOID. We found open data sets for break and entering counts (BNE), theft count, robbery count, and fire count. In addition, we collected Google Street Views of each house.

We concentrated on image processing and machine vision as we tried to extract information form the Street Views. We experimented with colour shifting, applying the Sobel operator, histogram of oriented gradients, and autoencoders for feature extraction and dimensionality reduction. An example of the output of the Sobel operator is shown in Figure 1.

Figure 1: An original Street View (left) and the result of resizing, gray-shifting, and applying the Sobel operator (right).

The transformed images were then encoded with an autoencoder that has three convolutional layers (each followed by a max pooling layer), the output of which is flattened and fed into two dense layers, outputting a single value. The autoencoded value, along with the numeric crime data were all standardized for clustering via k-means.

To determine the optimal number of clusters, we used the elbow method. Given the results shown in Figure 2, we concluded that 5 clusters is the optimal number of clusters for our data set.

3. RESULTS AND DISCUSSION

Our model was able to distinguish houses into different classes based on our data. The sampled houses were divided into five classes, each class representing a different level of what we interpreted as house risk. Table 1 shows the numeric results of clustering.

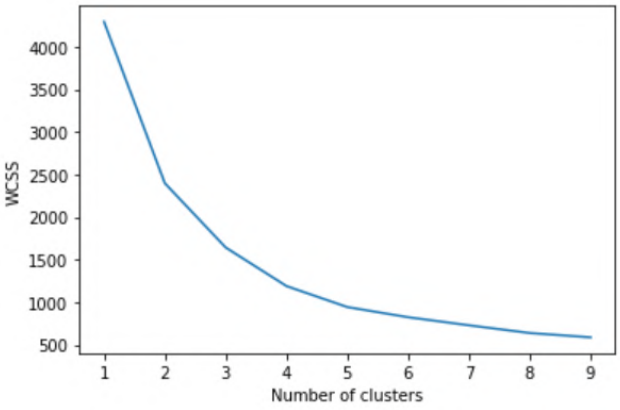


Figure 2: Results of K-means clustering elbow curve.

Table 1: Average feature value for each class (classes 0 to 4)

Feature	*C-0*	*C-1*	*C-2*	*C-3*	*C-4*
BNE	62	431	807	59	162
Theft	9	65	149	8	19
Robbery	31	120	475	24	64
Fire	57	212	422	47	129
Image	3.6	1.5	0.6	1.7	2.2

For instance, Class 2 corresponds to high levels of crime and fire while Class 3 corresponds to lower levels, implying that Class 2 represents houses with a higher risk then Class 3. A visualization of the values corresponding to each class are shown in Figure 3.

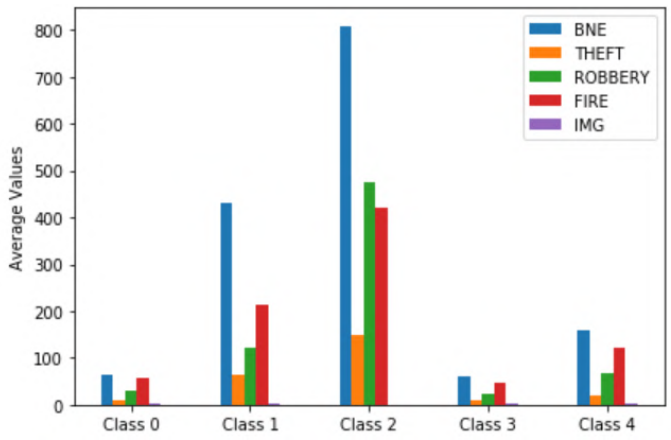


Figure 3: Visualization of the resulting risk classes.

The visual representation of the principal component in Figure 4 reveals that the individual points in each cluster are similar and that the centroids are accurate.

While the clustering of crime rates and instances of fire seem accurate, the physical representation and significance of the auto-encoded images is not as clear. For in instance, Class 0 and Class 3 (both considered

lower risk) have relatively high image values of 3.6 and 1.7, respectively while Class 2 (a higher risk class) has a small value of 0.6. We inferred that a higher image value implies a lower risk, but it is still unclear.

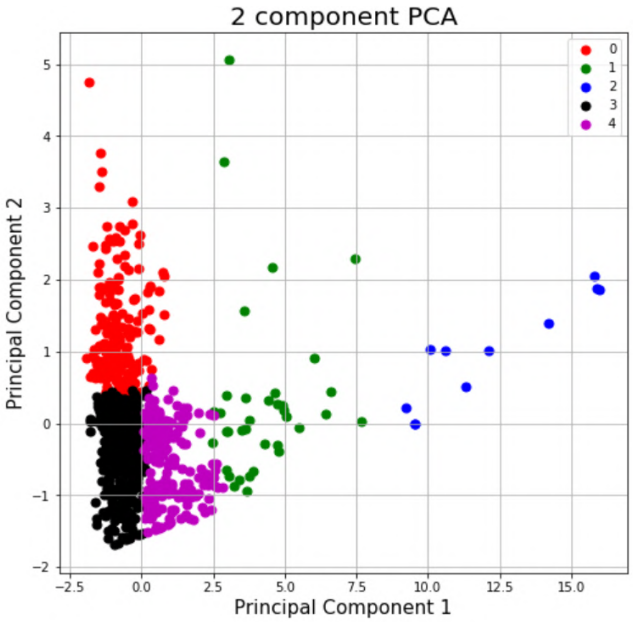


Figure 4: Two dimension representation of the data using principle component analysis, color-coded by clusters.

It is hard to quantify the performance of an unsupervised model. However, given the similarities within (and the differences between) each cluster we concluded that the model classifies houses with an acceptable level of certainty.

The auto-encoder performance was quite poor, especially on the images after being operated on by the Sobel kernel and as a result it was not used as a feature. This was expected considering we tried to condense a 240x256 photo into a single value.

4. CONCLUSIONS AND FUTURE WORK

The motive behind this project was to identify risks associated with insuring a house using image processing and unsupervised learning to analyze Google Street Views and neighbourhood statistics. Techniques such as colour shifting, histogram of gradients, applying the Sobel operator, and auto-encoding were used in the pre-processing stage for feature extraction. Moving forward, the data was standardized to relate image features to numerical data sets. Based on the optimal number of clusters for a k-means classification obtained using the elbow method, the combined data were separated into classes of varying risks.

The existing model could be further refined by expanding the feature detection techniques. The team would explore algorithms such as Scale Invariant Feature Transform, Speed Up Robust Feature as well as image segmentation. Revising the image processing would allow for the unsupervised model to distinguish and develop more precise clusters leading to more meaningful outputs. The current clustering mechanism will also need to be further revised by weighting the features before clustering to prioritize the most influential features.

In order to improve the precision of the model, we will need to analyze features specific to each address so our model can confidently differentiate between neighboring houses. In order to accomplish this, the team will need to access data such age of house, average income, family status, structural integrity, revision of electrical/mechanical supports in the house etc. With a clearer understanding of risk analysis, reinforcement learning could be a possible next step to improve the confidence of our model.

Ideally, a fully functional model would assign risk for smaller regions or individual houses within a neighbourhood in a more user-friendly manner. A possible output could portray a heat map based on risk varying from 0-100. Additionally, the model would span over regions across Canada, compared to the current model that focuses on neighbourhoods within the Greater Toronto Area. The challenges associated with the future developments include implementing more relevant and specific data sets and training a model that would have a simple visual output.

REFERENCES

[1] A. Olaode, G. Naghdy, C. Todd, "Unsupervised Classification of Images: A Review". International Journal of Image Processing. 8. 2014-325.

[2] S. Mabu, K. Kobayashi, M. Obayashi, T. Kuremoto, "Unsupervised Image Classification Using Multi-Autoencoder and K-means++", Journal of Robotics, Networking, and Artificial Life. 5. 2018-75

[3] K. Kita-Wojciechowska, L. Kidzinski, "Google Street View image of a house predicts car accident risk of its resident", Department of Bioengineering, Stanford University. 2019

HelpingHand: The ASL Translator

Colin Cumming[1], Dennis Huynh[2], Griffin Clark[3], Alexia Quinn[4], Nicolas Wills[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: 16cmc21@queensu.ca
2 e-mail: 16dh24@queensu.ca
3 e-mail: 17gsc2@queensu.ca
4 e-mail: 18amq@queensu.ca
5 e-mail: 17nvw@queensu.ca

Abstract: *We began this project with the intention of bridging the communication gap between the hearing impaired that require on American Sign Language (ASL) for communication and those that do not know ASL. There is a problem in society where people are uneducated in ASL, even though it is estimated 500,000 people natively speak in ASL* [1]. *The project hoped to increase the accessibility of the hearing impaired and improve inclusivity in day-to-day life. To solve the problem, a convolutional neural network was designed to translate ASL alphabet signs in real time and display the translated output to the user via a web application. We were successful in developing a model which was 93% accurate in a controlled environment but performed at 64% accuracy in real time.*

1. INTRODUCTION

1.1 Motivation

There are currently over 500,000 people in the United States and Canada who use American Sign Language as their natural language [1]. This is only a portion of the one million adults in Canada who have a hearing-related disability, a number which is projected to increase by over 20% in the next few years [2]. There is an abundance of programs translating text to ASL; however, resources in translating hand signs are limited. Everyone deserves to be able to communicate with others, regardless of the presence of a hearing disability. According to Communication Services for the Deaf, 72% of deaf children's families do not use sign language and 98% of people who are deaf do not receive education in sign language [3]. The gap in communication between users and non-users of ASL needs to be addressed in a way that is easy to use and accessible to the multitude of people who want to communicate with users of ASL.

1.2 Related Works

There have been several projects that have attempted to make a sign language translator. First, Google has developed AI software that tracks the shape and motions of hands. The developers, Valentin Bazarevsky and Fan Zhang, say that the freely published hand-tracking software serves as the basis for sign language detection. However, it will be up to the developers to create an application that does sign language translation. Another invention is haptic gloves that translates sign language into an Android application which then reads the text aloud [4]. Similarly, DeepASL, a glove-less technology developed by a team from Michigan State University, translates hand signs with a camera, then sends that video feed through a deep learning algorithm to identify the word, and finally, it constructs a sentence. Both of these inventions translate hand gestures to words, which leads to the problem of expression gained from facial expressions and speed of signing. Also, translating words or sentences with these methods would not include the sign language vernacular. Finally, a Hungarian company called SignAll has developed full translation through the use

of cameras that considers body language, facial features, speed, and hand gestures per person with the use of computer vision and AI [5]. However, it only translates to text that displays on a screen.

1.3 Problem Definition

For ASL to be implemented into a conversation, both the speaker and the listener require knowledge of the language. This leaves many Americans and Canadians unable to effectively communicate with people that have hearing loss. In order to bridge the communication gap, we set out to create an ASL real time translator. For something like this to be effective in day to day use it needs to be portable, easy to use and be able to translate sign language quickly and accurately. We are not the first team to come up with the idea of making ASL more accessible. The multitude of projects that are related to this field gave us a good understanding of what would be possible in the time period allotted. The published work also allowed us to do valuable research to choose the type of neural network that would work best for our project.

As mentioned in *Section 1.2*, Google recently developed a software to track the movement of hands in a live video. After looking into this feat of engineering, we decided that using this level of AI would become too complex for our project. DeepASL, a Michigan State research project is another amazing example of communication barriers being broken. The team used a HB-RNN as well as a probabilistic framework to create single word as well as sentence translation. Although this was much deeper dive into the problem than what we were planning, the published paper gave us a step-by-step demonstration of how to go about our problem. After going through all our research, we decided that with the time given, translating single letters and numbers would be the best course of action.

2. METHODOLOGY

2.1 Dataset generation

The beginning of the project began with sourcing a dataset. This dataset needed to be extensive enough for the model to be able to determine signs in multiple lighting conditions and hands at different angles. A dataset was procured from an online dataset repository website Kaggle [6]. This dataset consisted of 87,000 images of the ASL alphabet. This included the 26 letters of the alphabet and three additional signs for *space, delete,* and *nothing*. This provided 3,000 images of varied lighting and angles for the model to be trained on.

The images had to be preprocessed using OpenCV and Numpy before they could be provided as input to the model. The preprocessing involved converting the images to greyscale (rational described in the following paragraphs) as well as resizing them to improve the speed of the training. The images are paired with a label that identifies the corresponding ASL alphabet sign to the image which is represented as a one-hot encoded value signifying a specific letter of the alphabet. This label is coupled with the Numpy matrix data that represents the image. These values together create data that is readable by the model allowing it to be trained on the image data.

2.2 Model Creation

The proposed solution involved creating a convolutional neural network (CNN) that was capable of translating ASL to text and displayed to the user. The CNN was developed using Keras, a machine learning library build as an additional interface to the well-known library TensorFlow. Keras was chosen as the machine learning library of choice due to its ease of use and power. We concluded on using a CNN for the solution as they work well with unstructured data and are commonly used for image classification purposes.

After the model was trained on the data, it was evaluated on a separate testing set of data. This data was prepared in the same way as the training data but was excluded from the training set. Results from the evaluation can be found in section *3. Results and Discussion*.

2.3 Additional Analysis

To improve the accuracy of the model, additional procedures were considered. A recurrent neural network (RNN) was created to perform background subtraction on the images. This was never tested to see if the background subtracted images created an improved model. Multiple pre-processing techniques were applied on the images including Gaussian blur filters, edge enhancing, and greyscale. Minimal accuracy difference was observed, and the accuracy

differences were not documented. The design team applied a greyscale filter in the final model as it reduced the colour dimension of the image, improving training times. Feature engineering and modification of hyper-parameters were applied in an iterative fashion to create the model.

3. RESULTS AND DISCUSSION

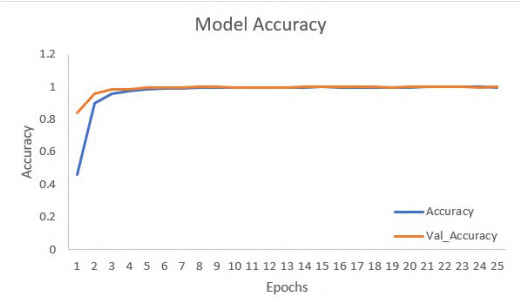


Figure 1 - Graph depicting the model accuracy over time represented in epochs, or rounds trained.

Figure 1 shows the model's accuracy after a given number of epochs. The accuracy plateaus at 93% accuracy. Despite this value, practical accuracy was much lower when the user inputted data had very noisy backgrounds. From this, we learned that the image preprocessing technique used must work well in very noisy environments, which was not considered during the development phase of the project.

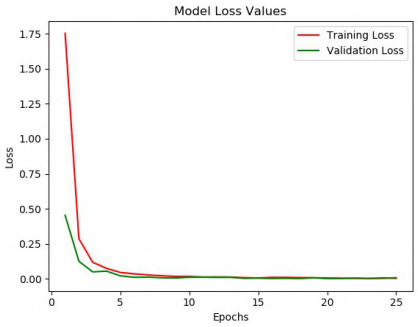


Figure 2 - Graph showcasing the trend of the mean loss values over time represented in epochs, or rounds trained.

Figure 2 shows that the sum of the errors between the train and test guesses plateaus at zero as the number of epochs increases. Ideally, the loss would exhibit a more gradual decline towards zero. This figure is indicative of overfitting which may have contributed to the lower practical accuracy.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 50, 50, 32)	320
conv2d_2 (Conv2D)	(None, 50, 50, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 25, 25, 64)	0
conv2d_3 (Conv2D)	(None, 25, 25, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
conv2d_4 (Conv2D)	(None, 12, 12, 64)	73792
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 6, 6, 32)	18464
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 32)	0
dense_1 (Dense)	(None, 3, 3, 1024)	33792
dropout_1 (Dropout)	(None, 3, 3, 1024)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_2 (Dense)	(None, 26)	239642

Total params: 458,362
Trainable params: 458,362
Non-trainable params: 0

Figure 3 - Document showing the model structure and parameters.

Finally, Figure 3 shows the mode layers that were used. The model alternated between convolutions and max pooling layers with a dropout layer to prevent overfitting. This network design was decided upon after rounds of feature engineering and model retraining. This network structure produced the best results. There is then a final dense output layer to create a one-dimensional array which represents the model's guess for the letter of the alphabet.

4. CONCLUSIONS AND FUTURE WORK

In the future, we hope to expand the database to include numbers, words, and the entire alphabet for HelpingHand as it can only translate 24 letters (it did not include J or Z because they required movement). Another feature that would be implemented is ensuring that images of signs can be translated accurately in any environment as the current model can only translate when there is no noise in the background. Finally, we hope to be able to translate video and in real-time through a mobile application for Android and iOS to make it more convenient than accessing HelpingHand through a web application.

REFERENCES

[1] Michelle Jay, "American Sign Language | Start ASL," *Start ASL*, 20-Mar-2020. (accessed Mar. 25, 2020).

[2] "Facts and figures," *Canadian Hearing Services*, 18-Nov-2013. (accessed Mar. 25, 2020).

[3] Sophia Waterfield, "Today is #ASLDay, but how is it used In today's society?," *Newsweek*, 15-Apr-2019. (accessed Mar. 25, 2020).

[4] Emily Matchar, "Sign Language Translating Devices Are Cool. But Are They Useful? | Innovation | Smithsonian Magazine," *Smithsonian Magazine*, 26-Feb-2019. (accessed Mar. 25, 2020).

[5] Devin Coldewey, "SignAll is slowly but surely building a sign language translation platform," *TechCrunch*, 14-Feb-2018. (accessed Mar. 25, 2020).

[6] Akash Nagaraj, "ASL Alphabet," 22-Apr-2018.

Plant Identification

Travis Cossarini[1], Luka Antonyshyn[2], Stephen Obadinma[3], Emma Landry[4], Hannah Lacey[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: travis.cossarini@queensu.ca
2 e-mail: 15la@queensu.ca
3 e-mail: 16sco@queensu.ca
4 e-mail: emma.landry@queensu.ca
5 e-mail: hannah.lacey@queensu.ca

Abstract: *Our group worked on the recognition of tree species based on a snapshot of a single leaf. The dataset that our model was trained on was the Flavia Dataset, containing 32 classes. In order to accomplish this task, we started with custom CNNs using the Keras Python library. We then transitioned into transfer learning using the VGG16 architecture to attain higher accuracy, which reached 92.7%. All models were trained using a virtual machine on Microsoft Azure, to ease the computational burden of training large models.*

---

1. INTRODUCTION

1.1 Motivation

With the proliferation of cameras throughout society, image recognition technologies are being deployed on an increasingly large scale. The growing prevalence of object and facial recognition presents an opportunity for an increase in the quality of everyday life.

1.2 Related Works

This problem has been solved through a variety of approaches, ranging from probabilistic neural nets (PNN) [3] to convolutional neural nets (CCN). The majority of high accuracy projects have been done using CNNs due to their excellent accuracy in image recognition. However, the vast majority of projects do not consider a large number of classes, making their findings less useful in a practical setting.

1.3 Problem Definition

Field biology is a demanding subject as it requires biologists to know numerous species of plants by memory or carry around large books detailing the various species.

This was a problem that our group was looking to solve with this project. By developing a flexible web-based application, biologists would be able to identify a wide variety of species without the need to memorize characteristics or carry around hefty encyclopedias. This in turn would allow them to work more efficiently.

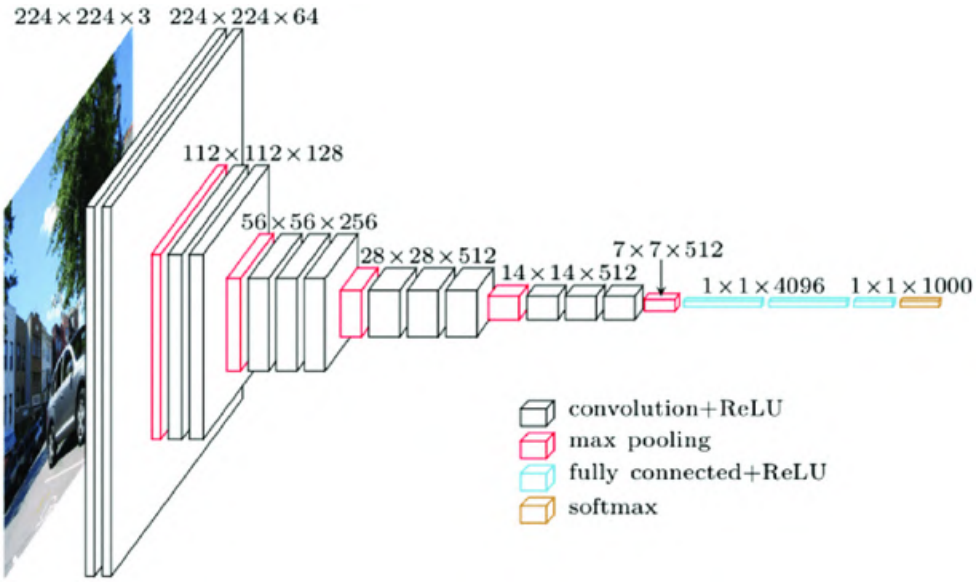


Figure 1: VGG 16 Architecture [2]

2. METHODOLOGY

The Flavia Dataset contains 1907 snapshots of leaves belonging to 32 classes (species) of trees.

Figure 2: Canadian Poplar from Flavia Dataset

Several different methods were used, including custom CNN's and transfer learning. Initially, custom CNNs were constructed using Keras. The most sophisticated of which utilized a nine-layer architecture with rectified linear unit activation. Strides were used to lower the computational intensity. However, due to the complexity of the images, these simple architectures were not intricate enough to achieve any useful accuracy.

We next moved to the topic of transfer learning. Whereby, a pretrained CNN is partially reweighted on a specific dataset, in our case the Flavia Dataset. The VGG16 architecture was chosen as our starting point and can be seen in Figure 1.

800 images were selected for training, 600 images were selected for validation and 307 images were selected for testing. The images were sheared, rotated, and zoomed to expand the data set.
The models were trained using a categorical cross-entropy error metric and a stochastic gradient descent optimizer. An initial learning rate of 0.0001 was used, with learning rate decay. The model was trained for 50 epochs, with 22 steps per epoch.

Due to the large number of classes, the computing power necessary to train a model complex enough to have high accuracy on the test dataset had to come from a cloud computing resource. The resource chosen was Microsoft's Azure's Machine Learning resource. All of the models were trained using this resource. Initially, the dataset was uploaded to Microsoft Azure's blog storage utility where it could easily be accessed during training and a workspace was created.

The process that was followed to train a model first involved creating a Keras script that handled training. This was followed by creating a remote compute target that the training script would be run on. Then a Jupyter notebook was made that ran code which connected to and created all of the required resources on Azure, like an estimator and an experiment, and essentially made the Keras training script run on the compute target from earlier, which trained and validated a specific model. The resulting model was then obtained from storage and used for testing.

3. RESULTS AND DISCUSSION

After training and testing several different implementations, a VGG16 architecture was trained for our image classification task. The VGG16 network has achieved 92.7% top-5 accuracy on ImageNet [2]. A training accuracy of 95%, a validation accuracy of 93%, and a test accuracy of 88% were observed for the trained model. While relatively high accuracy was achieved with the model, training data was limited and unvaried, as all of the images were taken in a laboratory setting.

Below is a graph of the accuracy of a custom nine-layer CNN constructed using Keras. The accuracy can be seen to cap off at 80%, which is too low for practical applications.

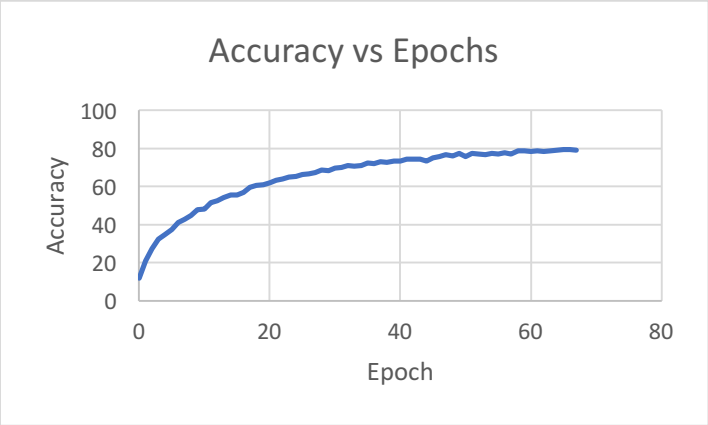


Figure 3: Accuracy vs Epochs of a Basic CNN

4. CONCLUSIONS AND FUTURE WORK

As states in the previous section, through employing the VGG16 architecture and transfer learning, we were able to achieve 92.7% accuracy. This is acceptable for applications in the field. However, this accuracy could be increased through a more rigorous training algorithm. By increasing the available computation resources, we would be able to use more complex architectures than the VGG16, allowing for more accurate predictions.

Additionally, we have developed an application through Android Studio, which will allow for agile deployment in the field. The interface between the app and CNN was not complete at the time of the conference but is a focus moving forward. This app will allow biologists and aspiring botanists to use our app for quick and easy identification of tree species as we work to expand the dataset that our model functions on.

Additionally, a move towards realistic staging of photos is necessary as well. The Flavia dataset contains only lab photos with a solid white background, which does not occur in the field. For the app to be of real use, the model will need to be able to function on a variety of backgrounds.

Figure 4: Splash Screen from the Application, on a Pixel 4

REFERENCES

[1] O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints. M. Vrigkas, C, Nikou, and I. Kakadiaris, ”A Review of HumanActivity Recognition Methods”, Front. Robot. AI, 16 November 2015.

[2] Zhang, et al. “Accelerating Very Deep Convolutional Networks for Classification and Detection.” ArXiv.org, 18 Nov. 2015, arxiv.org/abs/1505.06798.

[3] “Flavia, A Leaf Recognition Algorithm for Plant Classification Using PNN.” Flavia, A Leaf Recognition Algorithm for Plant Classification Using PNN, flavia.sourceforge.net/.

Sonic Reinforcement Learning

Ted Munn[1], Simon Dudtschak[2], Chen Sun[3], Jessica Li[4], Liam Gauthier[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 email: ted.munn@queensu.ca
2 email: simon.dudtschak@gmail.com
3 email: 17cs51@queensu.ca
4 email: 19jal@queensu.ca
5 email: 18lmg3@queensu.ca

Abstract: *Motivation for this project came from the work being done by OpenAI in reinforcement learning research. Videogames were selected as the environment of choice since there already exist a great set of resources and implementation is feasible without hardware. The goal of the project was to develop an agent that can play the classic game Sonic the Hedgehog. Different learning algorithms were researched and the most successful was found to be the NEAT algorithm for this project. Our most sophisticated agent was able to complete the level that it trained on consistently after about 500 episodes of training.*

---

1. INTRODUCTION

1.1 Motivation

The inspiration for this project stems from a combination of interest in learning about the field of reinforcement learning shared by the team and the work being done by OpenAI. OpenAI is a company partly run by Elon Musk, that is tasked with the goal of discovering and enacting the path towards safe artificial general intelligence.

One of the areas of research that this company focuses on is the application of reinforcement learning using video games as the environment. Video games are a great platform for reinforcement learning because they allow for continuous feedback of the three elements key for reinforcement. These elements are a state, reward function, and action space.

Reinforcement learning is a field that remains in its infancy and is rarely applied in industry. Because of this, the range of problems to which existing reinforcement learning algorithms can be applied easily is limited. For this reason, we elected to use the Gym Retro library developed by OpenAI to streamline the development of our project in the Sonic environment.

1.2 Related Works

The environment of Sonic the Hedgehog was selected because there exist multiple resources for the same project. One research paper focused on creating a model that generalized well. The ability of the model to generalize in this case is measured as the ability of the agent to complete a new level after being trained.

One of the issues in the current field of reinforcement learning is that most models are tested in the same environments that they are trained in. The result is a different kind of overfitting that occurs, where the measured performance is biased.

1.3 Problem Definition

The goal of this project is to train an agent to play the classical game, Sonic the Hedgehog - Genesis, using reinforcement learning.

For this project, our team decided to focus on implementation of a more primitive agent without trying to make it generalizable. Understanding the

different avenues of reinforcement learning and how to implement using different libraries was challenging enough.

2. METHODOLOGY

Implementation of this project took place over two phases, research and development. This process was unique from typical data analytics projects since this is a relatively new field meaning there is no traditional development pipeline.

The team began with research on different kinds of reinforcement learning and which could be viable when applied to our environment. A Convolutional Neural Net (CNN) architecture and Neural Networks through Augmented Topologies (NEAT) model were chosen as the two approaches to be used.

To test how these two methods compare, OpenAI's Gym Retro library was used to create an environment and reward function. The two models were implemented using KerasRL and the NEAT library.

CNN's are used to filter inputs to neural net (NN) architecture. They help to reduce the dimensionality of the input and make the image being viewed more easier to use during calculations. The filter function can vary a great deal but for this application an average of 9 pixels was used to reduce the screen dimensionality.

NEAT models function by beginning with randomly generated weights. The weights are then tested in a complete episode before the model is changed. Models that are the most successful, i.e receive the highest reward value, are used as the basis for the next generation and are mutated and tested. Models that receive low rewards are not iterated upon and die out.

Training was conducted by having each agent play multiple times in the same state, where the state is the level in the game being played. Each agent played trough the state exploring which actions taken equate to the highest reward.

The reward is computed with a custom function that uses game variables to calculate performance. The primary metric for this application was x progress towards the end of the level.

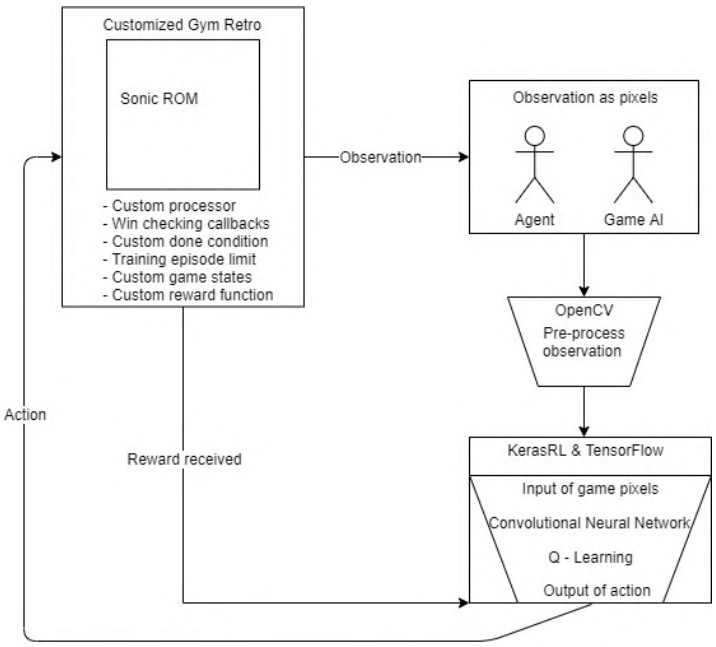


Figure 1: The architecture of different libraries, and how feedback to the model is generated based on the agent's actions in the environment.

3. RESULTS AND DISCUSSION

The reinforcement learning method that had the most success was the NEAT algorithm. After 500 episodes of training, the agent was able to consistently complete the level trained on.

The CNN developed trained for over 1000 episodes but challenges were encountered when it came to updating the weights. This was caused by a communication error between the gym Retro library and the keras model implemented.

It should be noted for future iterations of the project that changing the environment (game/state) used in the gym retro library is straightforward but creating new models requires integration of the environment and model libraries each time. After training for a certain period of time without results, the team decided to focus on the NEAT solution.

As mentioned before, creating a model that is generalizable is a different degree of challenge when developing RL projects. The model trained on one state with NEAT was unable to perform well on a different state. This is an example of the kind of overfitting seen in RL problems mentioned before.

4. CONCLUSIONS AND FUTURE WORK

There is certainly room for future iterations of this project. Using what has already been developed, the project could be expanded multiple ways.

The agent could continue to be developed in its performance and robustness. More research could be conducted to further develop the existing architectures to see if there is another configuration that is more ideal for generalization. Creating a model that is not overfitted to a particular game state is an aspect this project did not get to investigate thoroughly.

Training a more generalizable model would require development on two fronts. First the model would have to be more sophisticated in its architecture so that it would overfit to specific environments less. This more sophisticated architecture could then be trained in a range of environments to be able to perform well in a similar but new environment.

Creating generalizable AI was the focus of the research paper mentioned earlier [1] and the contest run by openAI in the past. An agents generalizability was measured by how well it performed in a randomly generated Sonic Map. `This kind of generalization has been demonstrated to be feasible to a successful degree and could be tested in any kind of environment/game.

Testing different architectures and types of models could be another area of research. Currently there are multiple reinforcement learning algorithms and approaches that could be applied to this project. Measuring how each kind of model performs given the same training environments could be used to learn what kinds of problems are ideal for each.

REFERENCES

[1] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, John Schulman, “Gotta Learn Fast”, OpenAI, 23, Apr 2018

Trading Card Vision System

Jonathon Malcolm[1], Alex Wassef[2], Hannah Berthiaume[3], Aidan Turnbull, Anne Liu

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: j.malcolm@queensu.ca
2 e-mail: alex.wassef@queensu.ca
3 e-mail: 17hmb2@queensu.ca

Abstract: *This trading card vision system is an application that will automatically determine the physical card quality of trading cards. It is designed to detect the edge quality and the logo centering quality in trading cards to be able to assist in card grading. The edge quality detection has currently achieved* 86.3% *accuracy and the logo detection are able to achieve a predicted grade (± 1) with an* 84% *accuracy. A simple web application was also developed to allow a user to easily interface with the system.*

---

1. INTRODUCTION

1.1 Motivation

Over the last decade, the portion of equity trading being done through automated means has increased drastically [1]. Human information intake is limited to simply the information we read, but machines can process terabytes of data in a matter of minutes, which is a gamechanger. While machines have already drastically changed the stock market, they have yet to affect some other trading markets.

Sports card trading is a particularly interesting market. One aspect of sports-card trading that has been particularly time-consuming and strenuous for this market is the determination of physical quality. Typically, before trading a card, the quality must be evaluated and verified by an external agency. The card is mailed to this agency, where workers evaluate the quality of the card and grade it. However, these agencies currently act as bottlenecks, as the process can take months to go through and is extremely costly. Automating this process could create significant cost and time savings.

In this project, we seek to create an automated method for the determination of physical card quality.

1.2 Related Works

Throughout academia there are many other related applications that involve identifying the quality of different objects in the real world. One very similar application is identifying cracks in concrete walls to help predict future failures [2]. A newer deep learning-based method for crack damage using convolutional neural networks was produced by Cha et al. [3]. In this paper a deep convolutional neural network is used for detecting concrete crack without calculating the defect features. The system is trained on images of 256 x 256 pixel resolution and these smaller images were created by scanning a large image in key areas where the system should examine for cracks. With this method they achieved 98.22% accuracy on their dataset of concrete cracks.

1.3 Problem Definition

The task is to automatically determine the physical quality of the cards. Current card grading systems look at several aspects of the cards:

1. Edge Quality
2. Logo Centering
3. Corner Quality
4. Surface Quality

The criteria for each aspect is described as the following:

1. Edges should be smooth and undented

2. The logo should be parallel to the edges of the card as well as proportionally distanced from each side of the card
3. Corners should meet at a 90° angle and be unbent
4. The surface should be free of dents, scratches and colour imperfections

The first two aspects of grading are included in the scope of this project as the corner quality and surface quality will need additional images to compute.

2. METHODOLOGY

The first step before any aspect can be evaluated, is locating the card within the large image. This is essential because the user does not need to place the card exactly in the center of the image, but rather as long as the card is contained in the image then the system should perform as expected. To accomplish this, the precise corners of the card need to be evaluated to a high degree of accuracy.

2.1 Corner Detection

Initially the captured image is read into the system as a high-resolution image. That image is then down sampled into a lower resolution image and blurred with a gaussian blur. This is to facilitate a more robust initial canny edge detection to find the rough corners of the card. The edge image is then further refined using a closing morphology (specifically a dilation followed by an erosion of the edge). The morphed edges are then piped through a contour finder using Suzuki et al's [4] method of finding contours in an edge image. With the contour defined, the corners can then be found by searching for the four vertices in the image. For each vertex, the high-resolution image is cropped to a 256 x 256 pixel image and the process is repeated to determine a more accurate corner location. The four refined vertices are then saved to be used later.

2.2 Logo Centering

To accomplish the centering detection, the card position and the relative logo position need to be determined. The card position can be determined from the four refined vertices previously computed. ORB feature detection is used to determine unique key points in the image. The key points in the card image and the key points in the logo image are compared using brute force matching to determine a mapping between the two images. When this is complete a square bounding box around the logo is added which can be seen in Figure 1. This is used along with the card edges to determine the distance vertically and horizontally between the logo and the card edge. The ratio between these two distances determines how centered the logo is where a 1 is perfectly centered and the more the ratio differs from one the less centered the logo is.

Figure 1: The logo outlined in white along to be compared to the edge of the card.

2.3 Edge Defect Detection

The edge quality pertains to the number of defects in the edges of the card in question. Similar to Cha et al. [3], we utilized a similar concept of creating multiple low-resolution images to train a CNN to detect defects. Using the previously computed card corners, the card perimeter is used as the scanline to capture smaller images. Following the scanline, a smaller image of 256 x 256 pixels is captured every 50 pixels to ensure that all defects are contained. These smaller images are shown in Figure 2. Each of these images is fed in to a custom neural network. This network consists of a pre-trained ResNet50 model along with additional layers to turn the model into a binary classifier and obtain appropriate defect information. Each of the smaller images is then classified as an image with or without a defect and this information is used to determine a grading for the quality of the edge of the card.

Figure 2: An example of a smaller image of the found edge in take directly from the high-resolution card image.

3. RESULTS AND DISCUSSION

The overall system behaved as expected for the two aspects graded. Determining the accuracy of the centering detection is difficult to quantitatively evaluate, as such, a simple qualitative evaluation is performed. The accuracy of the edge detection is quantitatively examined below.

The logo centering was run on multiple different images and each of the images was manually examined to determine the centering quality. These qualities were then compared to that of the system and are shown in Table 1 below. The variance is how close to perfect did the system predict the centering detection, 0 being the system was correct, 1 being the system was off by one quality ranking, etc.

Variance	*Occurrences (n=50)*
$p = 0$	40 %
$p = 1$	84 %
$p = 2$	98 %

Table 1: Accuracy of the logo centering detection.

The edge detection can be quantitatively evaluated based on the trained neural network. When training on 5 Epochs with 25 trials per Epoch, the system was able to achieve 86.3% accuracy for whether a defect was present in the edge images. The system training graph is shown below in Figure 3. It's evident that the system is being over-fitted because of a lack of training data. This can be alleviated by creating more data or adding in more dropout which is talked about in the next section. Overall the system performs with the current best accuracy found when examining different models.

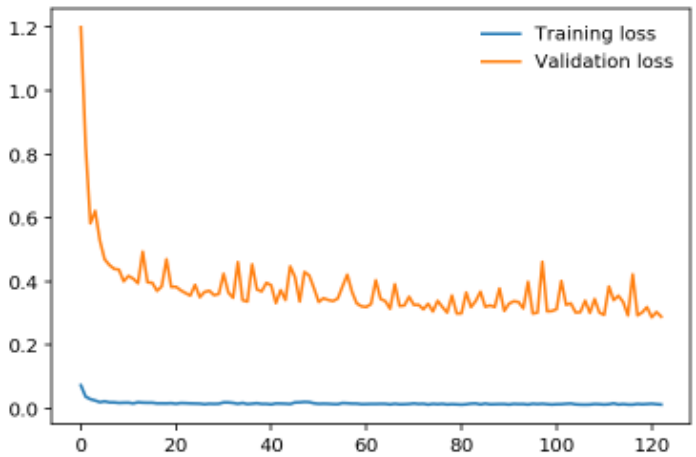


Figure 3: Training and Validation loss over 5 Epochs of 25 trails each.

4. CONCLUSIONS AND FUTURE WORK

Overall, the aspects of the system that were intended to work did as expected. The centering detected was able to determine centering with a high accuracy and the edge detection achieved 86.3% accuracy.

The next steps for this system would be to incorporate corner detection by training another neural network on the corner images of each card and determining a quality from that. In addition to that we would suggest examining different neural network architectures to improve the accuracy of the trained model as well as add more well labeled data to the dataset for training. In this neural network, it is evident that over-fitting is occurring because the training loss is well below the validation loss.

REFERENCES

[1] I. Aldridge and S. Krawciw, Real-Time Risk: What Investors Should Know About Fintech, High-Frequency Trading and Flash Crashes, Hoboken: Wiley, 2017.

[2] T. Nishikawa, J. Yoshida, T. Sugiyama and Y. Fujino, "Concrete Crack Detection by Multiple Sequential Image Filtering," Computer-Aided Civil and Infrastructure Engineering, vol. 27, no. 1, pp. 29-47, 2011.

[3] Y.-J. Cha, W. Choi and O. Buyukozturk, "Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks," Computer-Aided Civil and Infrastructure Engineering, vol. 32, no. 5, pp. 361-378, 2017.

[4] S. Suzuki and K. Abe, "Topological Structural Analysis of Digitized Binary Images by Border Following," Computer Vision, Graphics, and Image Processing, vol. 30, no. 1, pp. 32-46, 1985.

Natural Language Processing

Entity Resolution

Tania Sidhom[1], Mohammad Rashid[2], Ben Barcados[3], Khalid Rajan[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: tania.sidhom@queensu.ca
2 e-mail: 17mrr6@queensu.ca
3 e-mail: 16balb@queensu.ca
4 e-mail: 17knr@queensu.ca

Abstract: *Since data often consists of a vast amount of duplicated information, our goal was to identify relationships between entity-pair mentions that referred to the same object but were represented in different formats. Our team partnered with ThinkData Works, a Toronto-based start-up, to create an entity resolution model to extract valuable insights from textual data. The first step in this process involved building a named entity recognition model to recognize person names, organizations, and addresses. We built a pipeline of supervised learning classifiers that predict these entity labels based on pre-defined properties. For the entity resolution component, we present a method that uses unsupervised learning combined with similarity score computations that identified matching entities after they had been identified by the first model. Using a graph database, we constructed knowledge graphs to visually display the relationships found.*

1. INTRODUCTION

1.1 Motivation

The motivation behind this problem was to extract useful interactions from information hidden in text. Within textual data, there exist many occurrences of the same entity that are represented slightly differently. To the human eye, this can be easily identified, however automating the process is a complex task due to the unstructured and noisy nature of text. In many applications, as the volume of data grows, so does the need of inference between entities [1].

1.2 Related Works

There is no wide-spread solution for this problem, as finding the interactions across mentions heavily depends on the text source and entities that are being considered. Most work in entity resolution involves setting rules and performing redundant computations that match records based on the entity that is being worked with. However, this often becomes ineffective when large datasets contain entities with many different fields since comparisons become intensive and costly. Some have implemented blocking to combat this problem, since it divides records into disjoint sets. Comparisons are made within the same block to avoid entity pairs that will never match [2].

1.3 Problem Definition

To tackle this problem, it was split up into two parts: named entity recognition (NER) and entity resolution (ER). An entity is a term or phrase that represents an object in the real world. The entities used for the purpose of this project were person names, addresses, and organizations. The NER step is to classify a given entity into its predefined category when given labelled entities as input. The ER model builds on this to identify duplicates that are present in the entities. This requires a detailed examination of the structure and attributes of each entity type to be able to accurately determine matches. The relationships are linked to a graph database for knowledge graphs to be formed.

2. METHODOLOGY

2.1 Data

The data was acquired from ThinkData's Namara platform, which is a marketplace that provides users with access to large open-source datasets.

To construct a training set for the NER model, the only relevant data points were those that fell under the categories of person names, organizations, and addresses. After extracting about 1500 datasets from Namara, the data was filtered column-wise, where a column was extracted only if more than 60% of its cell contents were classified as one of the three entity types. To accomplish this, we used the open-source python NLP library SpaCy, which easily detects person names and organizations. The addresses were identified based on whether common words such as “avenue” or “ave”, “street” or “st”, etc. were present.

As there were expected inaccuracies with the extracted columns being classified correctly, our team manually filtered the remaining irrelevant information. We then automatically created three labelled ‘csv’ files, one for each entity type, and standardized the text by removing quotation marks and irrelevant punctuation.

2.2 Feature Engineering

The data was initially vectorized using term frequency-inverse document frequency (tf-idf), which associates weightings to words based on their importance in the entity. For example, the three most important words identified in organization names were “center”, “medical”, and “inc”. In addition to the vectorizer, other numerical features were extracted. The following chart displays the features along with their importance when evaluated under our highest-scoring NER model.

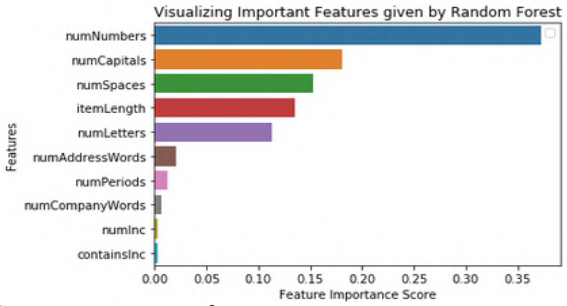


Figure 1: Importance of Features

2.3 Named Entity Recognition

Since current NER pre-trained libraries such as SpaCy do not support all entity types to the level of accuracy that is needed for our problem, we used a variety of classification models to predict entities from scratch. This was done using supervised learning, where the train and test sets came from the three datasets compiled in the data acquisition step, and the features from above were inputted.

2.4 Entity Resolution

Using the datasets from the NER model, we augmented the data by introducing variations to existing entities to define input pairs into our model. We further normalized the data by lowercasing all letters. The pairs were compared numerically using column vectors, which varied between each entity type. Taking person names as an example, each column consisted of a similarity score between the first name string and the second. The Jaro measure was also computed since it was originally developed for names in the US census. To compare first names and last names individually, we used the Levenshtein distance which computes the minimum cost of transforming one string into another (i.e. minimum edit distance):

$$D(i,j) = min\begin{cases} D(i-1,j-1) + d(s_j,t_j) \\ D(i-1,j) + 1 \\ D(i,j-1) + 1 \end{cases}, \quad (1)$$

where $d(s_j, t_j) = 1[s_j \neq t_j]$ and s & t are strings.

The affine gap distance compared the full names with each other since it handles long gaps and differences more efficiently. With similar reasoning, the address column vectors consisted of affine gap distance, Levenshtein distance, and Smith Waterman distance compare the strings and numbers. The company names used both Levenshtein and Dice, which tokenizes the strings into sets and accounts for missing words between the entities being compared. We determined the most optimal arrangement and number of scoring mechanisms applied per entity type using the theory of each scoring mechanism and by testing the affects of each one on the three entity types**.** More on the theory behind the scoring mechanisms can be seen in the Principles of Data Integration book [3].

The unsupervised learning models used were the Expectation/Conditional Maximization Algorithm (a probabilistic model) and K-means clustering, which partition entity pairs into clusters of matches and non-matches. Both models use the column vectors as input.

3. RESULTS AND DISCUSSION

3.1 Model Results

The following models and performance values resulted from our NER modelling phase.

Model	*Accuracy*	*Precision*
Gaussian NB	0.60	0.79
Logistic Reg.	0.83	0.83
Neural Network	0.86	0.86
Random Forest	0.90	0.90
Adaboost	0.61	0.61
GBM	0.88	0.88
KNN	0.85	0.85
SVM	0.88	0.88

Table 1: Results of our NER models

Informal parameter sweeps revealed that parameter configurations had little effect on performance. Poor performance of Gaussian Naïve Bayes may be explained by the fact that the features do not satisfy Bayes' assumption, which is independence of all features. Using Random Forest was most effective in classifying the entities.

The entity resolution unsupervised model results are presented below.

Model	*F1 score*
ECM	0.87 (addr), 0.76 (org), 0.86 (per)
K-Means	0.54 (addr), 0.51(org), 0.57 (per)

Table 2: Results of our entity resolution models per entity type

Each entity type was modelled individually, and the lower 'organization' score was due to a higher amount of ambiguity of the entity type. The reasoning for a lower K-means accuracy likely lies in the fact that the algorithm preforms well for linking entities in only the initial partition. ECM is iterative and takes the spread within clusters into account.

3.3 Knowledge Graphs

Knowledge graphs are directed graphs made up of vertices (nodes) and edges (relationships). The graphs were constructed using the graph database Neo4j, since traversing through knowledge graphs with large amounts of related data is faster than querying for the same data through a relational data base. Both NER and ER return information that can be made into relationships. Through a graph database, the nodes that share the entity types can be queried, for example, querying all nodes with type 'person' and returning every person in the database. Relationships between resolved entities can also be stored so that the graph can reference two nodes representing the same entity [4].

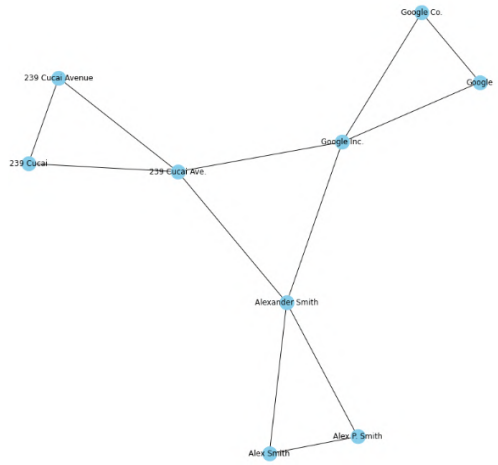


Figure 2: Knowledge sample graph output

4. CONCLUSIONS AND FUTURE WORK

In the project's current form, three entity types can be discovered (people, organizations, and addresses), and duplicate mentions of those entities can be resolved when presented in a different format. The accuracy levels of the highest-performing models are significantly high, however scalable large-scale entity resolution requires further applications and research for full success.

There are several potential extensions that should be considered. For example, introducing more entity types while ensuring scalability and expanding the network of relationships within the database. Currently, ER techniques mostly perform pairwise operations which rely on a pair of entities as input. Developing a method that can take in multiple entities and examine sentence boundaries will significantly increase the insights found in large volumes of data.

REFERENCES

[1] L. Li, "Entity Resolution in Big Data Era: Challenges and Applications", Proc. of the International Conf. on Database Systems for Advanced Applications, 2018

[2] F. Mandreoli, M. Montangero, "Data Handling in Science and Technology", Data Fusion Methodology and Applications, Vol. 31, 2019

[3] A. Doan, A. Halevy, Z. Ives, "Principles of Data Integration", Morgan Kaufmann, 2012

[4] Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases:* Beijing: OReilly.

Happy Transformer

Eric Fillion[1], Umur Gokalp[2], Logan Roth[3], Xaiver McMaster - Hubner[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: eric.fillion@queensu.ca
2 e-mail: umur.gokalp@queensu.ca
3 e-mail: 14lr18 @queensu.ca
4 e-mail: 17xjem@queensu.ca

Abstract: *Over the past few years, technology companies have been developing and releasing highly capable language models called Transformer models. However, these Transformer models are difficult to implement using conventional methods. To combat the issue of complexity and accessibility, we created and published a library called Happy Transformer that makes it easier to implement Transformer models. So far, the library has been downloaded over 2500 times and is well received by researchers from around the world. To demonstrate that Happy Transformer allows users to use state-of-the-art Transformer models easily, we competed in a public Kaggle competition that involved text classification. By using just five lines of code for the AI portion of the submission, the team placed in the top 1.5%. Happy Transformer will likely be continued as a future QMIND project, and this new team will continue to create an impact within the AI community.*

---

1. INTRODUCTION

1.1 Motivation

In 2016, Google published a paper titled "Attention Is All You Need," revolutionizing the field of NLP (natural language processing.) This paper outlined a new architecture for language models (LMs) called the Transformer. [1]

There have been significant breakthroughs in NLP due to the emergence of the Transformer model architecture. The GLUE (General Language Understanding Evaluation) benchmark is the standard benchmark that is used for evaluating the effectiveness of LMs [2]. In July of 2019, Facebook AI released a Transformer model called "RoBERTa" that outperformed the human baseline on the GLUE benchmark by achieving a score of 88.1% compared to the human baseline of 87.1% [3] [4].

We witnessed the increasing popularity and functionality of Transformer models and wanted to apply these models to solve a well-known NLP task called the Winograd Schema Challenge. However, we quickly discovered that it was incredibly complex to implement these models. So, we set out to create a tool that makes it easy to use state-of-the-art Transformer models.

1.2 Related Works

When we first began working on Happy Transformer, the only way to access many of the features of conventional Transformers was to use Hugging Face's transformer library (HFTL). HFTL is effective for graduate-level researchers who have time to write 100s of lines of code and learn about complex theoretical concepts. However, without extensive background knowledge and investing lots of time, it is nearly impossible to use HFTL effectively. [5]

When the team started working on Happy Transformer, there were only a few other libraries in the same category. Most notably, a library called Fitbert which was developed by a company called Qordoba. Fitbert allows users to perform masked-word-prediction with only a few lines of code. However, Fitbert and other similar libraries lacked fine-tuning capabilities for masked-word-prediction

and text classification, two fundamental NLP tasks. They also lacked some more specific functionality like question answering, text classification and next sentence prediction. [6]

Currently, a library called Simple Transformers is the most advanced and well-known library in the same category as Happy Transformer. However, Happy Transformer has an edge in terms of user-friendliness, clear documentation and offers entire features that Simple Transformer does not have like next sentence prediction. [7]

1.3 Problem Definition

In recent years, there has been an emergence of highly sophisticated NLP models based on the Transformer architecture. There have been attempts at making these models easier to use by companies like Hugging Face and Qordoba, but they either lack essential features or are too complicated for amateurs.

The team set out to create an open-sourced library that has a wide range of functionality for Transformer models, while still being easy to use. The team emphasized masked-word-prediction models since this is an area that has been adequately addressed by other libraries. The team also wanted to ensure that even NLP beginners can use the library by limiting the use of complex terminology and providing a substantial amount of high-quality documentation.

2. METHODOLOGY

2.1 Solution

We decided that the best solution was to create a wrapper on top of HFTL. Although HFTL is challenging to use, it provided the largest selection of features and was therefore chosen over less advanced libraries as the underlying platform for Happy Transformer to be built on top of.

Within Happy Transformer, we abstracted different HFTL models to only take the essential inputs from the user's perspective. Then Happy Transformer pre-processes the inputs and feeds the inputs into the selected Transformer model. The model's outputs are then processed before being returned to the user. Both the inputs and the outputs that the end-user deals with are easily understandable and minimal. Optional inputs are available for more advanced users who would like to have greater control over their models.

2.2 Design Process

There are many transformers available through HFTL, but we decided to focus on BERT, RoBERTa, and XLNet. These three were chosen due to RoBERTa and XLNet being state-of-the-art transformers and BERT having a large number of implementable pre-trained models.

We started with creating a basic wrapper on top of HRML that allows for the use of pre-trained models for masked-word-prediction. Then we moved onto adding fine-tuning modules for masked-word-prediction and text classification. Following this, the team implemented some more niche models for question answering and next sentence prediction. Table 1 illustrates which models have been implemented for each of the chosen transformers to date.

After we successfully implemented Happy Transformer, we wanted it to be readily available to anyone who would be interested in making use of it. To accomplish this, we decided to publish the library to the Python Package Index (PyPI).

Public Methods	HappyROBERTA	HappyXLNET	HappyBERT
Masked Word Prediction	✓	✓	✓
Sequence Classification	✓	✓	✓
Question Answering			✓
Next Sentence Prediction			✓
Masked Word Prediction Fine-tuning	✓		✓

Table 1: Transformers and the models implemented for each

2.3 Evaluation

There were three metrics we used to evaluate the project's success. First, the number of users the library would have. Next, the range of skill levels between our least to most advanced users. These two metrics combined would demonstrate how large of a market the library appeals to. Finally, the last metric would be our team's performance in a public Kaggle competition to demonstrate the state-of-the-art

capabilities of the models within Happy Transformer.

3. RESULTS AND DISCUSSION

From PyPI, there have been 2520 downloads to date, with 28 users giving the Github library a star. Happy Transformer has drawn users from all over the world with all ranges of expertise, from beginners to experts. The library has also received a dedicated post in a NLP spotlight series.

To demonstrate the capability of the library, the team competed in a Kaggle competition. The chosen competition involved taking a series of tweets and determining if the tweet was in reference to a natural disaster. The team made use of HappyROBERTA's text classification model for the challenge. Using just five lines of code to train and implement HappyROBERTA's text classification model, the team was able to place in the top 1.5% at the time of the conference.[1]

We were not expecting to have such a wide-scale adoption of the library. We were particularly surprised by the positive feedback we received from Ph.D. level users. Every day, we are gaining more users, and we are continually getting feedback from the NLP community. We were also pleasantly surprised by how easy it was to use our library to place at the top of a Kaggle competition. Our Kaggle competition results prove that even a beginner can compete amongst experts by using Happy Transformer.

4. CONCLUSIONS AND FUTURE WORK

Within the past few years, large tech companies have released state-of-the-art NLP models called Transformer models. The Happy Transformer team attempted to apply some of these models to the Winograd Schema Challenge but discovered that the models were incredibly challenging to use. So, the team created an API called Happy Transformer and published it to PyPI. This API makes it easy to fine-tune and use Transformer models with approximately five lines of code. Happy Transformer has been downloaded over 2500 times from users around the world. A broad range of people have stared the library, including Ph.D. level researchers, developers for AI companies and beginners. We have also used Happy Transformer to place in the top 1.5% for a Kaggle competition.

Happy Transformer will likely be carried on as a future QMIND project. This new team will focus on implementing requested features. For example, a user from Russia recently asked our team if we could implement multilingual models. Another user, who happens to be a Ph.D. level researcher, has asked us to implement a newer Transformer model called ALBERT. He has also asked us if we could provide more advanced customizability for masked-word-prediction fine-tuning.

In summary, we published a library that allows NLP beginners to use state-of-the-art NLP models. This library went on to be used by experts from around the world. Our project demonstrates that even undergraduate students can create an impact within the AI community.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, "Attention is all you need" Advances in Neural Information Processing Systems 30, 2017

[2] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding", ICLR, 2019

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", Arxiv, 2-19

[4] N. Nangia, S. Bowman, "Human vs. Muppet: A conservative Estimate of Human Performance on the GLU Benchmark," 57th Annual Meeting of the Association for Computational Linguistics, 2019

[5] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. L, M. Futowicz, J. Brew "HuggingFace's Transformers: State-of-the-art Natural Language Processing", Arxiv, 2019

[6] Quordoba, "FitBERT", PyPi, 2020

[7] T. Rajapaksem "Simple Transformers", Pypi, 2020

[1] This result excludes competitors who simply uploaded the leaked answers.

Lyric Generation using Machine Learning

Jacob Seiler[1], Ted Brownlow[2], Joshua Neizer[3], James McCarron[4]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada

1 e-mail: seiler.jacob1@gmail.com
2 e-mail: ted.brownlow@gmail.com
3 e-mail: 17jan3@queensu.ca
4 e-mail: jpmacarron11@gmail.com

Abstract: *Text generation is a powerful tool. In this paper, we go over how we created a text generating model to generate lyrics for songs, as well as a model for detecting a genre from an audio file. For the lyric generation, we created a dataset using the Genius API to scrape lyrics and built a Long-Short-Term-Memory recursive neural network. For genre detection, we used the GTZAN dataset and a feed-forward neural network. Our results were promising at ~74% and ~63% accuracy for the lyric generation and genre detection models respectively. Going forward we plan to implement more effective models, as well as a web application for people to interact with the project.*

1. INTRODUCTION

1.1 Motivation

Letters, words, and sentences are the basic tools that humans use to communicate. If we could teach a computer how to convey information in the same way, we could use that technology to better understand ourselves. The application of machine learning for text generation gives us that power.

New algorithms such as OpenAI's GPT-2 are already being developed and tested to generate essays and summarize articles [1]. Google is using a similar technology to track user actions with a text log [2].

Our team set out to create a model to generate lyrics for different genres of music given a phrase.

We also built a model to classify different genres of music given an audio clip.

1.2 Related Works

Recursive Neural Network (RNN) models are effective when dealing with text generation [3]. This is because to predict what the next word in character in a lyric is going to be, we need to know the phrase(s) that came before it.

For text-generation, there has been work done with character-based models and word-based models. GPT-2 uses a combination of the two called Byte Pair Encoding [4].

Word-based models have been shown to have higher accuracy, while character-based models have been shown to better reflect the punctuation and grammar of actual language [5].

LAMBADA is a task designed to evaluate a model's ability to create sentences using earlier data [6]. GPT-2 scored 63.24%. Humans usually score about 95% [5]. We can use this test to evaluate our model.

For genre detection, there has already been a lot of work done analyzing audio spectrograms to determine genres [7]. We will be using a feed-forward neural network and the GTZAN dataset for this problem [8].

1.3 Problem Definition

We worked to create a model for lyric generation using an RNN and a model for genre detection using a feed-forward network.

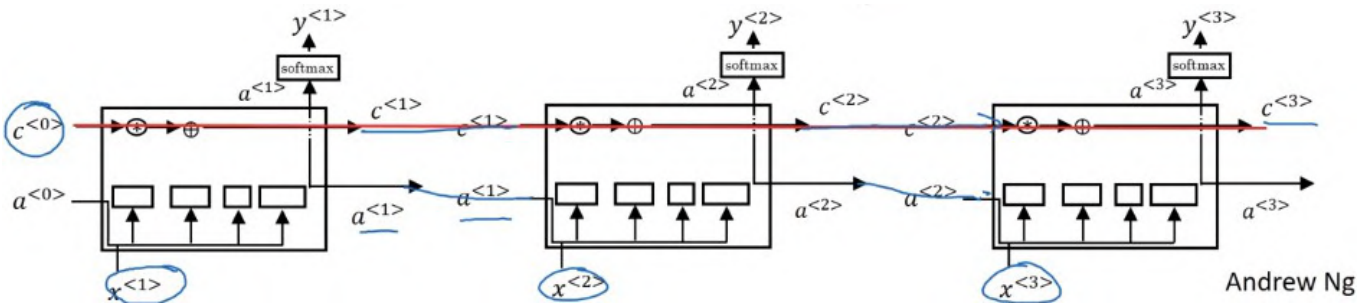


Figure 1: LSTM diagram, from Andrew Ng's Deep Learning specialization.

We chose to implement a Long-Short-Term-Memory for the lyric generation model since they are better at passing information from start to finish. They achieve this by giving each cell memory that stores what happened before, as seen in figure 1.

For the lyric generation model, we had to determine how we were building our data sets, and what type of network we would be using for the model.

Additionally, we had to decide between using a character-based model and a word-based model. We opted for a character-based model since we wanted to see if the model would pick up on grammar and punctuation specific for the lyrics being generated.

For both the lyric generation and genre classification models we had to decide what genres we would be using. We chose the 10 genres that are already included in the GTZAN dataset since this would keep things simple. The genres are Blues, Classical, Country, Disco, Hip-hop, Jazz, Metal, Pop, Reggae and Rock.

2. METHODOLOGY

For the genre detection model, we started with the GTZAN dataset. We used python to extract the spectrograms for each of the sample audio clips, and then extract features from those spectrograms.

For the actual model, we created a simple network with five layers of decreasing size. We set this to fit using a large batch size.

For the lyric generation, we started with web-scraping a list of the top songs for each genre and using the Genius API to obtain their lyrics.

We determined a list of "good" characters that the model could use and remove any outliers from the data. We also created some other cleaning functions such as making all characters lowercase and removing all text within square brackets (as is commonly found in the form of "[Verse]" or "[Chorus]" in most lyrics.)

An RNN model was created using an LSTM layer, Dropout, Output, and activation layer. We decided to utilize dropout to help reduce overfitting.

3. RESULTS AND DISCUSSION

We were able to create and train both models to a fair degree of accuracy. However, there is some overfitting happening.

Below are the results for both models.

A.I.	*Accuracy*	*Loss*
Lyric Gen.	73.93 %	0.9754
Genre Det.	62.99 %	1.0448

Table 1: Results of our models

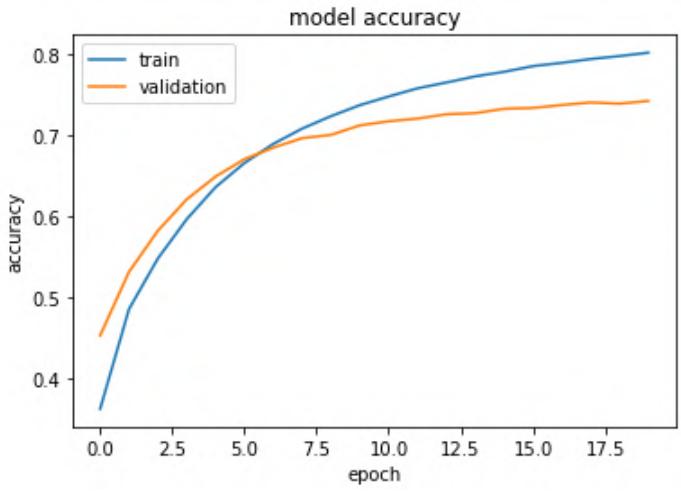


Figure 2: Graph of model accuracy for lyric generation model

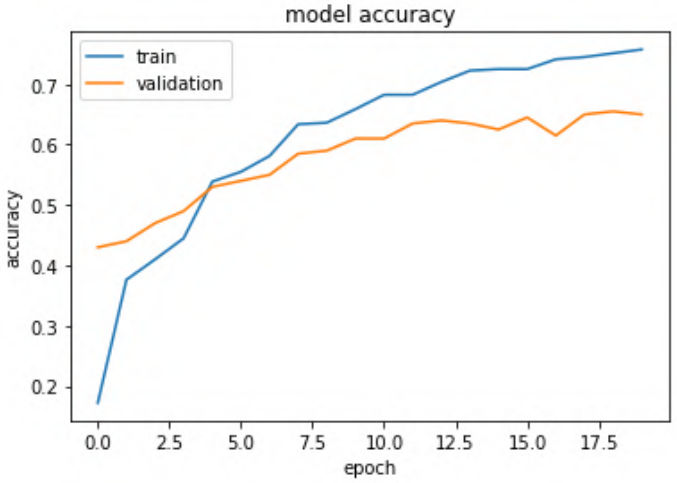


Figure 3: Graph of model accuracy for genre detection model

Analyzing the resulting data, we can see that our models have room for improvement.

The genre detection model shows signs of overfitting as the training accuracy is much larger than the testing accuracy, as seen in figure 3.

The lyric generation model shows signs of overfitting as well, as seen in figure 2. The validation curve shows that the model is effective in learning and generalizing the data.

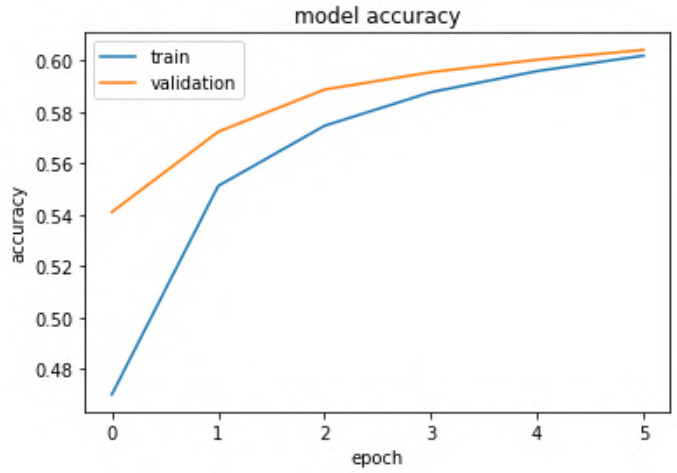


Figure 4: Graph of model accuracy for lyric generation model trained on LAMBADA dataset.

We also trained and tested the lyric generation model on the LAMBADA dataset to compare its performance to GPT-2's 63.24% [5]. Our model scored 60.68%, as seen in figure 4.

4. CONCLUSIONS AND FUTURE WORK

While we are happy with the project in its current state, we are working to improve and build upon what we have.

To improve the lyric generation model, we are experimenting with gathering larger datasets and doing more preprocessing to our data to ensure our data is more generalized for the problem. When trained with the LAMBADA dataset our model validation curve showed less signs of overfitting, as seen in figure 4. That is why we will be prioritizing our efforts on the data.

To improve the genre detection model, we are looking into implementing a convolutional neural network to be trained on the audio spectrogram images.

We are also working to develop a web application and API where users will be able to upload their audio clips to have the model generate lyrics for the detected genre.

5. REFERENCES

[1] J. Vincent, "OpenAI's new multitalented AI writes, translates, and slanders", The Verge, 2019

[2] A. Bala, A. McGlinchey, S. Sen, J. Jacoby, H. Hon, "Automatic text generation", Google Patents, 2003

[3] I. Sutskever, "Training Recurrent Neural Networks", University of Toronto, 2013

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, "Language Models are Unsupervised Multitask Learners", OpenAI, 2019

[5] P. Bojanowski, A. Joulin, T. Mikolov, "Alternative structures for character-level RNNs", arXiv, 2015

[6] Y. M. G. Costa, L. S. Oliveira, A. L. Koerich, F. Gouyon, "Music Genre Recognition Using Spectograms", IEEE, 2011

[7] G. Tzanetakis, P. Cook, "Musical genre classification of audio signals", IEEE, 2002

News Summarization

Robert Topham[1], Philip Basaric[2], Youngjun Lee[3], Sarah Nassar[4], Natalie Nova[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada

1 e-mail: robert.topham@queensu.ca
2 e-mail: 17pb22@queensu.ca
3 e-mail: 18yjl@queensu.ca
4 e-mail: 18sbn@queensu.ca
5 e-mail: 17nn19@queensu.ca

Abstract: *In surfing the plethora of content on the Internet, it can prove difficult to discern fact from fiction, especially when social networking platforms bombard users with high quantities of information. In order to safeguard against deception by misleading titles and biased political messages, a text summarizer was developed to condense news articles into a few representative sentences to allow users to decide if they are interested in reading the articles. Four natural language processing models were used to accomplish the summarization task and each of the models' effectiveness was measured. A trend that was observed, and is supported by studies, is that the extractive summarization approach, which concatenates important sentences from the original text, outperforms the abstractive technique, which identifies important concepts and generates new sentences. A Google Chrome Extension that implements the summarizer with the model that yielded the highest scores is in the works and will soon be able to help people on their knowledge-seeking journeys.*

---

1. INTRODUCTION

1.1 Motivation

Accompanying recent innovations in the field of natural language processing (NLP) by Devlin et al. with the creation of Transformer neural network architectures is an associated stream of advancements in extractive and abstractive text summarization methods [1]. Such developments are paramount because the high volume of information available on the Internet presents an issue with identifying what is true and important. There is an evident need for an effective, accurate, and readily available means of summarizing text, such as that in news articles. Thus, the problem is three-fold; 1) the most accurate text summarization technique must be determined, 2) the model that constructs the best summaries must be chosen, and 3) the selected model must be integrated into a platform that can be available for public use.

1.2 Related Works

With regards to the issue of optimal summarization method, there two distinct summarization approaches: extractive and abstractive. The extractive text summarization approach involves the integration of the most important sentences from the original text. According to Gambhir and Gupta, a common way to determine sentence significance is with the K-means clustering method, which is illustrated in Figure 1 [2]. This method involves clustering tokens on a similarity basis where more similar sentences are clustered together [2]. The tokens that occupy the largest clusters and are closest to the cluster centroid are selected to be sentences in the summary [2]. In contrast, the abstractive summarization mechanism involves the generation of new sentences with the most important concepts from the text. Gambhir and Gupta suggest that given the increased complexity of the

abstractive method and due to extensive NLP pre-training, extractive techniques are more effective [2]. Given the nature of the abstractive approach, the resultant summary can be inaccurate, provide misleading or created information, or be unrelated to the source text. This is evident from the work of Radford et al. with the release of their Generative Pretrained Transformer 2 (GPT-2) abstractive text generator. Radford et al. cited issues with their model confusing article details and low scores were achieved on the benchmark Recall-Oriented Understudy for Gisting Evaluation (ROUGE) scoring system in comparison to an extractive summary generated by randomly selecting three in-text sentences [3]. Pioneered by Chin-Yew Lin, the ROUGE scoring system compares a candidate summary to a reference one to assess its accuracy [4]. The ROUGE scoring system calculates the F1, precision, and recall scores for the generated summary by comparing it to some given reference summary [4]. As such, the ROUGE scoring system is utilized to assess the candidate summary's adequacy, as it evaluates the likeness of the generated and reference summaries by considering how many n-grams of keywords are contained in both summaries [4].

With regards to the optimal model, in reference to the work of Yang et al., the top four performing NLP models are Bidirectional Encoder Representations from Transformers (BERT), GPT-2, Robustly Optimized BERT Pretraining Approach (RoBERTa), and XLNet [5]. According to benchmark NLP assessments that Yang et al. used to evaluate each of these models, including the General Language Understanding Evaluation (GLUE) benchmark, Stanford Question Answering Dataset (SQuAD), and RACE scoring systems, XLNet is the best model [5].

With regards to maximizing user reach, as reported in the work of Hanif et al., 59 percent of study participants use Google Chrome as a web browser [6], and according to a 2018 study by the Pew Research Center, approximately 34 percent of American adults read news online [7]. Accordingly, a Google Chrome Extension would be a fitting user interface to employ to maximize the reach of a news summarization application.

1.3 Problem Definition

Attention is an important mental resource and it is imperative that people choose wisely what is worth their time. With today's technology, many are at the mercy of information that is constantly being thrown their way. Along with informative news articles, there is misleading content and information of questionable authenticity. To help people determine which articles deserve their full attention, a tool that provides a summary of key ideas can be developed to allow users to get the essence and decide if they wish to read more.

2. METHODOLOGY

The first stage of the design process consisted of data cleaning, tokenization, and lemmatization. A British Broadcasting Corporation (BBC) dataset was used for acquiring a collection of news articles. The text from the articles was cleaned and tokenized on an n-gram basis using Pandas. These tokenized data were then lemmatized – tokens in the collection were associated with a vector using token embeddings. The tokens were generated on a per-sentence basis using the Happy Transformers pre-trained GPT-2, BERT, RoBERTa, and XLNet models and tokenizers. Once the sentences were tokenized, they were averaged to ensure that each token had the same length. The tokens were then plotted according to their similarity to each other using K-means clustering, as seen in Figure 1. From the three clusters with the most tokens, the sentence that was closest to the cluster centroid was selected to generate the article summary.

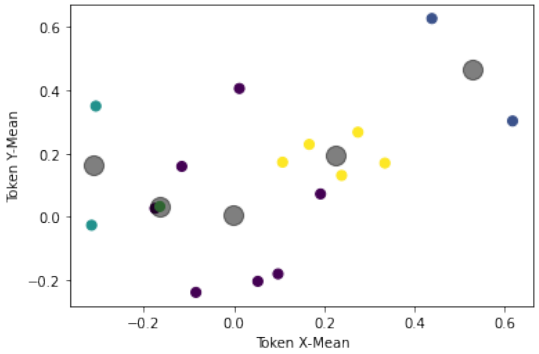


Figure 1: A plot illustrating the clustered tokens from the K-means clustering operation.

The second stage of the design process was the model evaluation. To evaluate the different candidate NLP models and select the one best-fitted for the specific news summarization task, a consistent metric needed to be employed. The ROUGE metric is the most commonly used, and most robust metric for evaluating summarization and machine translation tasks and was selected as the primary evaluation criterion for the effectiveness of the models.

The final stage of the design process was the creation of a front-end interface through which users could

access the summarization algorithm. To ensure that the design solution provided a seamless experience for the user, it was decided that the application should be implemented as a Google Chrome Extension. This enables the user to remain on the news website while obtaining the summary as a closeable window.

3. RESULTS AND DISCUSSION

The ROUGE metric was used to determine the summarization performance of each model through its F1 score. The F1 score is given by Equation 1 below.

$$F1 = \frac{(2 \times Precision \times Recall)}{(Precision + Recall)}, (1)$$

The F1 score measure is used to observe how the generated summary balances between relevant phrases and unnecessary ones. Key words were taken from the article headline because it is usually the author's summary of the main content. The F1 scores in Table 1 were determined by computing each model's individual F1 score on five different articles and their corresponding headlines. Each model's individual F1 scores were then averaged to determine which model had the best performance across the testing articles. A high F1 score indicates that the summary contains a considerable number of terms from the headline and is an accurate one. As expected from the research, all extractive text summarization models outperformed the abstractive one. Because XLNet yielded the most promising results, as seen in Table 1, it was chosen to be proceeded with.

Model	*Type*	*F1 Score*
GPT-2	Abstractive	6.25%
BERT	Extractive	13.33%
RoBERTa	Extractive	36.97%
XLNet	Extractive	57.73%

Table 1: The F1 scores for the tested models.

After testing the models, the front-end platform was created to provide an easy-to-use interface for users to employ the XLNet model. The model was deployed on Google App Engine as an API along with a web application to make pull and push requests in real time. When the model was run locally, results were returned within 20 seconds. With the model being hosted on a cloud service, the returned results of each summarization took 90 seconds on average because each pull request required the model to go through training mode once more with the new article.

4. CONCLUSIONS AND FUTURE WORK

As the model continues to demonstrate the ability to output results that are syntactically correct, further training should enhance its understanding of grammar and breadth of vocabulary. Once the model is deemed to have accrued enough data, it will be re-integrated into the web platform without intermediary training and should yield quicker results.

Using pre-trained models ensured a strong foundation for specializing them for the summarization task. In the future, the scope of the model's capabilities should be extended to summarize a wider variety of texts, such as research articles.

With the ever-growing amount of digital information, it has become increasingly important to choose the most relevant and informative. The text summarization tool hopes to facilitate this process by becoming more accessible, accurate, and efficient.

REFERENCES

[1] J. Devlin, and M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Cornell University, 11-Oct-2018.

[2] M. Gambhir, and V. Gupta, "Recent automatic text summarization techniques: a survey," Artifical Intelligence Review, 29-Mar-2016.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," OpenAI Blog, 14-Feb-2019.

[4] C. Lin, " ROUGE: A Package for Automatic Evaluation of Summaries," University of Southern California, 25-July-2004.

[5] Z. Yang, and Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," Cornell University, 02-Jan-2020.

[6] M. Hanif, M. Vighio, Z. Hussain, and N. Memon, "Comparative Study of Top-Ranked Web Browsers," Bahria University Journal of Information and Communication Technologies, 01-Apr-2015.

[7] A. Mitchell, "Americans Still Prefer Watching the News - And Mostly Still Through Television," Pew Research Center, 08-Dec-2018.

Robotics

Robot Object Detection and Autonomy

Sam Cantor[1], Ted Ecclestone[2], Adam Cooke[3], Tanner Dunn[4], Buchi Maduekwe[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 email: s.cantor@queensu.ca
2 email: 15eje1@queensu.ca
3 email: adam.cooke@queensu.ca
4 email: tanner.dunn@queensu.ca
5 email: 17ctim@queensu.ca

Abstract: *As autonomous robotics and computer vision become increasingly demanded in the industry, developing efficient systems to analyze data and perform computations on edge devices becomes fundamental in advancing autonomy. The goal of this project was to develop a real-time object detection library for general purpose robotic systems, and evaluate its effectiveness through an autonomous navigation task. The library was implemented on a TurtleBot3 with the Robot Operating System. The solution used YOLOv3, a single-pass deep neural network optimized for the COCO dataset, run on a custom architecture to enable distributed computation. It offers customization, scalability and integration for different autonomous tasks, as well as the option to retrain weights or change the model configuration. With advancements in architecture compatibility and optimization of visualization functions, this solution would serve as an indispensable resource for developers looking to quickly and effectively deploy unique object detection systems in autonomous robotics.*

---

1. INTRODUCTION

1.1 Motivation

With the increase of autonomous vehicles and smart IoT devices, there are a number of emerging opportunities for computer vision and deep learning to advance the industry. More specifically, effective object detection models could provide meaningful analysis on data and improve autonomy through proper deployment. Developing libraries to provide seamless integration and scalability of deep learning for robotic systems would allow users to easily deploy unique solutions without needing an in-depth knowledge of neural networks.

1.2 Related Works

Many companies are already utilizing deep learning with robotics to aid in autonomous tasks such as warehouse management and self-driving vehicles. Often, these solutions are not open-source and/or were developed for a specific use case. As a team, we wanted to provide a meaningful solution that would not just address a specific problem but could easily be adapted for additional uses. While there have been examples of multi-agent object detection [1] and mapping for service robots [2] with ROS, they do not directly address the motivation for a general purpose library. Additionally, most open-source libraries for robotic object detection offered either minimal features, limited performance or a non-intuitive interface. For these reasons, our library serves as a combination of many of the related works that we were inspired from.

1.3 Problem Definition

The goal of this project was to develop a real-time object detection library for general purpose robotic systems. In order to evaluate the effectiveness of our library, we will apply and demonstrate the enhanced

perception system with an autonomous navigation task, where the robot will locate and traverse to any specified object in a room. In addition, the goal is to publish this as an open-source library for others to easily use.

2. METHODOLOGY

2.1 Hardware

The solution was evaluated on a TurtleBot3 Waffle Pi equipped with a front-facing camera and LiDAR sensor. In order to reduce the computational load on the Raspberry Pi, algorithmic tasks were performed by a separate laptop with a 1050Ti graphics card.

2.2 Model and Data

There is a large variety of standardized object detection models to choose from. The desired model would be accurate, fast and functional in busy environments. You Only Look Once (YOLO) is a model developed by Joseph Redmond that offers state-of-the-art object detection with a single pass neural network [3]. This allows the model to perform at fast speeds while still achieving the same accuracy as competing models in high-context environments. With the most recent improved version of the model, YOLOv3, the option of the YOLOv3-tiny configuration compromises accuracy but vastly improves the max detection frame rate. We evaluated both configurations of this model to see if they would be a suitable fit for real-time object detection on the TurtleBot.

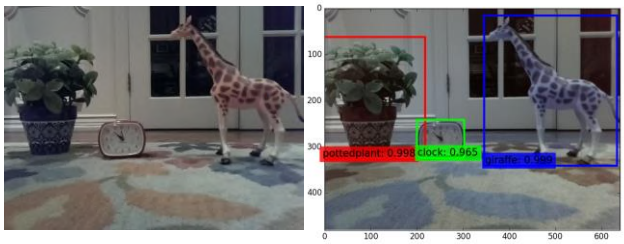


Figure 1 - YOLOv3 Performed on the TurtleBot3

Ideally, training of a model should be on applied to data that would demonstrate the use case for the deployed system. For this reason, we decided to use a subset of the COCO dataset, which contains a diverse set of common objects in high-context environments. Thus, our library will be able to support general purpose robotic applications and effectively detect objects in busy objects.

2.3 Robot Operating System (ROS)

Using ROS, we set up subscriber and publisher topics, where data and information could be communicated between the Raspberry Pi and computer. This enabled the model evaluation, traversal algorithm and feedback system to all run on the laptop, saving computing power for the TurtleBot. The Pi would simply read the data from the LiDAR sensor and front-facing camera and publish it for the computer to receive. We chose to use a max buffer, single queue subscriber when receiving the data on the laptop, as it would reduce individual data package loss and disregard old data.

2.4 Model Architecture

There were many factors to consider when choosing a deep learning architecture for YOLOv3. Libraries such as OpenCV would typically be the approach for a model of this configuration, but ROS's incompatibility with Python3 made this option not viable.

We chose to develop our own sub-library to integrate with Darknet, an architecture developed by the author of YOLO. This enabled us to design our ROS scripts and navigation algorithms in Python and run the neural network in C through Darknet. In addition, by utilizing C we were able to accelerate our single-pass network through CUDA and NVIDIA cuDNN. Parallelizing our model on the GPU vastly improved the performance of the detection system.

2.5 Navigation and Avoidance

In order to effectively test our real-time object detection, we wanted to apply it to a full application. The goal of this program was to declare a desired item in a room from the model's list of trained objects, and the robot would locate and drive towards it. The program actively uses a collision avoidance algorithm to ensure it does not hit any other objects. Since the navigation algorithm was not the focus of this project, it would spin around while it did not see the desired object. Once detected in the front camera, the robot would steer towards the object and slow down until it reached the proper distance.

3. RESULTS AND DISCUSSION

The main variable that we wanted to test our model on was response time. The goal was to achieve near real-

time performance, where the robot would be able to react and identify objects as soon as they were in the camera frame. This would be a valid metric to determine if the library would be suitable for use.

We were already aware that YOLOv3 would perform at a desired speed. With a 320x320 image, YOLOv3 can detect objects with an inference time as low as 22ms while retaining a 51.5 mean Average Precision (mAP) score on the COCO dataset [4]. In a vacuum, this model score would be ideal for all robotics, but it was important to consider data size and transfer rates.

When testing YOLOv3-tiny with data from the robot camera, we found that it was able to detect objects considerably faster than anticipated, with an inference time as low as 5ms. The problem is that the model's accuracy was not up to the standards we expected. The output images indicated that many objects were not detected, and that some labels such as humans, were often falsely detected. We decided that this was not unacceptable for autonomous robotic tasks.

With the full YOLOv3 configuration, we achieved results closer to what we desired. Factors such as the camera quality, framerate, resizing and processing power of the Pi negatively affected the model, but a response time of 50ms was achieved while scoring 46 for mAP-50 on our reduced COCO dataset. We considered this a suitable application for real-time robotics.

One unexpected metric that contributed to reaction time was publishing latency. After predicting the objects in frame with the neural network, the results are displayed and saved to an image file, which is then published to a visualization software for the user to see. This process takes roughly 350ms total. If providing user feedback was not needed, this would greatly improve the real-time performance of the robot, but this was incredibly important for testing and evaluating the system.

Algorithm	*mAP-50*	*Response (ms)*
YOLOv3	45.6	50.2
YOLOv3-Tiny	28.1	5.4
Image Publish	-	350.3

Table 1: Accuracy and Response Time for Image Algorithms

With the bottleneck of the publishing latency, the robot is still able to make quick decisions on how to navigate to objects, but must operate at a slower speed to preserve accuracy.

4. CONCLUSIONS AND FUTURE WORK

Our library successfully integrates state-of-the-art deep learning models with the Robot Operating System workflow to produce near real-time object detection in dynamic and high-context environments. Robotic systems are easily able to implement the library to customized use cases by retraining the weights or uploading their own configuration model.

To reduce the total response time in the system, we aim to implement a more efficient method of publishing condensed images to a user interface, as this played as the biggest factor in slowing the robot's performance.

We are looking to expand our library to function with a wider variety of deep learning models and architectures, including Keras and PyTorch. The goal is to make deployment of computer vision tasks for autonomous robotic applications as easy and scalable as possible.

REFERENCES

[1] R. Reid, A. Cann, C. Meiklejohn, L. Poli, A. Boeing, and T. Braunl, "Cooperative multi-robot navigation, exploration, mapping and object detection with ROS," in 2013 IEEE Intelligent Vehicles Symposium (IV), Gold Coast City, Australia, 2013, pp. 1083–1088, doi: 10.1109/IVS.2013.6629610.

[2] S. Ekvall, D. Kragic, and P. Jensfelt, "Object detection and mapping for service robot tasks," Robotica, vol. 25, no. 2, pp. 175–187, Mar. 2007, doi: 10.1017/S0263574706003237.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," arXiv:1506.02640 [Online]. Available: http://arxiv.org/abs/1506.02640.

[4] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," p. 6.

Project Repository:
https://github.com/QMIND-Team/dair-Perception

Robotic Path Planning

Luca Sardellitti[1], Jack Ellis[2], Jack Malloch[3], David Edwards[4], Eduard Varshavsky[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 email: 16ls53@queensu.ca
2 email: 15jge@queens.ca
3 email: 17jcm@queensu.ca
4 email: 17dae1@queensu.ca
5 email: 18ev@queensu.ca

Abstract: *There has been many scientific advances towards self-driving vehicles in the past two decades. Our goal was to create a system based on artificial intelligence that will execute a path dynamically between two points. Pre-built machine learning algorithms were built upon and Gazebo simulations were created for testing. A Q-Learning structure was used to teach the robot a policy. The inputs for the algorithm were 26 distances from LiDAR data, and the predefined goal box. Four stages were tested, all with varying difficulty. The agent had a 99% success rate in the easiest stage, while having an 80% success rate in the most difficult. During simulations, the robot perceivably attempted to avoid obstacles, however many cases may not have been the robot's fault. This may be why the success rate dropped across stages. To continue this work, more advanced stages could be created to optimize the agents learning.*

---

1. INTRODUCTION

1.1 Motivation

There has been a boom in academic and industrial attention towards self-driving vehicles in the past two decades, with useful advances in transport, military, and industrial production [1]. Our project focused on further researching the applicability of a robot to travel autonomously between two points using artificial intelligence. The research was meant to explore if it is possible for a built model to accomplish this, with further projects being able to build off this algorithm.

1.2 Related Works

Our project was meant to be researched and developed using a middleware called Robot Operating System (ROS), with Python being the programming language used to code the model. The project focused on making a low-profile robot called TurtleBot3 Waffle Pi achieve the path planning objective we were motivated by.

A similar objective was attempted in the Turtlebot3 manual where the robot used machine learning to reach a point in space while avoiding both mobile and immobile obstacles [2]. We further explored this by finding both the model and the reward function used in the manual and used it as a baseline to implement our version of reinforcement learning [3].

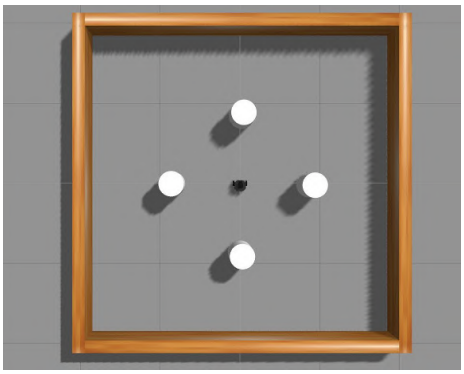


Figure 1: Example Stage Found in the Turtlebot3 Manual

1.3 Problem Definition

Our goal was to make this model available to tackle problems self-driving cars and robot assistants may have when getting from one point to another. More specifically, our problem was to create artificial intelligence that will let an algorithm predict and execute a path dynamically between two points.

2. METHODOLGY

2.1 Environment

The team began the design process by proceeding in a detailed analysis of the Ubuntu environment. Afterwards, it was essential to understand the robot operating system (ROS); a software that provides essential functions to develop robot applications. Each member became familiar with ROS nodes, packages, metapackages, RVIZ, RQT as well as the ROS CLI. The team then began to assess different Gazebo worlds.

2.2 Reinforcement Learning

Once the team was confident with the ROS and Ubuntu environment, the team began the preliminary design stage by using the suggested pre-built Machine Learning algorithms to train the Turtlebot3 robot in four different Gazebo worlds. Each stage consisted of a set perimeter where the robot had the task to navigating itself to the predefined goal. The algorithm used a Q-Learning based structure, as this is standard to learn a policy for a given agent under specific circumstances. The algorithm allocated a positive reward when the robot reached the goal or got closer to the goal, and a negative reward when it collided with an object or a wall. The only inputs to the algorithm were 26 obstacles distances around the robot found through lidar sensors, and the location of the predefined goal.

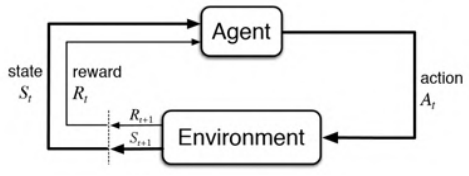


Figure 2: Basic Representation of Q-Learning

2.3 Process

The first stage consisted of a confined space made of four walls. The robot was then placed in the second Gazebo world that included four cylindrical pillars. The third world introduced motion to the pillars that would move as a unit in a constant circle. The final world consisted of the pillars moving as a unit in random linear directions while rotating. The team then established a design process that would allow the Turtlebot3 to learn and adapt in different Gazebo and real-world environments. It consisted of targeting certain aspects of the neural network and the python algorithms to improve the robot's learning abilities. Members would make impactful changes to the script and run it until the next meeting, where the team could analyze the outcome of the change together. The team evaluated the solution based on the speed at which the model learned, its ability to learn in different Gazebo Worlds, and the improvement from the default algorithm.

3. RESULTS AND DISCUSSION

The deep Q-Learning neural network used to train the path planning robot performed differently in each of the stages it was tested on. The accuracy of the model was measured as a success percentage, which is a measure of the chance that the robot reaches the goal position before exceeding the time limit or colliding with any obstacles or walls. The results for each of the four stages tested can be seen below in Table 1.

Stage	*Success Chance*
Stage 1 (No Obstacles)	99 %
Stage 2 (Stationary Obstacles)	90 %
Stage 3 (Constant Motion Obstacles)	85 %
Stage 4 (Random Motion Obstacles)	80 %

Table 1: Results of the Model on Different Stages

From analyzing the results, a clear trend was that the more unpredictable the stage becomes, by adding obstacles and changing their motion, the lower chance of the robot successfully reaching the goal.

After watching the simulations of the results of the experiments, the following reasons for these trends emerged. Firstly, it is important to note that when there were no obstacles present in the environment, the network would always make the robot move toward the goalbox in the fastest route possible. This makes it more difficult for the network to train on the harder stages with obstacles because it is likely that an obstacle is or will impede the shortest path to the goal. When watching the simulations of the obstacle filled stages, it was noticed that the robot perceivably tries to avoid an imminent obstacle by turning out of the way. However, this method does not always lead to the robot successfully reaching the goal for the following potential reasons. Sometimes, the robot would avoid an obstacle by driving in a circle and returning to the same location later. If the obstacle was still there when it returned, this would lead to an infinite loop of circling around, never reaching the goal and exceeding the time limit, being considered a failure. This could be considered only a partial failure because the primary objective of the robot in a realistic situation should be to prioritize not colliding with obstacles over reaching the end goal.

An alternative method of testing could be created where these situations are classified as impossible to reach the goal and the network must halt because it cannot proceed with its task. Additionally, when the robot failed by colliding with the obstacles it is usually because the obstacles in the experiment were not sentient and would not alter their path for any reason. This would lead to situations where the robot would do everything it could to avoid hitting the obstacle, but the obstacle would be the instigator of the collision. Again, this type of failure could be overlooked in a realistic scenario because it could be assumed that the moving obstacles would be human controlled obstacles or other path planning robots. In these situations, the other obstacles would be responsible for these collisions, and therefore the accuracy of the model could be improved.

4. CONCLUSIONS AND FUTURE WORK

The team took on the task of building a model for a the TurtleBot3 Waffle Pi that is capable of autonomous travel between two predetermined points. The project was written primarily in Python, but used ROS as an intermediate, and Gazebo to simulate training.

Our team began with a set of suboptimal prebuilt deep Q learning models corresponding to four Gazebo worlds with increasing difficulty. At first, alterations were made to python code, such as the reward function, percentage of random action, and timed decrease in random action. After improvements plateaued, changes were made to the network structure, simplifying the preexisting model. With each alteration, two to five days were given between meetings to allow ample training time.

We gauged success as a percentage of goals achieved before a collision or time ran out. Over the four aforementioned stages, success fell from 99% to 80% as difficultly was increased and obstacle predictably was decreased. Since human behavior is difficult to model, the obstacles do not perfectly represent a realistic scenario our project aims to solve.

In the future, the team would focus on two main objectives. First, setting a portion of goals as unachievable. Success in these scenarios would be exhausting time without a collision. This aims to prioritize obstacles avoidance is in training. The other goal would be a better fourth Gazebo world for the final product to be trained on. The obstacle movement and spacing could be optimized to allow for the most accurate model to be created.

REFERENCES

[1] Chaocheng Li, Jun Wang, Xiaonian Wang and Yihuan Zhang, "A model based path planning algorithm for self-driving cars in dynamic environment," 2015 Chinese Automation Congress (CAC), Wuhan, 2015, pp. 1123-1128.

[2] “ROBOTIS e,” *TurtleBot3 e-Manual*. [Online]. Available:http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Accessed: 01-Nov-2019].

[3] Robotis-Git, “ROBOTIS-GIT/turtlebot3_machine_learning,” *GitHub*, 10-Aug-2018. [Online]. Available: https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning.

Robotic traversal and pothole avoidance

Jacob Laframboise[1], Jack Demeter[2], Jesse MacCormac[3], Daniella Ruisendaal[4], Anne Broughton[5]

QMIND – Queen's AI Hub
Queen's University, Kingston, Ontario K7L 3N6, Canada.

1 e-mail: j.laframboise@queensu.ca
2 e-mail: jack.demeter@queensu.ca
3 e-mail: 17jgm1@queensu.ca
4 e-mail: 17dar8@queensu.ca
5 e-mail: 17akb6@queensu.ca

Abstract: *As robotics becomes ever more integrated into industry and society, improvements and insights into how robots avoid obstacles are critical to enhancing robotic abilities. We investigate this problem by developing an autonomous pothole avoidance system on the TurtleBot3 with the Robot Operating System. We developed a classifier CNN for pothole identification, and a localizer CNN to locate the pothole in frame. We calibrated pixel output to real-world coordinates, and developed a control module to avoid potholes. Our classifier CNN performed with 97.9% testing accuracy, and our localizer with a testing MSE loss of 0.4096 pixels. Each component successfully completes its task in the pipeline to avoid a pothole. We aim to improve component integration for full autonomy, and improve our dataset with more variation to enhance our model's ability to generalize.*

1. INTRODUCTION

1.1 Motivation

As the field of robotics continues to develop at a rapid pace it becomes increasingly integrated into our lives. For fully autonomous driving, robotic path planning and obstacle avoidance must be addressed. Without safe obstacle avoidance systems robots could only traverse pre-determined paths safely, which would restrict their application.

1.2 Related Works

The Autonomous Pothole Avoidance (APA) problem has been widely explored. More generally, pedestrian/obstacle detection and avoidance [1], and path planning [2] have been extensively studied. Each obstacle avoidance work first classifies, then localizes, and avoids the obstacle. This APA project drew on this structure and segmented the process into these common stages.

1.3 Problem Definition

This work explores how robots dynamically identify obstacles and conduct avoidance through path planning. This work aims to develop a system to detect the presence of a pothole, locate the pothole's relative position, and modify the traversal path to avoid it.

2. METHODOLOGY

2.1 Hardware and ROS

The solution was implemented on a TurtleBot3 Waffle Pi[1] with a Raspberry Pi Camera Module v2.1 on the Robot Operating System (ROS).

[1] https://www.turtlebot.com/

Three nodes were implemented on the ROS: a classifier node, a localizer node, and a pothole avoidance logic node. The classifier node took in image frames from the camera, and published a Boolean flag indicating presence of a pothole. If the classifier determined a pothole was in frame, the localizer node would generate a bounding box for the pothole in the image.

2.2 Classifier

A convolutional neural network (CNN) was used to identify if a pothole is in the image frame. This network comprised of five convolutional blocks, where each block consisted of a convolutional layer, max pooling, and dropout. The first convolutional layer had 16 nodes, doubling each subsequent convolutional layer. A (3,3) kernel with rectified linear unit activation was used. Dropout was applied with increasing strength towards the latter convolutional blocks. The model was completed with a 128-node dense layer followed by flattening, and a final single dense node with sigmoid activation. This model takes in a (128, 128, 3) RGB input, and outputs a one or zero to indicate whether a pothole is in frame. A 20% test split was used on a dataset totaling 7002 labelled images sampled from video. Data was augmented with rotation, translation, and reflection. The model was trained with the Adam [3] optimizer using binary cross entropy loss for 10 epochs.

2.3 Localizer

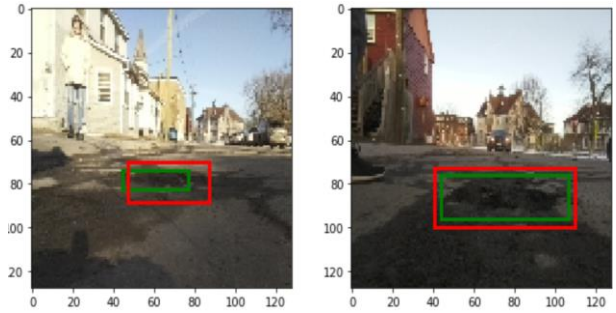


Figure 1: Output of localizer, with annotated bounding box in green and generated bounding box in red.

A CNN with similar architecture was used to localize the pothole in the robot's camera image. The difference is the model is finished with four dense nodes with linear activation for regression. This model takes in an RGB image of shape (128, 128, 3), and outputs a bounding box represented as (x, y, width, height) where (x, y) are the coordinates of the top left corner of the bounding box. 4248 images sampled from video were annotated using LabelImg[2], and a 20% testing split was used. Data was augmented with rotation, translation, and reflection. The model was trained for 20 epochs with the Adam optimizer on a mean squared error (MSE) loss function.

Figure 2: Augmented data sample, with rotation and translation. The green box is the transformed manual annotation.

2.4 Camera calibration

To map pixel coordinates to 2D world coordinates, the camera was calibrated with an array of markers at known distances on the floor. GIMP[3] was used to identify the pixel coordinates in an image captured with RViz[4] for each point. The Homography algorithm [4] was used to compute a linear map between pixel coordinates and 2D real-world coordinates. This calibration matrix was tested on new points to assess the calibration error.

2.5 Logic

To avoid the bounding box coordinates, the angle at which the robot must turn to avoid the pothole plus a 10cm safety margin is calculated. The robot then drives straight until it is at the same latitude as the bounding box, when it moves forward the height of the bounding box. Here, the robot is past the bounding box of the pothole, so it turns back at twice the original angle. It drives the same distance, returning to course and completing a symmetric trapezoid.

[2] https://github.com/tzutalin/labelImg
[3] https://www.gimp.org/
[4] http://wiki.ros.org/rviz

3. RESULTS AND DISCUSSION

3.1 Classifier

The classifier reached an accuracy of 97.9% on the testing set. A qualitative examination of its errors shows that it tends to error when there are significant cracks in the road that may be mistaken for potholes.

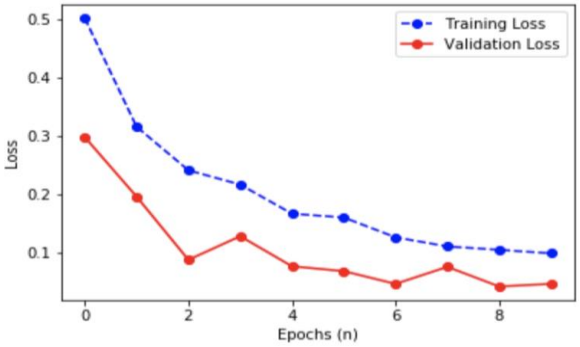


Figure 3: Learning curve of classifier

3.2 Localizer

The localizer achieved a MSE training loss of 0.5682 pixels and a MSE testing loss of 0.4096 pixels.

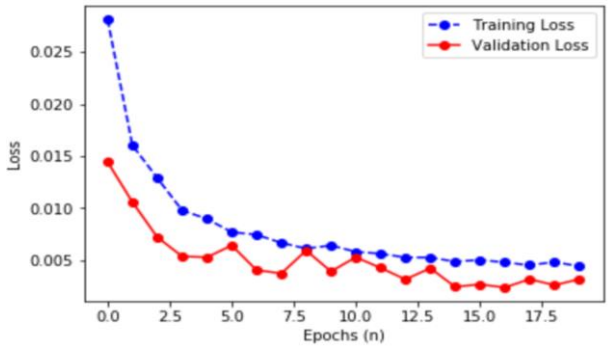


Figure 4: Learning curve of localizer

3.3 Avoidance logic

The avoidance algorithm was accurate in most cases; however, the algorithm occasionally over rotated the robot, breaching the safety margin.

4. CONCLUSIONS

Individual components required to deploy a pothole avoidance system were successfully built. The classifier and localizer quantitatively and qualitatively performed well on the potholes studied. Upon examining error cases in the models, it is likely that the models are overfitting to specific potholes.

4.1 Future work

We aim to integrate all components for fully autonomous operation identifying and avoiding potholes. The classifier and localizer are already integrated with the localizer only applied when the classifier detects a pothole. However, this will be integrated with the pothole avoidance logic, so that the robot can avoid a pothole it detects in frame with no intervention. Significant integration testing will be required.

In order to reduce overfitting in the classifier and localizer models, we need to collect a more varied dataset with a greater variety of potholes. Furthermore, a camera mounted higher off the ground would allow for more detailed images of the ground ahead. This could be achieved with either a different model of robot, or a different camera mount. These improvements should facilitate fully autonomous performance.

5. REFERENCES

[1] A. Pandey, “Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review,” *Artic. Int. J. Robot. Autom.*, 2017.

[2] M. N. Zafar and J. C. Mohanta, “Methodology for Path Planning and Optimization of Mobile Robots: A Review,” in *Procedia Computer Science*, 2018, vol. 133, pp. 141–152.

[3] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[4] E. Dubrofsky, “Homography Estimation,” 2009.

This work can be found at:
https://github.com/jlaframboise/Robo-Traverse