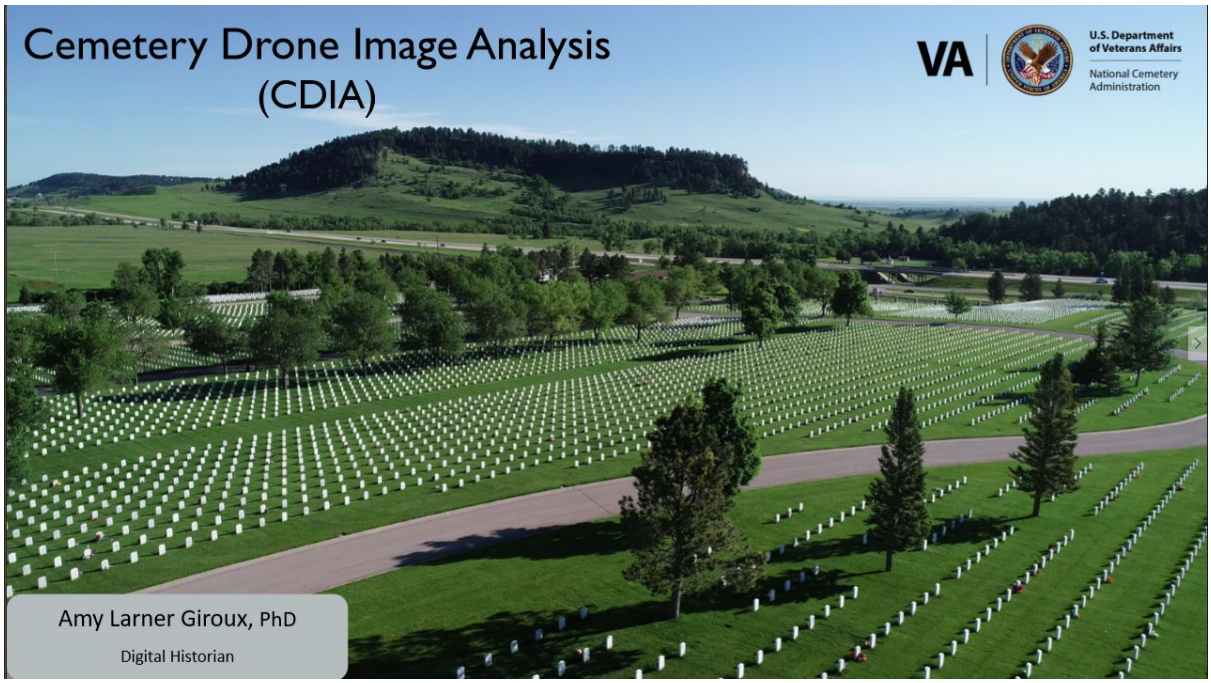
Group 06 - Cemetery Drone Image Analysis

Computer Vision & Machine Learning



Team:

Nicholas Evans

Maleah Jamieson

Alec Kerrigan

Grant Mullinax

Nicolas Soto

Sponsored By:

Dr. Amy Giroux

Table of Contents

Executive Summary

This cemetery drone project, sponsored by Dr. Amy Giroux, consists of 2 main sections. The first being the machine learning aspects, and the second being the database and user application.

Dr. Giroux flies a drone over national cemeteries, capturing high quality images of the headstones and surrounding areas from above. She then would take all these images and condense them into one large image of the entire cemetery. Previously, Dr. Giroux would create polygons for each of the headstones in this image by hand; this was a time-consuming task, considering these cemeteries were larger than 10 acres with hundreds of headstones.

Thus, the machine learning aspect of this project is to create a program that can recognize every individual headstone visible within an image. It needs to create a polygon around these recognized headstones, and use the geographic information given about the cemetery location to determine the latitude/longitude coordinates for each corner of the polygon as well as the centroid of it. This involves training our machine learning algorithm using images already collected by Dr. Giroux, as well as images taken from Google Earth.

As for the other side of the project, we are tasked with creating a SQLite database containing information for each cemetery. This will store each headstone within a cemetery, with a stone ID for each headstone (ordered in row order). Each headstone will have five coordinates,

the four corners and the centroid of the polygon. This database needs to export a GeoJSON file for each cemetery containing all the information for that cemetery.

The user application, as detailed in meetings with our sponsor, is more of a stretch goal, we will be creating a basic screen where Dr. Giroux can see the headstone selected and correct the latitude and longitude of the headstone within a text box. With sufficient time left, we would create a more user-friendly version possibly creating an app where she can move the corners of the polygons herself, thus editing the coordinates visually rather than by changing the values directly.

Problem Statement

Before this project, Dr. Giroux needed to draw the polygons by hand. In her most completed cemetery, St. Augustine National Cemetery, she took the coordinates of every single headstone manually and drew polygons around them in the image. This takes an unbelievable amount of time, and would be much faster with a machine learning solution as she would not need to go through and draw individual polygons, just fix a few of them if necessary.

Figure #1: Images of the St. Augustine National Cemetery taken from the interactive maps page.

Also, measured coordinates would not necessarily line up in an image due to image stretching when representing something not flat in a flat image. In figure #1, you can see there is a slight disparity between the measured coordinates and the headstone locations on the image when you zoom in.

Motivation

Nick Evans

"My primary motivation for this project is to study and contribute to the growing and cutting-edge field of computer vision. I have had limited opportunities to explore the field in my classwork thus far, but this project provides a golden opportunity. Even though the practical

benefits of computer vision are apparent, it is a complex and open-ended problem that continues to be rigorously researched.

My hope is to gain a strong understanding of both the fundamental concepts and tools of machine learning and computer vision through building a meaningful real-world project that demonstrates the power and usefulness of this technology."

Alec Kerrigan

"Years ago, I stumbled onto the youtube channel "Computerphile", a computer science focused educational channel that featured longform interviews with multiple experts in the field. One video in particular caught my eye, one focused on the newly released "AlphaGo". AlphaGo was a deep reinforcement learning model capable of playing the ancient japanese board game known as "Go"

""Go" was a game once thought unsolvable by computers, particularly because of its immense complexity as well as gigantic domain space. Chess, on the other hand, has a relatively small number of board states, meaning that a traditional program could easily "brute force" the game. It was for this reason that most computer scientists for the past few decades had concluded that a computer would never beat a top human."

"However, that sentiment clearly did not last long, as AlphaGo was released in October 2015 and was the first computer program ever to beat a human professional Go player without a handicap. More importantly, it beat Lee Sedol in March 2016, the greatest Go player perhaps of

all time. He was so taken aback by this event that 3 years later, he retired from Go, saying that there was no point in playing if an AI could beat him."

"I was instantly hooked on the concept of deep learning. I decided right then and there that I wanted to not only pursue a degree in computer science, but also a doctorate focusing on deep learning. While taking community college classes, I self taught myself the ins and outs of deep learning. Once coming to UCF, I immediately got involved in undergraduate research in machine learning and AI. In my junior year, I published my undergraduate thesis "Reinforcement Learning for Optimal Control of Network Epidemic Processes". I have also been working in two research labs at UCF and have two papers pending."

"I was extremely excited to get involved in a deep learning project for senior design, especially one related to computer vision, which I did not have the time to have very much experience in. Additionally, the ability to use my machine learning knowledge for a good cause and give family members of veterans closure and the ability to find their love ones is something I very much looked forward to when taking project. Working on this project and expanding my horizons in the realm of machine learning has been immensely rewarding to me and I look forward to where it takes me. "

Nicolas Soto

“I have always been fascinated with technology and its rapid advancement in today’s current applications. Once I discovered that I could focus my daily life on multiple fascinating technologies through the field of computer science, pursuing a degree in computer science was fairly obvious. One of the more exciting technologies being used today in various applications is machine learning.”

“Machine learning has a broad spectrum of use. Its use ranges from simple mobile video games to more critical areas such as the medical field, the military, and law enforcement. Machine learning can affect the lives of people on a daily basis even on the smallest scale, and for me, being able to help others and make people's lives easier is something that I have always had a passion for.”

“When Amy Giroux introduced the cemetery drone project to the class, I instantly knew it was a project that would help fuel my passion and introduce me to a more in-depth look at machine learning and its multiple applications in the real world. The cemetery drone project will make her professional work easier and more efficient, and being that she works around the world in order record data of significant grave sites and requires cutting-edge image processing for daily tasks, the completion of this project will not only help detect gravestones from aerial images, but it will possibly pave way for future projects or ideas of people who will face similar challenges that can be tackled through a similar application of machine learning.”

“This project will be a great step forward in my endeavors to improve people's daily lives through technology and will gain me valuable experience in the computer vision and machine learning field.”

Maleah Jamieson

“Personally, I find learning new technologies and new applications for technologies I already know to be the most exciting part of projects. While I am very interested in the machine learning aspects of this project, particularly the mass detection of features in an image, I am also looking forward to learning more about backend development. I have not yet had the opportunity to work with databases so I am definitely enthused about that! I’m also excited to apply my knowledge to a large project that will actually be used, and is not just for a class.”

“This project will also help develop my professional skills, like working in a group and communicating effectively. I am trying to be proactive in my teamwork. It also feels great to automate a monotonous and time consuming task for someone, as hardly anyone looks forward to doing the same things over and over again.”

“More broad, I am interested in the impacts that our project will have in both the preservation of veteran legacies and historical education. This project definitely feels more fulfilling than the side-projects I’ve completed in the past for other classes! I’m hoping it will be a fantastic final project to finish my undergraduate journey here at UCF.“

Grant Mullinax

“My main interest for this project is the unique challenge it presents. Many projects and problems I run into are some extension of “get data from api, transform it, present to user in a nice UI” which i've done way too many times for my liking. This project is very different. It is complex in nature and the essence of the problem itself is the hurdle, rather than the technologies involved, or presentation to the user.”

“I’ve never had to do a project with any heavy duty machine learning work, and certainly not from scratch where I have to assemble my own huge dataset and create my whole network from the ground up, so this is all very exciting to apply what I've learned in my classes and things I've seen online in such an important and exciting field. Not only that but this project is also important to me personally, as my grandfather died last year and was buried in the Cape Canaveral National Cemetery, and it is great to honor him and people like him in such a direct way. Having visited that cemetery and seeing how beautiful and large it all was really does help materialize what I’m doing conceptually.”

Personal Difficulties

Within this section, each team member acknowledges their specific challenges within this first semester as well as actions they can take to ensure a smoother second and final semester of senior design.

Maleah Jamieson

I think the biggest struggle this semester has been ensuring good communication with my group members. Due to the Covid-19 pandemic, our classes are online and will continue to be online for the coming semester. Also due to the pandemic, it is not safe for us to meet in person, which makes working together much harder. When class is in person, it's much easier to just meet after class since we are all physically present. However, because of the online format and strictly online communication with team members, it is much harder to find a time to meet because everyone's schedules are vastly different than we expected, especially with the employment and economic impacts of coronavirus. Also due to the strictly online communication, it is much easier for group members to not acknowledge messages from other members, and for questions to go unanswered. It is also easier for miscommunications to occur, as voice/tone actually plays a very important part in communicating, and messages do not convey this as well.

The actions I can take to make our communication more effective and reduce the potential for miscommunication is sending follow-up messages to my team members acknowledging their messages and questions, as well as pinging (using the built-in functionality of Discord) the person I am directly referring to ensure that they are notified of my message. I've also started phrasing my questions more actively to avoid the situation where I'm waiting for a response from a team member. For example, instead of asking a question like "should I do [action]?" I've been phrasing it like "I'm going to do [action] at [time], let me know if you have any other suggestions/feedback/etc. before then" so I'm not waiting on a response from anyone

before starting a task. It adds agency for others to respond to the message, and gives myself a set time to start the task so I can schedule around it as well. I think it has been pretty effective so far, so I will continue to do this.

Another difficulty I had during this first semester was learning the technology. I have never done any back-end development before this project, so catching up to the knowledge of some of my group members was difficult. In order to combat this I did two courses on LinkedIn Learning about database fundamentals and SQLite using Python. In addition to my efforts this semester, next semester I plan to work hard on the database aspects specifically, reaching out when I have trouble instead of silently powering through my issues with the technology. Teamwork makes the dream work after all!

Grant Mullinax

The biggest struggle for me this semester has definitely been keeping a good mindset in these solitary times. I always end up really affected when every day starts blurring together it becomes harder to chop up my day and stay on top of my time and the things I need to do and things start slipping through the cracks. It's difficult to stay motivated and on task when everything in the world is so crazy and life is so dull at the moment. Communicating with my teammates is also harder because in the same way it's very easy for myself and others to let things fall through the cracks when everything can be ignored so easily due to the lack of face-to-face communication.

Keeping everyone on the same page is noticeably harder than previous group assignments because communication is so detached and sparse. The way to avoid that is probably to schedule group meetings for everyone just to make sure everyone is talking and has something to say instead of just messages on a screen, but in the same way it can be hard to think things need to be addressed so directly when in an online call as opposed to just talking about it in person. Next semester I hope I am able to keep my groupmates more in the loop on what I'm doing and better stay in the loop with what my group mates are doing so we don't end up working past each other and forgetting important details that the others know.

Alec Kerrigan

"Personally, because my largest motivation for studying computer science was machine learning, my personal knowledge in other fields was lacking. Up until this project, I had actually never implemented any sort of frontend whatsoever. While web development is not a big part of this project, I had actually never even implemented "Hello World" in html."

"I had originally come on this project with the intent on primarily tackling the machine learning aspect of things, and taking the reins on deep learning. However, after carefully taking a look at what I was lacking and what additional skills I needed, I ended up volunteering to take the project manager role. Throughout the years of research, I had taught many people the various subjects I had learned, and therefore I felt I would contribute more value to the project by teaching others deep learning. I hope taking a broader role on the project will ensure that I develop a stronger skillset."

Broader Impacts

The intention of this project is to aid Dr. Giroux with the gathering of coordinate information of all the visible headstones in cemeteries. This is so she can attach biographical information of veterans to each of these locations in an online resource for the National Cemetery Administration Veterans Legacy Program. The purpose of this program is to connect veterans commemorated at national cemeteries to the broader public, through K-12 educational tools, mobile applications for cemetery visitors, and biographies of veterans available online.

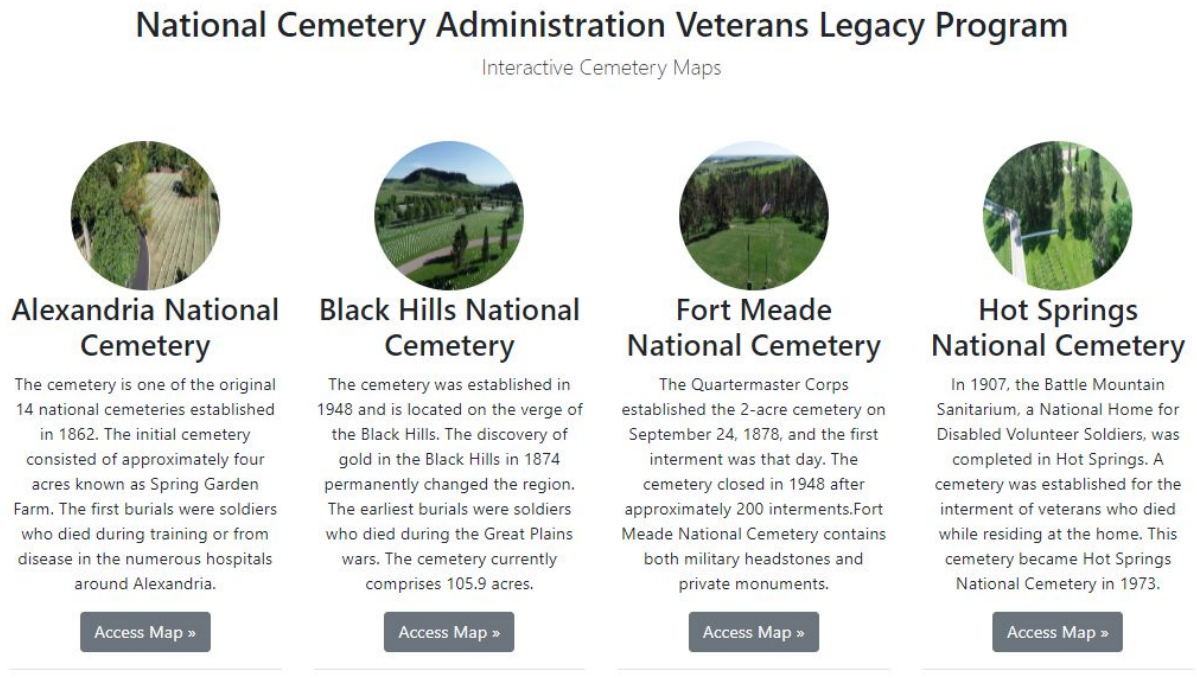


Figure #2: Interface for the interactive cemetery maps that Dr. Giroux is creating.

This has much broader impacts than just our team and Dr. Giroux. We are helping to preserve the memory of hundreds of veterans commemorated at these national cemeteries by

creating a resource capable of locating where their headstones are. Dr. Giroux will be using what we create and attaching biographic information to the coordinates we generate with our image analysis, digitally preserving their legacies. Our project also contributes to K-12 education and connecting communities with the veterans commemorated at national cemeteries, celebrating American history and the veterans that fought for our freedoms by ensuring that the community learns about them and continues to celebrate their memory. Digitally logging this information also helps for when headstones get weathered down and the writing on them becomes illegible over time.

Summer Semester Milestones

In the table below, the weeks begin on Sundays and the first week starts on May 17th, 2020.

Week Number	**Milestones**
1	Set up meeting schedule with sponsor Determine method of online communication with group members Set up group meeting times
2 - 3	Figure out roles and tasks Research machine learning technologies

	Complete 3 pages each for design document
4	Complete SQLite tutorial Set up database
5	Prototype user application/data entry screen
6	Discuss prototype with sponsor Make changes based on feedback Finish 3 pages each for design document
7	Finish image processing for training Finish 3 pages each for design document
8	Finish 6 pages each for design document Revise and finalize midterm design document
9	Complete 7 design document pages each
10	Complete remaining design document pages each
11	Finalize design document

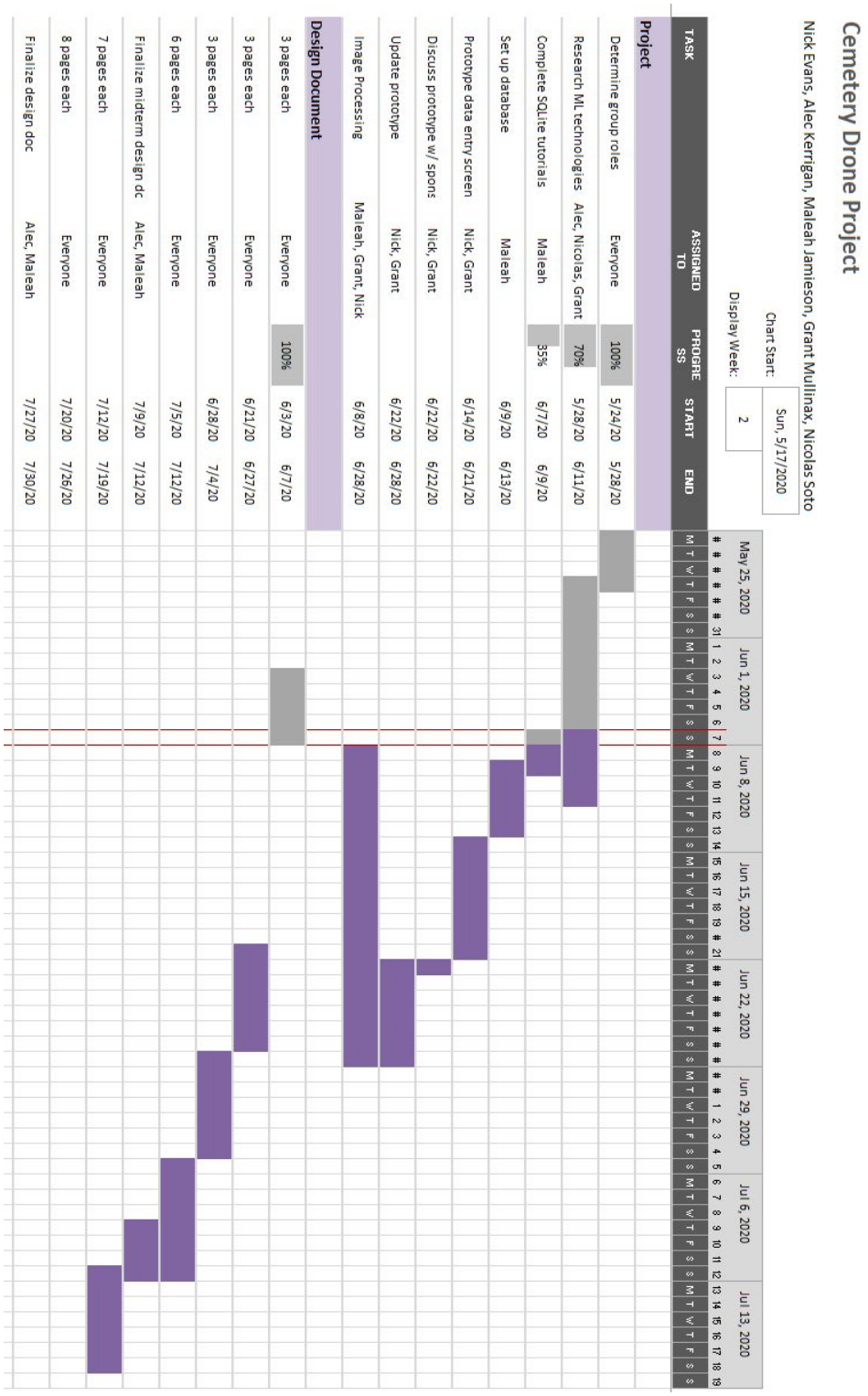


Figure #3: GAANT Chart for the summer semester.

Summer Milestone Reflection

Unfortunately we were unable to get the image processing task completed as above. Our GAANT chart was the ideal for our summer tasks, but because our design document was our main deliverable for the semester we had to shift our focus towards that, and instead prepare to tackle the image processing task at the very beginning of Fall. We also spent the entire semester researching our topics, thus we continued to be in the research phase after the task was meant to be completed. This isn't a bad thing, as more knowledge on the subject is always helpful. It would have been better to extend that task for the whole semester in the GAANT chart to accurately reflect this.

Background Research

Necessary to understanding which approaches to take for this problem is understanding what research has already been done in the field and domain of computer vision as it pertains to the particular problem of object detection and object marking. Rather than attempt to retread old ground, a survey of available methods ought to better inform the reasoning behind which methods our team used, as well as areas where the system can be improved in the future. Headstone Marker borrows many concepts from across the study of computer vision, and therefore explaining the background for each one is important.

Computer vision as a research subject has been around since the 1960s. As the name implies, computer vision attempts to mimic the human visual sensory system in order to provide

more detailed information about the world around the computer. Despite the monumental aspirations of this task, computer vision was originally envisioned as merely a summer project for MIT professor Papert Seymour.[1]

In this single summer, he and his grad students hoped to essentially solve the entirety of computer vision, a problem which at the time was not at all well-defined. In fact, his proposal for the project claimed that they would have successfully programed an entire pattern recognition system by the end of July, with the project only starting in early June. Clearly, this endeavor failed, but the field of computer vision was spawned from this project and has continued to evolve ever since.

Convolutional Neural Networks

For the past decade, deep learning has been the preferred method in machine learning. Papers discussing the method of deep learning that being feedforward multilayer perceptron's, have been around since the late 1960s.[2] However, it is worth noting that the computing power to actually use these networks for any relevant problem did not exist at the time. In addition, the methods for actually training and tuning these networks were not understood until many years later. At the time, the most common method for training multilayer perceptrons, also known as

---

[1] Papert, Seymour (1966-07-01). "The Summer Vision Project". *MIT AI Memos (1959 - 2004)*. hdl:1721.1/6125.

[2] *Ivakhnenko, A. G.; Lapa, V. G. (1967). Cybernetics and Forecasting Techniques. American Elsevier Publishing Co. ISBN 978-0-444-00020-0.*

"neural networks", were simple heuristic methods with results that were only marginally better than random brute force.

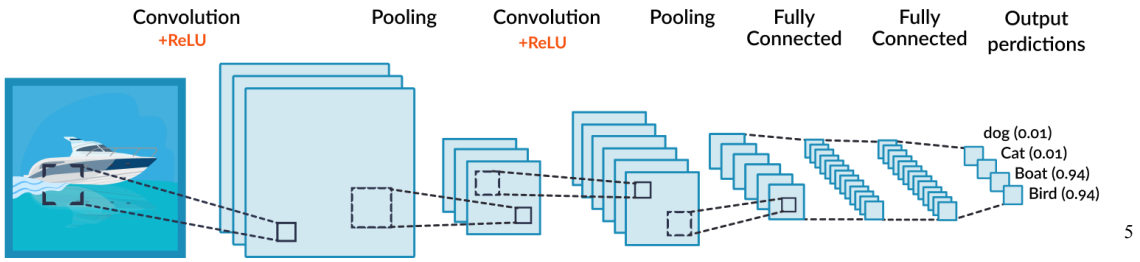
Backpropagation, the method for using partial derivatives to dynamically adjust the weights within the neural network, was outlined in the late 80s but was still not viable on a large enough scale for similar computational restrictions as above. In 2012, Yann LeCun's research group published their seminal paper outlining how to utilize modern GPUs and other advances in computer science to make backpropagation more viable[3]. Since them, deep learning has in large been the most prominent approach to machine learning in nearly all disciplines.

The use of convolutions was laid out in the mid 80s with a concept called "neocongnition" [4] . While the term is rather loaded, in essence it was the first attempt to learn not only features about raw data, but also relationships between data physically close to each other. Clearly, this is important for computer vision where simply laying out every pixel in a line for input to a dense neural networks would cause positional information from the image to be lost.

---

[3] LeCun, Yann A., et al. "Efficient backprop." *Neural networks: Tricks of the trade*. Springer, Berlin, Heidelberg, 2012. 9-48.

[4] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). *Biological Cybernetics*. **36** (4): 193–202. doi:10.1007/BF00344251. PMID 7370364. Retrieved 16 November 2013.



[5]

A "convolution" passes over an image that multiples its values with the pixel values on each channel of the image. This allows the network to simultaneously learn abstract features about an image while also reducing the information in the down to more relevant chunks. Typically, a "max pooling" layer is used between each convolution.

Max pooling examines the convolution and chooses a rectangular size to reduce it down to based on the greatest value on each region in the current image size. The purpose of these max pooling layers is to simply answer the question of whether some abstract feature exists within the image region or not, rather than retain redundant or useless detailed information about that feature.

After a series of convolution and max pooling layers, the remaining information is fed into a fully connected dense layer. This purpose of this final series of layers is to simply answer questions about the features extracted from the image such as "how many cars are in an image", or in our case "at which x/y region in this region of the image does a headstone exist if any". This is by far the simplest approach for this project and will be the first method used.

---

[5] Image Credit: https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/

Object Detection with Convolutional Neural Networks

The ability to recognize and classify images given to a convolutional neural network lends itself naturally to an object detection task as required by this project. In order to successfully complete the headstone marking tasks, our machine learning model must be not only capable of detecting when a headstone is present with relatively high accuracy, but also accurately report the position. These tasks may seem very similar to a human but are completely different and unrelated to a computer.

Typically, there are three distinct computer vision tasks.

- Image Classification: The input is an image with a single object with the output being a class label.
- Image Localization: The input is an image with one or more objects, and the output is the location of the objects.
- Image Detection: A combination of the two above tasks.

For our tasks, there is one and only one type of object that needs to be recognized: a headstone. Therefore, our tasks falls into the image localization task. Despite this, object localization and object detection are actually the same two tasks, as an image localization system must also be able to filter out objects it is not looking for, and therefore functionally must have the same capabilities as an object detection system.

Region with CNN features (R-NN)

The first most prominent object detection system is the ”Regions with CNN features” model, which has been the state of art for object detection models[6] . This model has been tested extensively against large image databases like ImageNet, allowing for it to classify and identify potentially millions of different objects. Regions with CNN features (R-CNN) breaks up the image into multiple overlapping sections that are then sent through a series of convolutional networks to determine candidate regions for images.

Once candidate regions are determined, those regions are sent into a second CNN that instead attempts to classify the object once a region is selected. This is considered a more “brute force” method for implementing object localization as it typically has to select a very large number of overlapping regions for identification instead of learning to find the regions purely through machine learning.

---

[6] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.

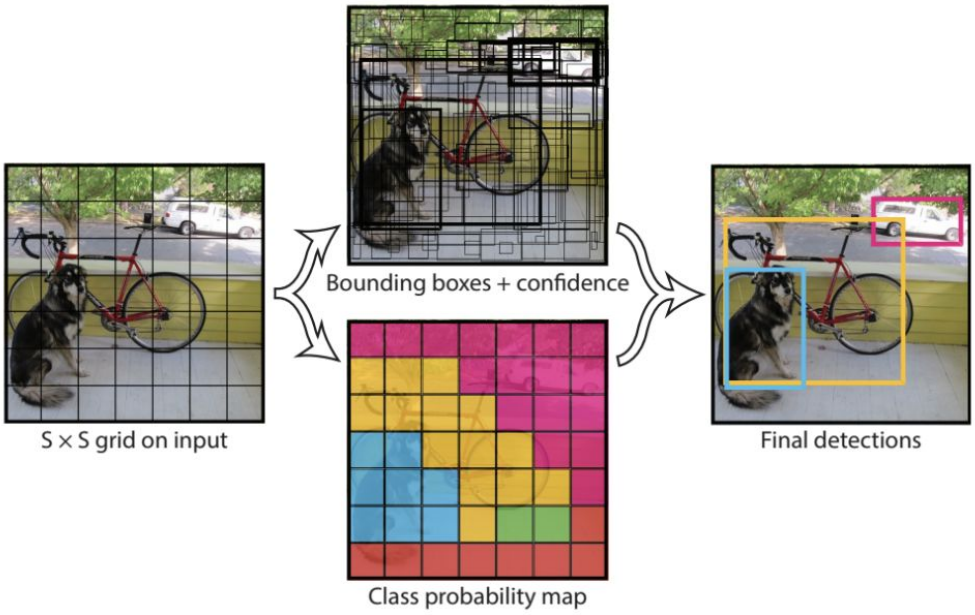
You Only Look Once (YOLO)

"You Only Look Once" or YOLO[7] attempts to be a unified object detection system that can run on anything, be used anywhere, and identify everything. It is often considered the current most powerful object detection model. The namesake for YOLO comes from the fact that it attempts to identify the objects and bounding boxes directly, rather than looks for candidate regions. This model provides less accuracy, obviously because its predicting both the object and object location, but is much faster.

YOLO's method is rather simple. First, the entire image is broken up into predetermined sections. Then, each section must predict for itself if it contains an object, and if so where is it. Then, the sections where objects were detected are slowly merged and the process is repeated on the larger sections. This allows YOLO to narrow down quickly where objects might be, which allows for the avoidance of redundant calculation. (diagram from the paper)

---

[7] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.



While YOLO is an extremely powerful tool that has broken many benchmarks in the domain of object detection and localization, it is far too overkill for our purposes. We only wish to identify a single type of object, and therefore our system ought take advantage of the face we already know what object we're looking for and where to mark it.

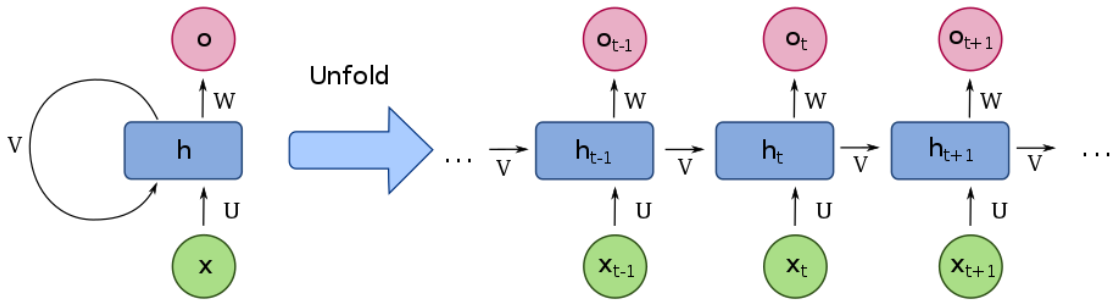
However, the headstone detector can take advantage of YOLO by using the same segmentation process to look for areas of possible interest when searching for headstones. Our next candidate network will use this process to break the image into sections, then identify areas of interest on the segment that may contain a headstone or parts of a headstone. If it manages to break a certain threshold, it will combine that segment with other segments of high probability to choose areas where a headstone is likely to occur and finally mark them.

Transformer Model

Recurrent Neural Networks

While convolutional neural networks are the primary machine learning model for computer vision, there has been extensive research into alternative means. Overall, the only thing needed for any machine learning model is an input, output, and function for the model to learn. Due to this, researchers have created many alternative models for object detection.

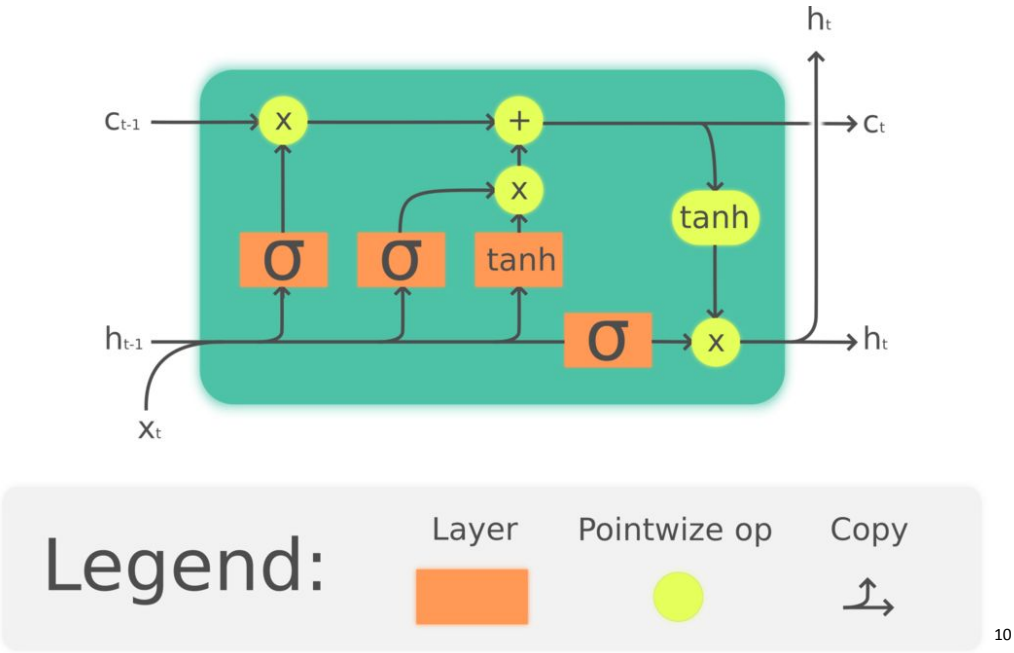
Recurrent Neural Networks, also known as RNNs, are a machine learning model that are designed to learn features about sequences and time.[8] They were originally outlined by David Rumelhart in 1986. RNNs learn by embedding functions within the neural network nodes themselves that are trained individually during each pass of the data. More specifically, time series data is meant to be inputted one piece at a time, but still retain some information about the relationships between each piece of data in time.



[8] Dupond, Samuel (2019). "A thorough review on the current advance of neural network structures". *Annual Reviews in Control*. **14**: 200–230.

By including functions within the nodes, themselves that update during each dataset during each time step, the neural network can therefore learn time-based information.

The downside of vanilla recurrent neural networks is that they can often retain too much information, and therefore redundant features can be learned over time that damage the utility of the netowrk. The solution to this problem is an alternation to recurrent neural network called "Long short-term memory"[9] These types of neural networks use the same type of embedded functions as recurrent neural networks, but also contain additional functions within each node that allows for the node to "forget" information. Like all parameters to a neural network, this "forget gate" function is learned as well.



[10]

As seen in the diagram above, each node receives information directly from the data being pushed through the neural network (ct-1) as well as information from the previous LSTM

[9] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". *Neural Computation*. **9** (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.

[10] Guillaume Chevalier / CC BY (https://creativecommons.org/licenses/by/4.0)

function (Ht-l). This means that the network is simultaneously learning information about the data itself, while also learning when to retain and forget information about the overall time series. Long Short Term Networks for the past few years have dominated most time series problems, in particular those in natural language processing, in the research literature.

Recurrent Neural Networks for Computer Vision

By themselves, recurrent neural networks are not very popular for use with object recognition or classification. Overall, convolutional neural networks simply have a stronger track record for these problems. While it is possible to reframe an image as a type of time series data, overall this introduces far too much redundant information into the LSTM cells that can lead to a degradation of performance.

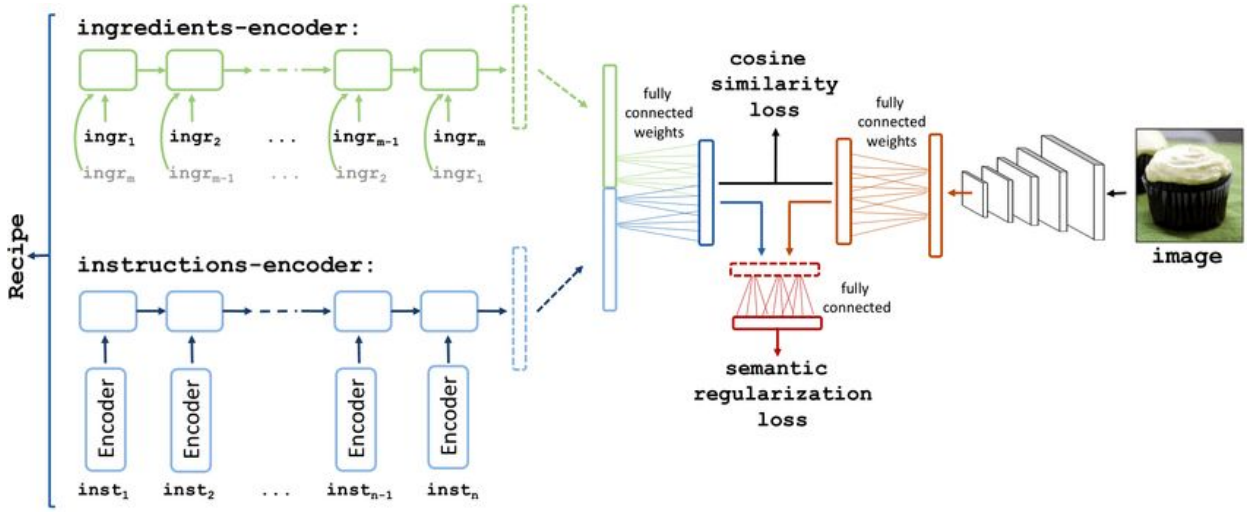
There do exists ways that recurrent neural networks can assist in the performance of convolutional networks, however. In the domain of object detection, it can often be difficult for a convolutional network, even an extremely complex one such as ImageNet, to understand the entire context of an image.

 [11]

For example, on the above image, a convolutional network would have to recognize the sky, the clouds, the mountains, the shit, the ocean, and more in order to properly understand the entire context of the image. However, convolutional networks are not particularly powerful for understanding the *context* of images so much as they are powerful for understanding the *content* of images. Therefore, it may have trouble fully articulating what is happening in the image.

To solve for this, a recurrent network can be introduced. Essentially, a recurrent network learns an "instruction set" for putting together the context of images. The encoded instructions are fed simultaneously into a separate network that the convolutional network is also fed into. This allows for the CNN to focus on understanding the raw content on the image, while the recurrent network learns a process for decoding what these different parts mean.

[11] 0-0t at Japanese Wikipedia / CC BY-SA (https://creativecommons.org/licenses/by-sa/3.0)



[12]

This is one method that can be used for headstone detection. Rather than trying to analyze the raw pictures of the graveyard images, we can instead have the convolutional neural network extract abstract features of the image that can then be decoded by the recurrent neural network for the purposes of marking the position of headstones. If a convolutional network tries to just read the raw pixel data of the image, it may run into problems with shadows, moss, and other natural occurrences than can obscure the position of the headstone. An instruction encoded may assist in creating rules for the machine learning model to decide when something is or isn't a headstone, which ought to lead to more accurate results. This is one approach that will be tested.

[12] Salvador, Amaia, et al. "Learning cross-modal embeddings for cooking recipes and food images." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

Attention Networks and Transformers

One of the largest problems with recurrent neural networks is of course their recurrent nature. They must be trained and executed sequentially, meaning that data points must be fed in one time step at a time, and subsequently classifications and decisions require the data to be fed in the same way. The not only slows down the training and execution of the network, but also prevents the machine learning model from understanding immediate relations and connections within the data.

Although the internal functions within the nodes are designed to learn these relations over time, it is impossible for a recurrent neural network to see that two data points are physically next to each other in some sequence. It is primarily for this reason that convolutional neural networks are preferred over convolutional networks for directly applications in computer vision. Recent advances from Google's research labs has formulated a new neural network architecture called “transformers” that aims to solve this problem.[13]

In order to directly read in sequence data such that these systems can be used for computer vision problems, the actual sequential aspect of the neural’; network needs to be eliminated. Instead, different parts of the time series, or in our case the image, needs to be read directly as entirely different data points without losing the benefit of these data points being

[13] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

directly next to each other. Some previous models such as ByteNet[14] and ConvS2S[15] have attempted this solution using only traditional convolutional networks. These models use a "signal threshold" to bias network toward relating data that is physically closer together. ConvS2S linearly degrades the relation signal as distance increases while ByteNet does this logarithmically.

The concept of attention networks that read in multiple parts of time series or image data once so that the network can learn which parts to "pay attention" to is not entirely a new concept. Previous research has been able to reduce the amount of data needed to be read in to understand the sequential aspect of data by orders of magnitude.[16] However, these systems still rely on some aspect of recurrent neural networks in order to function, and therefore make poor substitutes for simple convolutional networks with respect to the problem of object marking, and headstone marking in particular.

The recent google research, however, has successfully been able to eliminate all aspects of recurrent networks from the attention models and therefore open them up to use with image processing, computer vision, and object marking. Their model uses a separate network to embed

---

[14] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. arXiv preprint arXiv:1610.10099v2, 2017.

[15] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. arXiv preprint arXiv:1705.03122v2, 2017.

[16] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model. In Empirical Methods in Natural Language Processing, 2016.

or reform the input data into a universal language that a basic attention head can read and therefore pinpoint which parts of the image or sequence to "pay attention" to. This is known as the "Transformer" The architecture is as follows.

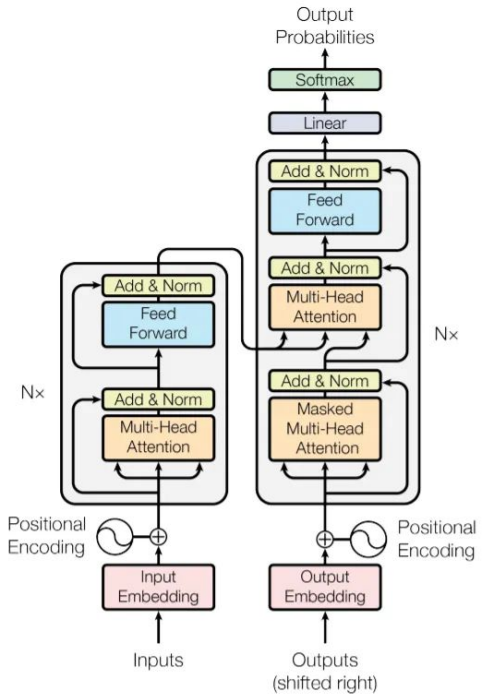


Figure 1: The Transformer - model architecture. [17]

By horizontally stacking multiple interactions of this "attention head", the different heads slowly feed into a single head that can learn some sort of information of classification. More importantly, the output of the attention heads is fed back into the inputs (as shown in the image), allowing more much more complex embedding.

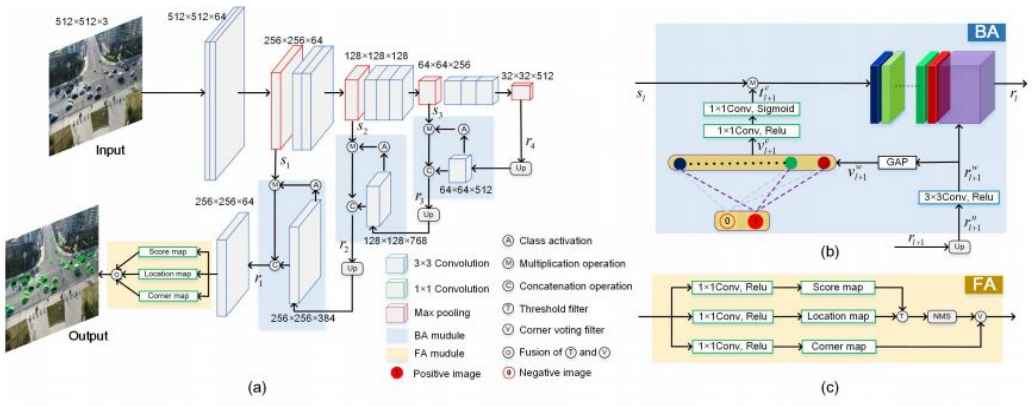
This method was found to have outperformed the state of the art on times series classification and generation by about 20%, while it surpassed object recognition tasks by a similar metric. However, this comparison was only against convolutional models that already

[17] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

included some sort of recurrent neural networks. It did not compare against purely convolutional models, and therefore it is not clear that this method by itself would be worth exploring for headstone marking.

Further research has shown that while convolutional networks may not be completely eliminated for this problem, it is possible that a heavy use of guided attention may be a stronger model for headstone marking than simply using a basic CNN to read the image and search for objects. Research just last year from the Chinese Academy of Sciences has shown that attention networks, through continued advancement, may be able to replace convolutional networks.[18]

Their guided attention network, or ”GANet” searches the image and uses a pretrained attention network to image down to candidate areas for the convolutional network to search. GANet from their paper is described below.



18

Cai, Yuanqiang, et al. "Guided attention network for object detection and counting on drones." *arXiv preprint arXiv:1909.11307* (2019).

Their model was trained specifically to spot and mark cars, but the same model can be trained to spot headstones, as there is nothing specific to cars in their paper. The simple explanation of their model is that an attention head is trained to "monitor" each convolutional layer in the network to slowly learn what sort of patterns are most important for spotting a car while also learning useless and redundant features.

Normally, a convolutional network needs to completely "reduce" an image down to its deepest layers before it can be passed to a simple dense network for classification. GANet, on the other hand, can actually signal to the network that a car is likely in a certain spot before the image is fully reduced, and only passing information to deeper networks if the model is not certain if a car is present or not.

Data Preparation

Planning

One issue we encountered was lack of training data for our complex ML models that are required for this project to be functional. Extremely high resolution images of 5 national cemeteries were supplied as images the network will be applied to, however despite this we will not end up with nearly enough data for the network to be trained on. In order to mitigate this issue we have come up with two primary solutions:

1. Splice the high resolution sample images, along with attached gravestone poly data automatically with some tooling that would take randomized chunks of these images and then perform a set of randomized operations on them such as scaling, rotating, color shifting, or skewing to produce significantly more sample images.

2. Retrieve and tag relatively low resolution images of cemeteries taken from google earth and use them for training data.

Figure #4: Example high-resolution graveyard image

Each piece of training data needs to have 2 components, the image, along with the set of geojson polygons. One of the drone scans we received has the data attached but we still need the polygon data that will end up being the output of the trained machine learning algorithm once it is working. For this we have a solution planned, along with a backup.

Tagging Gravestones With Polygons

For solving the problem of tagging the input data with polygons we once again have 2 solutions, one being preferable to the other.

1. The first is to use some algorithm to approximate the gravestone polygons due to their regularity and relatively solid color.

   Depending on the success of this approach, it may even be preferable to use in the end program if not much manual correction is required for accurate results. In this case we would have the machine learning instead output a series of points where gravestones are located, then the algorithm would take that point as input, and use the color and position of it to approximate an appropriate polygon for the gravestone.

Figure #: Example of how a tool might be able to see gravestones easily and detect their areas

2. If this approach does not work well, we can instead just tag each image by hand, or at least use the algorithm described in the first section with modifications on a user-controlled case-by-case basis with some correction in post. This case is dangerously close to failure as we're already pushing the limits of manual labor in terms of tagging.

Results

The prototyping tool created to solve a lot of the issues related to tagging the data was created in python, and uses the opencv library for quick interfacing with the user, along with easily available implementations of common image processing algorithms like canny edge detection. In the application users are shown a full image of the graveyard file they provide to the application, and are able to click on it, each click representing a gravestone. Once the user has selected every gravestone in the image they are able to exit out and save the data or image produced by their

clicks to an .png mask file that can then be fed into tensorflow for training. The interface is very primitive as it is not really intended for use by the end users.

Figure #5: Gravestone image before and after selecting gravestones with the prototyping tagging tool

In the final workflow, this prototype is to be replaced partially by open source tool LabelImg, which has much more complete functionality in regards to tagging images. The data will be exported from LabelImg in YOLO label format to a program that will take the bounding boxes and find the mask of each gravestone.

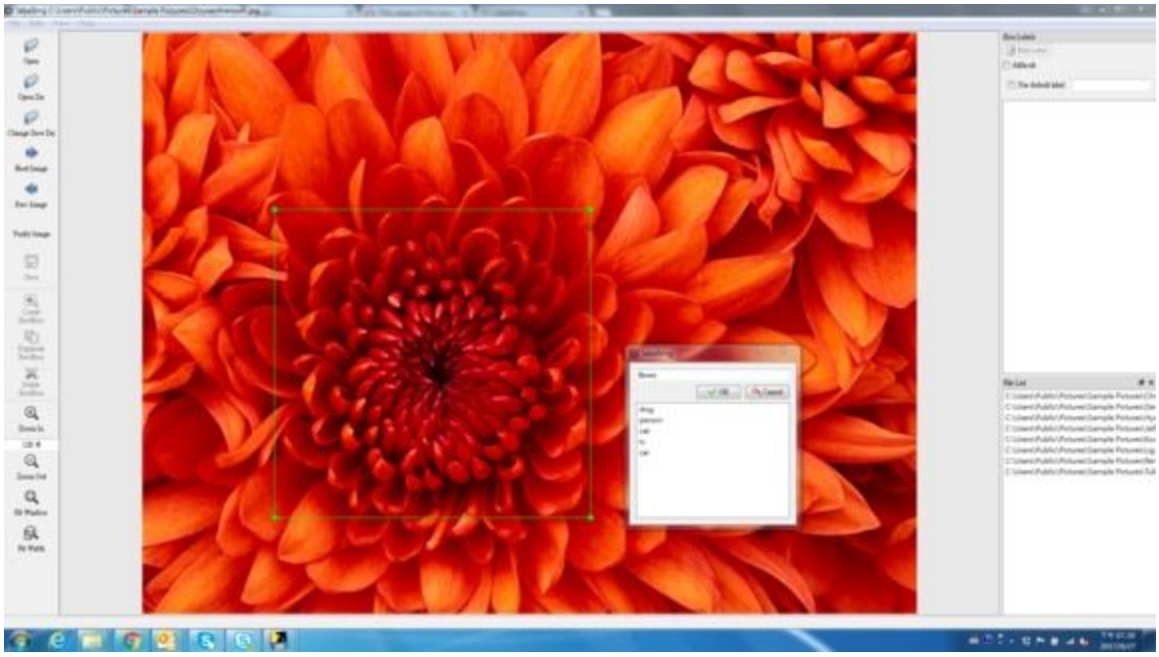


Figure #: LabelImg user interface

YOLO label format is very simple, just a .txt file with the following format repeated separated by newlines for each instance of a tagged object:

<object-class> <x> <y> <width> <height>

In our case, as we are only concerned with one object, gravestones, the object class will always be 0. Using this file as input to our program we can easily find the mask for each gravestone using a sufficient algorithm and export the whole thing as a .png mask, which is required for training. There are some additional optimizations that can be made with the bounding box selection as opposed to just simply clicking on the gravestone, such as the fact that if we see that the area for the gravestone goes outside the bounding box, the area is wrong. This

could help us fine tune our parameters on a grave-by-grave basis and get strong results everywhere on an image instead of just images where it is a specific brightness or darkness.

Finding the Gravestone Mask

The results we found from the prototyping tagging tool are strong, but not strong enough to replace the accuracy found in a machine learning model, so the algorithm described in the planning section has been relegated to be used as tooling to help us get all the gravestone masks and significantly speed up our labeling process for our training data.

The algorithm works as follows:
For a given pixel flood out, each one joining in all others orthogonally adjacent to it where the pixel passes a certain threshold.

Thresholding

A few options for the threshold were explored as it is the cornerstone of the whole algorithm, but the one chosen is as follows:

For pixel coordinates (x, y) and color of that pixel (r, g, b):

Brightness:

$$B = (r + g + b)/(255 * 3)$$

Normalized color:

$$C_n = (r/B,\ g/B,\ b/B)$$

For threshold:

$$T = abs(B_0 - B_1)/P_0 + dist(C_{n1}, C_{n2})/P_1$$

With adjustable parameters P_0 and P_1 for threshold T < 1

These parameters seem to have generally good results on all graveyards tested so far, but can be adjusted manually on a per-graveyard basis for better results.

This approach was chosen to minimize the effects of shadows on gravestones, as removing the brightness information helps minimize the effects of shadows on the flooding method. Some of the brightness information is still used in the threshold as many times the place where the shadow of the whole gravestone casts on the ground will be detected otherwise.

Thresholding Algorithm Explorations

Laplacian

Other parameters were tested for this threshold were explored, such as including the laplacian in the image in the threshold, which would bias the weighting towards where the image is changing color the most, in our case this would be the edges of the gravestones. The laplacian is defined as

$$L(x,\ y)\ = \frac{d^2I}{dx^2}\ + \frac{d^2I}{dy^2}$$

Where I is defined as pixel intensity, which was tested with both color difference, and normalized color difference, as described by C_n above.

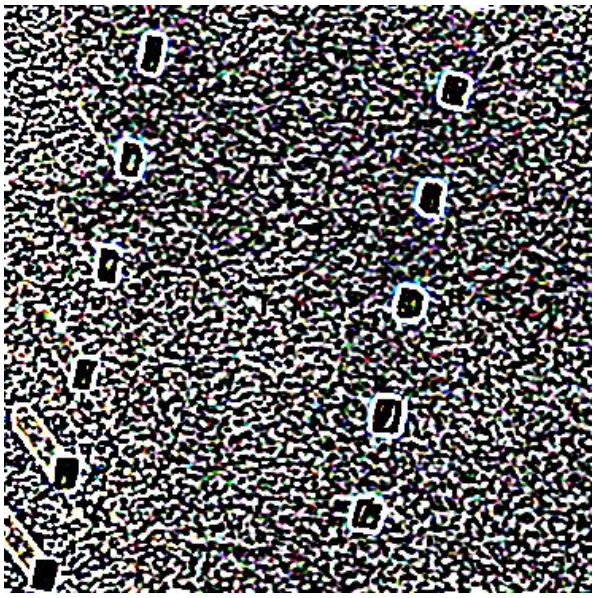


Figure #6: Laplacian image of the color difference of a graveyard image

Naturally, images come in discrete form with each pixel being a discrete piece of data as opposed to a function that might normally be differentiated, so the convolution kernels are commonly used to approximate the 2nd derivative of the image.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Figure #: Common 2nd derivative approximation kernels

Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm[19]

Once this laplacian image is found, it is used as a reference to bias the threshold to areas of less change.

For laplacian brightness L, and adjustable parameter P_L:

$$T_L = T + \frac{1-L}{P_L}$$

The threshold T_L can be substituted in the more general algorithm to the same extent.

[19] Tzutalin. (n.d.). Tzutalin/labelImg. Retrieved June 14, 2020, from https://github.com/tzutalin/labelImg

Canny

Along the same lines, the canny edge detection algorithm was also explored. Canny edge detection expands on the concept of the laplacian image, and uses the same concept of areas of high change along with an idea of edges being connected along their length.

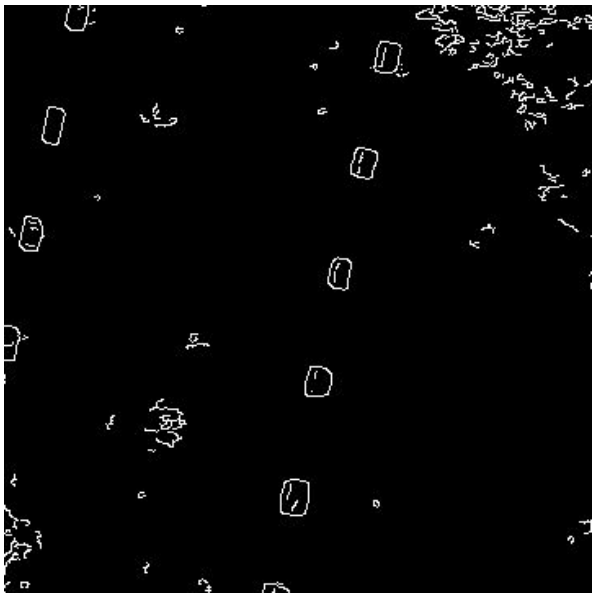


Figure #7: Left: Canny edge detection performed on a gravestone image

Right: Laplacian image of the same set of gravestones

The gradient of an image in a certain direction can be found in the same way as the laplacian image, by using a convolution kernel. However in this case since we need the vertical and horizontal components of the gradient to be seperated, we split the kernel into two, one for horizontal direction, and one for vertical.

0	−1	0
0	4	0
0	−1	0

0	0	0
−1	4	−1
0	0	0

Figure #: Example kernels for determining the vertical and horizontal derivatives of an image, respectively

The intensity gradient of an image is defined as:

$$G = \sqrt{G_x{}^2 + G_y{}^2}$$

With $L(x, y)$ being the laplacian image as defined above. With this gradient the angle of change in an image can be found using

$$\theta = \arctan(\frac{G_y}{G_x})$$

Once the angle of change for an individual pixel is found it is sorted into one of 4 categories based on precomputed values of arctan for certain angles.

Those categories being:

- vertical
- horizontal
- diagonally top right to bottom left

- diagonally top left to bottom right

Once this direction is found the two pixels opposing the pixel in question in that direction are checked. If the pixel in question is the maximum of the three then it is marked as an edge.

The next step in the canny edge detection algorithm is called Hysteresis Thresholding. In order for canny edge detection to operate, two thresholds need to be selected. One threshold at which, if edges have this level of gradient intensity, they are surely edges. In our case we will call it the maximum threshold. Another, where if an edge has less than this level of gradient intensity it is surely not an edge, and this will be called the minimum threshold.

Once all the edges are found in the image, edges that do not have an intensity gradient past the minimum threshold are discarded. Afterwards, pixels with intensity gradients higher than the maximum threshold are allowed through. Once a pixel is allowed through, all adjacent pixels are allowed through, even if they are not higher than the first threshold. Once this process is finished, the algorithm is complete.

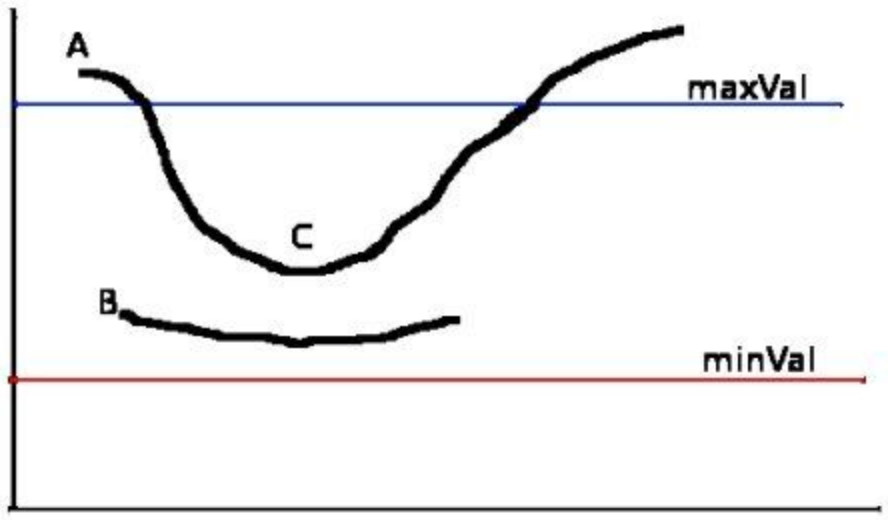


Figure #8: Visual representation of edge A - C being included in hysteresis thresholding, and edge B not being included

Source: https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html[20]

In our case, the canny edge detection algorithm was modified to only consider edges that formed a loop, as gravestones will always have edges on each side, provided that the thresholds used in the canny algorithm are set correctly. In the example below, this is the white edge, and this is as opposed to the blue edge, which does not form a loop.

Not only this, but the largest edge found for a circular edge removes all edges within it, as sometimes, an inner loop can be found in a gravestone, as seen in red in the example below.

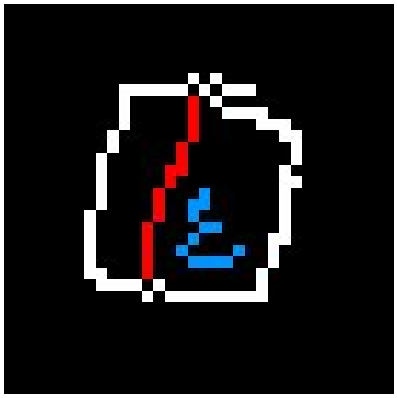


Figure #: Examples of edges that qualify as gravestone edges

White - included, Red/Blue - excluded

[20] Laplacian/Laplacian of Gaussian. (n.d.). Retrieved May 15, 2020, from https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

When included in the threshold, the edges found by this process were used as a hard limit for the gravestone, where any pixel labeled as an edge is immediately considered as not passing the threshold mentioned above.

This approach worked incredibly well when it worked at all, however it was very finicky when it came to different graveyards and the various maximum and minimum thresholds used in the hysteresis thresholding process, so ultimately it was not included in the current version as the results were already up to acceptable quality. If upon more heavy duty testing with a more robust data set it is found that the algorithm is too inaccurate, it could perhaps be included with measures to mitigate the issues seen with the thresholding, however, these were not explored due to time constraints. Along these same lines, another approach where all the parameters are optimized by using some hand-made reference grave mask(s) and then the application then finds the set of parameters that causes the algorithm to best match the grave mask could be very effective, as all the parameters were chosen by hand and could be very inefficient in the grand scheme of things.

Edge Expansion and Subtraction

After the thresholding is done, the pixels are put through a process where their edges are removed, then reapplied. More specifically, every pixel that does not have a pixel on every orthogonal side is deleted, and this process is repeated n times where n is some proportion of the number of pixels in the whole set.

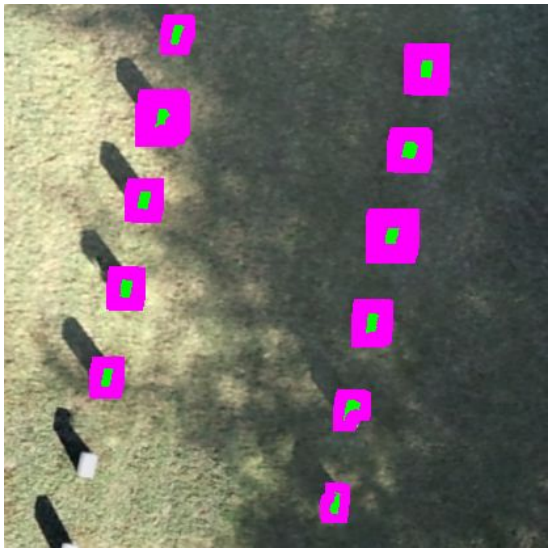


Figure #9: Exaggerated border addition/subtraction around gravestone areas

Once this is done, the border is “reapplied” by having each pixel add each of its orthogonally adjacent pixels to the selected set some amount of times. This process is performed to remove any unusual edges or irregularities on the gravestone area, along with making the selection area more generous if needed. In the figure at the start of the section, this reduced area is represented by the green area, and the expanded area is represented by the purple.

Finding the Gravestone Polygon

Once the approximate area of the gravestone is found, it can be used to assist in the finding of the polygons required in the training data. In order to do so our approach is pretty primitive.

From the set of gravestone area pixels, remove all pixels that are not on the edge. From an arbitrary starting pixel place a polygon vertex, then move clockwise along the edge, adding a

polygon vertex for an arbitrary amount of pixels until the starting pixel is returned to. We theorize an approach where you minimize the number of points that represent edge pixels within some linearity threshold may perform better but it has not been explored at the time of writing.

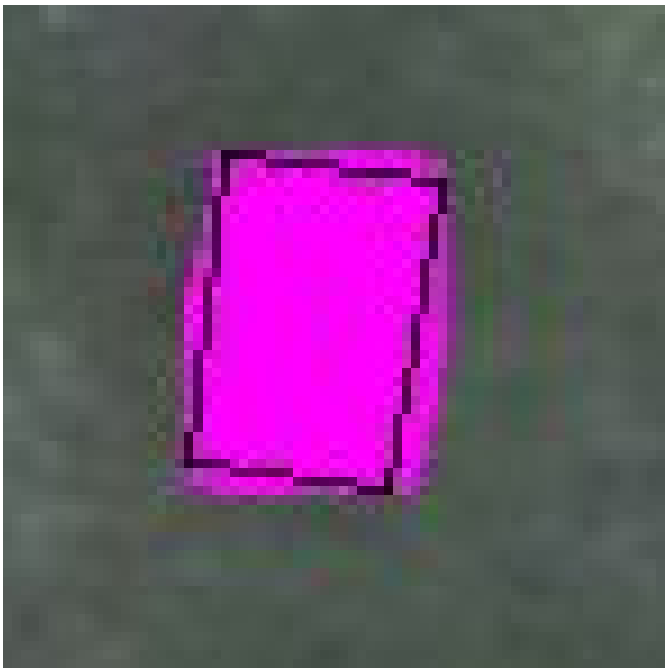


Figure #: Example area-to-polygon transformation

Expanding the Training Data

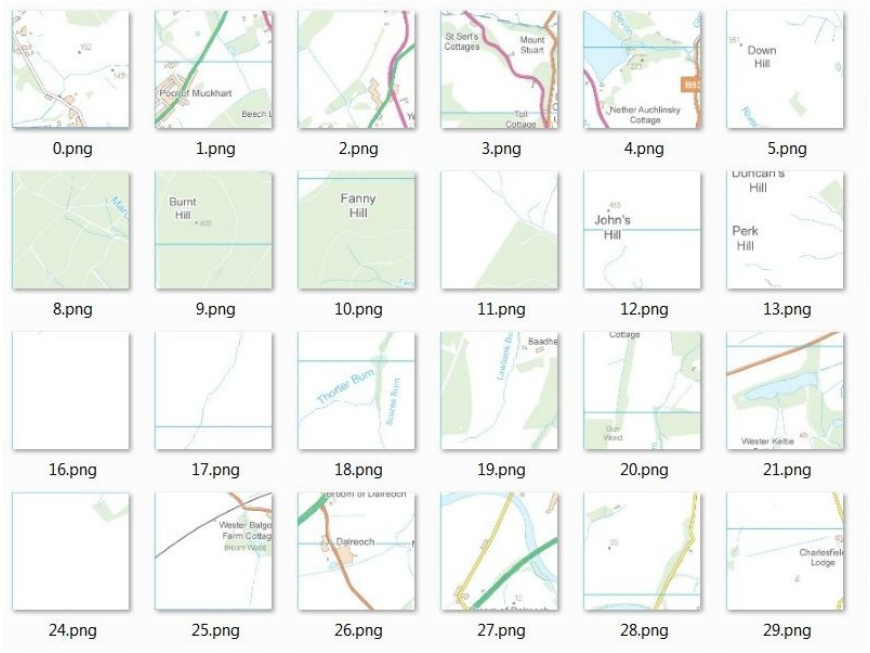
The amount of training data we need for this project is still far beyond what we are manually capable of doing, even with this tooling, in addition to being far outside the amount of image data we have on graveyards, so we have developed solutions to expand our dataset with what we have and cut down on tagging time. Machine learning for our problem requires 256x256 chunks

to learn from, and we can produce more than we should traditionally end up with by tagging the full size graveyard image supplied by Dr. Giroux initially, then feeding it into some process that takes that image, along with the polygon data outputted by the algorithm mentioned above, and the lat/long coordinate data supplied by Dr. Giroux and cuts it into various chunks that can later be fed into the system.

Figure #10: Example 256x256 chunks selected from a singular larger image

Not only that but these images can be cut in infinitely different ways, each potentially overlapping. Once this image has been spliced and the coordinate and polygon data has been transposed to the size of the cutout, the data can be further multiplied by rotating, mirroring, and resizing the 256x256 chunks along with larger sections of the image. Each time one of these transformations is applied the data once again has to be transformed to match, which is easily the most time consuming part of this problem. We had explored using a tool called GDAL2Tiles covered in another section of this document that does a lot of the transformations for us but

ultimately it was overkill for what we needed and the transformations were not complex enough to justify the additional overhead.



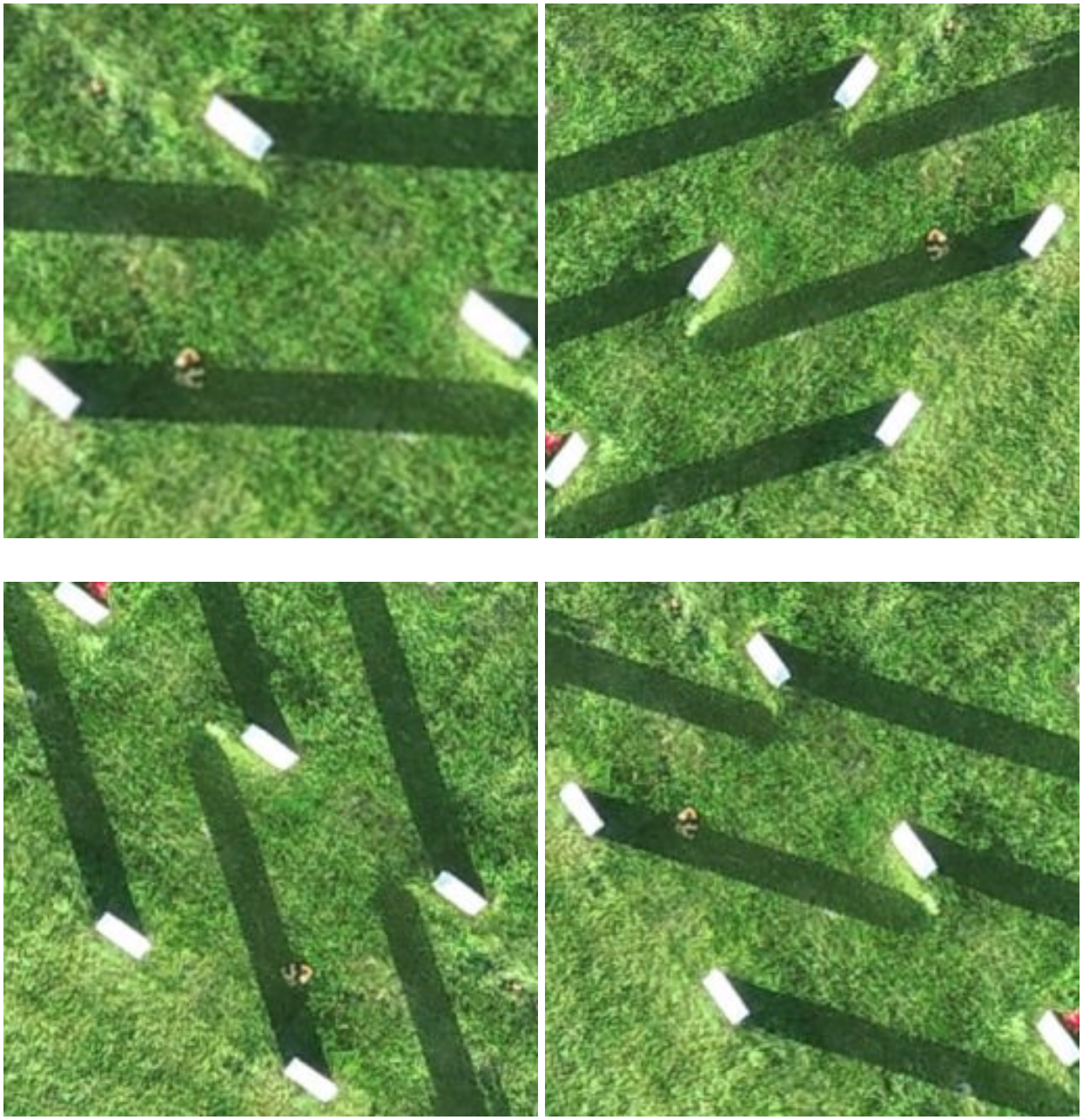


Figure #: 4 example transformations on a single image

Addendum: Concerns With Training Data

There are a lot of concerns to be had with the training data that might make them less suitable than is required and are difficult to mitigate and understand without first putting the system in practice.

1. Training the model off of the polygon coordinates provided from the training will probably lead the model towards seemingly arbitrary specification about what we are looking for, as there are a large range of viable coordinates for a given gravestone polygon considering how general the shape could be. We may have to reduce the specificity of the coordinates in order to lead the model towards a more general and easier to achieve solution that still has the results we want.

Mitigation: Train the NN with masks instead of coordinate data, so that estimations can be judged by area overlap instead of arbitrary coordinates. Once this area is obtained the polygons can be found using the approach detailed above in the “Finding the Gravestone Polygon” section.

2. If all these solutions for duplicating and transforming our data are not sufficient, we can turn to google maps for additional data, but there are some concerns with the resolution of the images being so significantly lower than what is supplied through Dr. Giroux’s drones, it may not be what we need.

SpatiaLite Consideration

SpatiaLite is a library that extends SQLite, allowing for the database to include geometric and geographic information (such as polygons and other data types found in GeoJSON files)[21].

One of the main benefits to using this library in conjunction with SQLite is that it allows for easy export of the database to a GeoJSON file, which is one of our project requirements. We could write our own way to do this, however SpatiaLite databases do have cross-compatibility with a lot of other major geographic softwares (for example, QGIS).

One of the drawbacks to using this library is that it may complicate moving information from the machine learning components to the database. It also makes development of the data entry screen more complicated, as the person tasked with that will have to also learn SpatiaLite, and their documentation is not well-maintained. Learning SpatiaLite involves a lot of sifting through outdated or obsolete documentation to get to the parts of it that are still relevant to the current version. Additionally, SpatiaLite is licensed under an MPL/GPL tri-license, where we are free to choose the best-fit license between MPL, GPL, and LGPL for our project. Using it would be something to discuss with our sponsor if we decided it was beneficial to our project to use it, as the source code for the project would have to be released.

---

[21] Furieri, Alessandro. "SpatiaLite." SpatiaLite. March 03, 2017. https://www.gaia-gis.it/fossil/libspatialite/home (accessed July 08, 2020).

After discussing the pros and cons of using this software, we decided that it was not worth the extra time investment to use SpatiaLite. The documentation for SpatiaLite is not well-maintained, so having everyone become familiar with it would take a while, longer than just creating a way to export to GeoJSON from the database ourselves. Also, the cross-compatibility with other programs becomes redundant since we are exporting cemeteries to GeoJSON format, which is also cross-compatible with the same programs (and more!) that SpatiaLite databases are compatible with.

GDAL Library

The Geospatial Data Abstraction Library (GDAL) is a library created in order to read/write geospatial data formats[22]. This library is used in some big-name applications, such as Google Earth and QGIS.

Overall, we determined that this would be an important library to use for our project. Since we deal with geoTIFF images and the coordinates embedded within these files, the GDAL library is instrumental in retrieving this information programmatically, and also embedding this information within other images. This is helpful in a lot of the aspects of our project, due to the geospatial requirements. Getting and setting coordinates using this library will help in our

[22] GDAL/OGR contributors (2020). GDAL/OGR Geospatial Data Abstraction software Library. Open Source Geospatial Foundation. https://gdal.org (accessed July 8, 2020).

annotation of the data we already have, allowing us to train the machine learning algorithm more effectively.

Some of the more useful aspects of the library include the gdalinfo command and the ability to get coordinate and spatial reference system information from geoTIFF and accompanying world files. The gdal2tiles command turns larger images of an area into zoomed-in map tiles with associated x and y coordinates.

GDALinfo

Using this will show the embedded geospatial information from a geoTIFF image. There are a lot of parameters for this command, but the most useful one to our project is -json. This will display the output in GeoJSON format instead of just listing the information it outputs.

However, running just the command along with the geoTIFF image will list the spatial reference system being used for the image, coordinates of the corners of the image, along with other metadata information. The spatial reference system seems to be important, as if we wanted to embed geospatial information into images ourselves we would need to declare this in each image. It tells what system the image was taken in, so we can calculate projections onto it accurately. The corner coordinates of the image are also important, as we can use this to determine where exactly the headstone we are detecting sits in the coordinate system, and calculate its latitude and longitude from this.

GDAL2Tiles Research

GDAL2Tiles is an open-source python script that creates small tiles and metadata from existing larger images, and Google Maps images. These small tiles follow the Tile Mapping Service (TMS) specification, which means these images will be compatible with other geographic and cartographic programs that follow the same specifications.

The benefits of using this as opposed to scripting our own version are long. Our sponsor Dr. Giroux is already familiar with GDAL2Tiles, and has used it to turn her drone images into mappable tiles. These tiles are then stitched together, creating a larger high-quality image of the entire cemetery with the correct geographic data embedded. It also already exists, which would save us development time and allow us to fine-tune the machine learning aspect of the project more which leads to more accurate detection of headstones.

Considering we are training from scratch, we need to annotate a lot of images for training purposes, so using existing code to create these images instead of scripting our own frees up more time to tackle that task. It also allows us more time for bug-fixing and ensuring that the coordinates calculated for each headstone are precise and correct. In addition to this, the GDAL2Tiles program also stores and names the images with their x and y coordinates, which is important for mathematically determining the location of headstones within the image.

There were very little drawbacks to using this script. We could write our own script to handle this, but it would take very valuable time away from annotating our training images and fine-tuning our machine learning algorithm. The program does do a few extra tasks, like creating

a directory for all the images and has the ability to tile images at different zoom levels. Our own script would be a tighter fit to our use case of just splitting images, but these other functions are to the benefit of our sponsor who uses them already.

Overall we determined that for annotation purposes, this software would be very helpful. It allows us to split up the cemetery images into images that contain a smaller amount of headstones, which increases the amount of training data we will have. This program also allows us to split up the images in different ways which will allow us to "cheese" our training data, meaning that we can get a lot more training images from the six cemetery images that Dr. Giroux was able to provide us.

GDAL Library with Python

There exists a GDAL Python API for our reference during this project[23], as well as a cookbook for common uses of the GDAL library within Python[24]. These two resources are instrumental in learning how to use GDAL with Python to create geometries, features, and read/write GeoJSON files. The first lists all functions and describes what they do, while the cookbook gives examples of them. The cookbook also describes common hangups and problems

[23] "GDAL/OGR Python API". May 7th, 2020. https://gdal.org/python/index.html (accessed July 27th, 2020).

[24] Erickson, J., Daniel, C., Payne, M., "Python GDAL/OGR Cookbook 1.0 documentation". https://pcjericks.github.io/py-gdalogr-cookbook/ (accessed July 27th, 2020).

in using the GDAL Python library and how to solve them. It also describes common misunderstandings that developers have when using the library and what to use instead.

OSGeo4W

The Open-Source Geospatial Foundation (OSGeo) hosts a collection of open-source geospatial programs and libraries that we thought worth looking into for our project. During this research phase, I installed everything using OSGeo4W as below because it allowed me to test out all the libraries I was researching at the time (SpatiaLite, GDAL, GDAL2). It downloads with a batch file allowing you to run scripts from all these things from the command line easily. I used this to familiarize myself with the commands and determine the usefulness of SpatiaLite, GDAL, and GDAL2 to this project before attempting to programmatically use them.

Also because we were just using GDAL2Tiles to aid with the annotation of our training images, I ran GDAL2Tiles with our larger cemetery images through the OSGeo4W batch file.

OSGeo4W Installation Guide

My steps to install this software are as follows:

1. Download installer from OSGeo4W
2. Click Advanced Install

3. Install from Internet, select a download site

4. Choose a root directory for the install

5. Select GDAL (and SQLite if you do not already have it) and GDAL2 libraries and python package

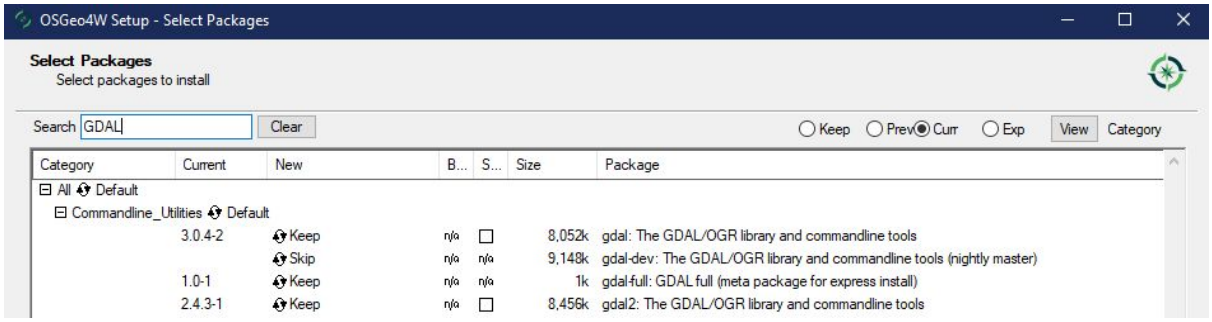


Figure #11: Selecting necessary packages to install GDAL and GDAL2 libraries and command line utilities.

   a. For this step, selecting GDAL and GDAL2 from the Commandline_Utilities section should be sufficient in getting the libraries you want for use with OSGeo4W. However if you want the python related scripts and bindings you will have to select those in the Libs tab.

6. Install the selected files

OSGeo4W with GDAL2Tiles User Guide

Running GDAL2Tiles went as follows:

Run OSGeo4W batch file

> gdal2-py2-env

> gdal2tiles.py --zoom=[#] [input-file] [output-folder]

Here, the maximum zoom level (according to the gdal2tiles.py script) is 32. This should be more than enough zoom to clearly see headstones, as I tested the program with zooms 20-30, and around 22-24 had the best quality in my opinion. Figure #12 displays the results of running GDAL2Tiles with the same image, with zoom levels 22 through 24.

Figure #12: Leftmost is zoom level 22, middle zoom level 23, and rightmost is zoom level 24.

I think that 23 had the best results. There were roughly 10-20 headstones per picture and most pictures included headstones. Anything larger than that will result in a lot of pictures of just grass and more cut off headstones. Also, the higher the zoom level the longer the output takes to make due to the amount of tiles the software needs to generate to cover the entire image. The downside to higher zoom levels, other than the increased output time, is that there are way more

images without headstones in them that we have to field through (a lot of the images are just grass or trees or road).

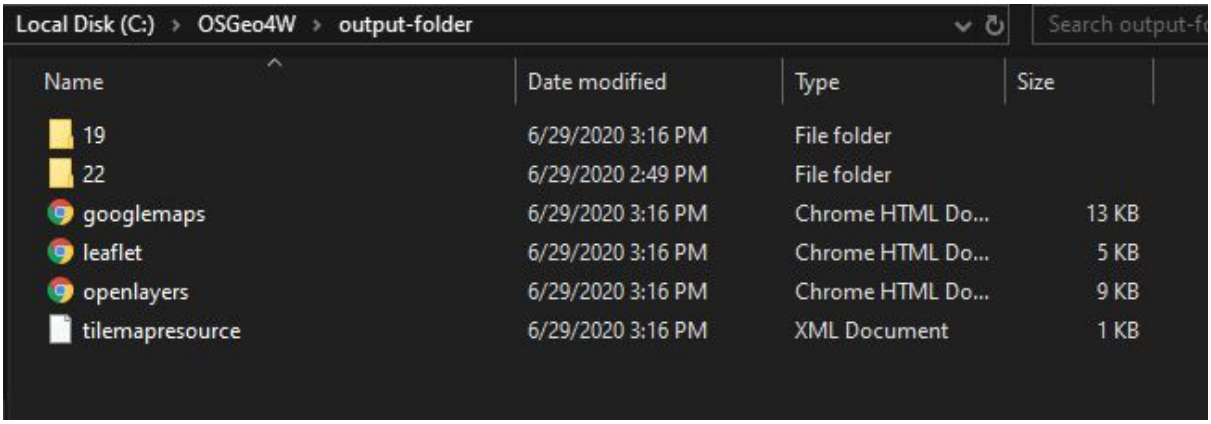


Figure #13: Contents of the output folder. The user had run gdal2tiles with the output image with zoom levels 19 and 22, as the folder names show.

This script outputs the images into the specified output folder. Pathing will look like output-folder\zoom-level\x\y.png. This means that the x and y coordinate of the image is the path and name of the image as output by the script (see figure #14 for an example of this). There is an algorithm to calculate latitude and longitude of the image from the zoom number, x, and y coordinates of that image (see figure #15). The most important thing to remember when using GDAL2Tiles is that the geoTIFF needs to have longitude and latitude information embedded, or at the very least have a world file (.tfw) associated with it in order for the script to access the coordinate information and label the tiles accordingly. Otherwise, the images will be tiled and labelled but the coordinate information calculated from them will not be correct.

Fortunately, Dr. Giroux was able to provide us with six different cemetery geoTIFFs and associated world files. So, with GDAL2Tiles and image manipulation techniques, we should be able to extract plenty of data in order to train our machine learning system.

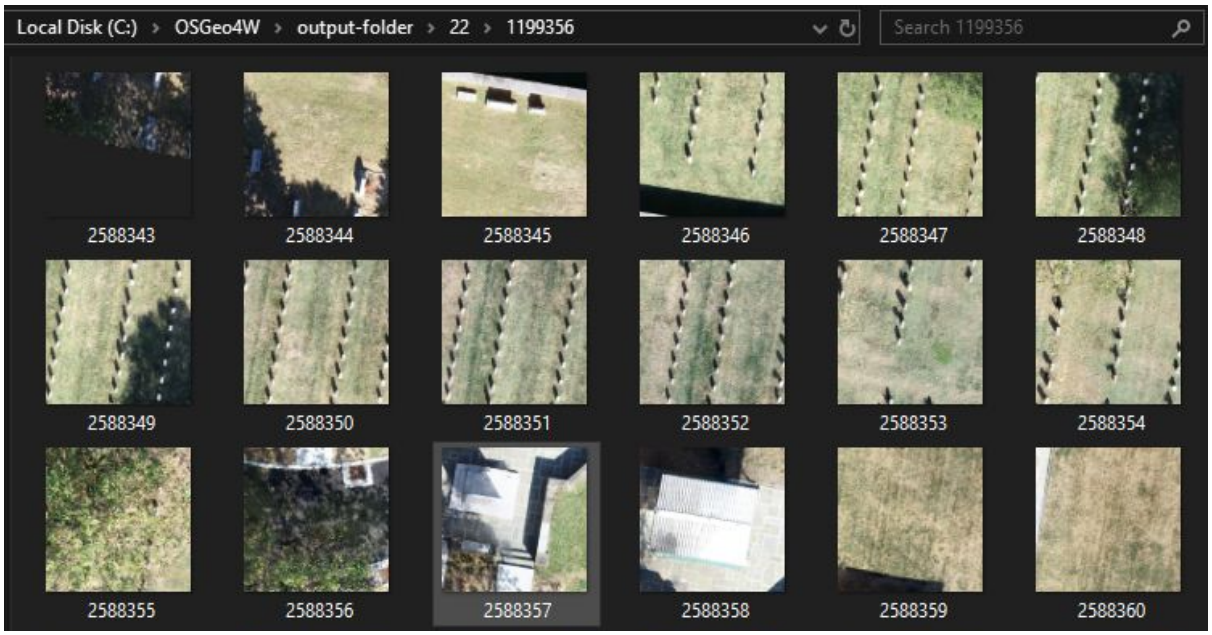


Figure #14: Navigating through the zoom = 22 folder shows folders labelled by x-coordinate, and opening those shows images labelled by y-coordinate.

Tile numbers to lon./lat.

```
import math
def num2deg(xtile, ytile, zoom):
  n = 2.0 ** zoom
  lon_deg = xtile / n * 360.0 - 180.0
  lat_rad = math.atan(math.sinh(math.pi * (1 - 2 * ytile / n)))
  lat_deg = math.degrees(lat_rad)
  return (lat_deg, lon_deg)
```

This returns the NW-corner of the square. Use the function with xtile+1 and/or ytile+1 to get the other corners. With xtile+0.5 & ytile+0.5 it will return the center of the tile.

Figure #15: Algorithm for gathering latitude and longitude information from gdal2tiles output, from the OpenStreetMap wiki.[25]

Licensing

GDAL is licensed under the MIT/X style license[22]. In summary, the software is open source and developers have free use of it as long as they include the licensing information "in all copies or substantial portions of the Software"[22] and also recognize that there is no warranty or liability attached to the software and the Licenser cannot be blamed for anything going wrong in projects the libraries are used in. Thus, we should include a copy of the license within our files considering that Dr. Giroux requested all the necessary libraries and softwares be packaged with the product.

SQLite is under the public domain, meaning it is free to use for anyone who wants to use it and uncontaminated by licensed code[35]. This will cause no issues during our project. There is actually a purchasable license for companies that require proof of license for all the software they use or do not recognize public domain, but this is unnecessary for our project. We will not need to include any licensing information from SQLite in our code repository.

---

[25] OpenStreetMap Wiki contributors, "Slippy map tilenames," OpenStreetMap Wiki, , https://wiki.openstreetmap.org/w/index.php?title=Slippy_map_tilenames&oldid=2007534 (accessed July 8, 2020).

The geojson Python library is licensed under the 3-Clause BSD License[26]. We should have no issues using this library because of the license. The only stipulations on using software with a BSD license is that if the source code or binaries ever gets redistributed, we are to include the copyright information of the software and a disclaimer of liability of the original creators of the licensed software. These are things that can be included with the files for our project very easily, so we should include a copy of this license within our repository also.

TensorFlow is licensed under the Apache License 2.0. This is very similar to the MIT license that the GDAL library has. A great summary of this license is as follows: "The Apache License 2.0 is a permissive license … [it] provides an express grant of patent rights from contributors to users. The conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications and larger works may be distributed under different terms and without source code."[27]. This means we are granted a license to any patent that covers this software, and do not have to worry about infringing on any patents. In using TensorFlow, we should include the licensing information within our repository, similar to the above softwares.

---

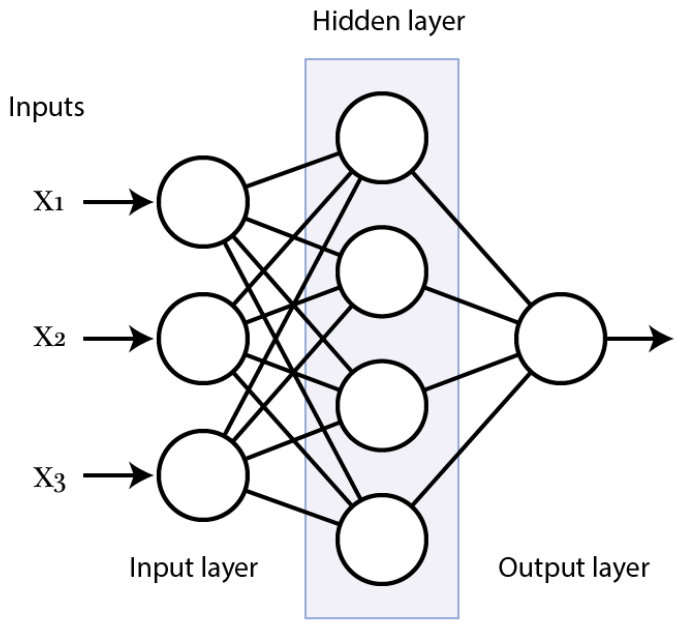
[26] Gillies, Sean, "geojson 2.5.0", August 9th, 2019. https://pypi.org/project/geojson/ (accessed July 27th, 2020).

[27] Owoyemi, Joshua, "Open Source Licenses in Machine Learning and Self-driving Car Projects". Medium. November 6, 2017. https://medium.com/@tjosh.owoyemi/open-source-licenses-in-machine-learning-and-self-driving-car-projects-e6fe2bce8641 (accessed July 28th, 2020).

Machine Learning

A Quick Overview of Neural Networks

Neural networks are a computational model inspired by the inner workings of the brain. A neural network involves layers of connected nodes, or neurons. Like the neurons of the brain, the neurons receive input from other neurons, and upon reaching an activation threshold, send some output to other connected neurons. Put simply, a neural network is a mathematical model that takes some input and ultimately map it to some output. A neural network can learn from data how to weight the connections between the neurons in order to become better at predicting the output we want. Neural networks have been employed to solve problems of object detection, image classification, speech recognition, and many others.



Simple example of the modeling of a neural network with inputs, hidden layer, and output

Source: Nick Evans

A neural net will usually consist of an input and an output layer, with some amount of "hidden layers" between them. In a densely connected layer, each neuron receives input from all the neurons in the previous layer and passes its output to all the neurons in the next. Each neuron will multiply its input by a parameter known as a *weight*, and add a parameter known as a *bias*. To introduce a non-linearity to the model, an activation function is applied to the result. The output of the activation function becomes the neuron's output. Data flows through the network until it produces its final output.

Through *supervised learning,* neural nets can use an algorithmic process to learn from training examples and become useful models when applied to real-world tasks. As the network is exposed to data, it can adjust its weight and biases to create a more accurate model. By training a neural network on enough examples (often thousands to millions), it can learn how to map its input to the expected output and become useful for making predictions on new data.

Forward propagation

The training process of a neural net, at its most basic, involves two steps. The first step, *forward propagation*, involves feeding the input through the layers of the network to produce the model's output. The neurons in each layer receive input from the previous layer and perform a computation on it based on their trainable parameters. The neurons of each layer apply their weights and biases, and an activation function is applied to result, which becomes the input into the next layer.

There are many possible choices of activation functions. For some time, the sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}}$, was a common choice. But in more recent research, the Rectified Linear Unit (ReLU) function, which is $f(x) = max(0, x)$ has gradually become the more common default choice (due to its relative computational simplicity as well as keeping a steadier gradient for large values.) A non-linear activation function ensures the results of each layer are not merely a linear function of their inputs. Since the class of linear functions is closed under composition, a

neural network will reduce to simple linear regression model if we use a linear activation function or forgo using one altogether.

After data is processed through all the hidden layers, it is sent through the output layer, which may consist of a single node or several depending on our problem. A price prediction model might have one output, but a model built to classify images might have one for each class it is trained to identify.

The difference between the expected and actual output of the model is computed by a *loss function*. Depending on the kind of task we want our model to solve, there are a variety of loss functions to choose from. In a simple regression problem, we might use the Mean Squared Error function. The Logarithmic Loss function is commonly used for classification problems. Whichever we select, our model will then attempt to optimize its weights and biases to minimize this loss function as best it can.

Back propagation

The second step, *back propagation*, involves moving backward through the layers, computing the gradient of the loss function with respect to all the model's parameters. The gradient is then used to adjust the training parameters of each neuron with the goal of minimizing the loss.

In an algorithm known as *gradient descent,* the model trains all its parameters based on the derivative of the loss function, adjusted by a *learning rate*. The choice of learning rate is also an important consideration when training a model. Too low, and the model will take too long to train. Too high, and the model will overcorrect.

These two steps are repeated for all the training examples for some amount of training epochs. With a well-designed model architecture and enough training data, even simple neural nets can produce impressive results.

Convolutional Neural Networks (CNNs)

Convolutional neural networks[28], commonly abbreviated as CNNs, can produce better results faster than simple artificial neural networks when working with images as data, as in our project. A naïve implementation of a neural network on image data might involve processing the value of each pixel as a different input unit. This results in two immediate issues. The first is the vast computational overhead involved in such a large number of input units. Large images would produce networks too complicated for practical purposes. The second is that the results of the network would not be translation-independent, i.e., pixel values for an object in one corner of an image would be treated as entirely different input that the same object in a different pixel location.

---

[28] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Rather than training a model based on specific patterns in the raw pixel data, CNNs allow a model to better learn to extract and identify the high-level features of an image and generalize more effectively to the wide variety of data present in real-world images. The key is the addition of *convolutional* and *pooling* layers that filter the input data in a process known as *feature extraction.*

Convolutional Kernels

Instead of processing an input for each pixel of an image, infeasible for very large images, convolutional layers filter an image down to major features upon which we can train our model. Convolution involves the application of a filter, known as a kernel, over the input data. Convolutional kernels take advantage of the intuitive fact that close pixels in an image are more related than distant pixels to help a model generalize.[29]

The kernel is represented by some matrix of values. The kernel is centered over a pixel, and the pixel's filtered value becomes the summation of the results of multiplying the pixel's value and its neighbors' values by their respective elements in the kernel. The kernel is then shifted by a predetermined amount known as the *stride* and re-applied, until it has passed over and filtered the entire image.

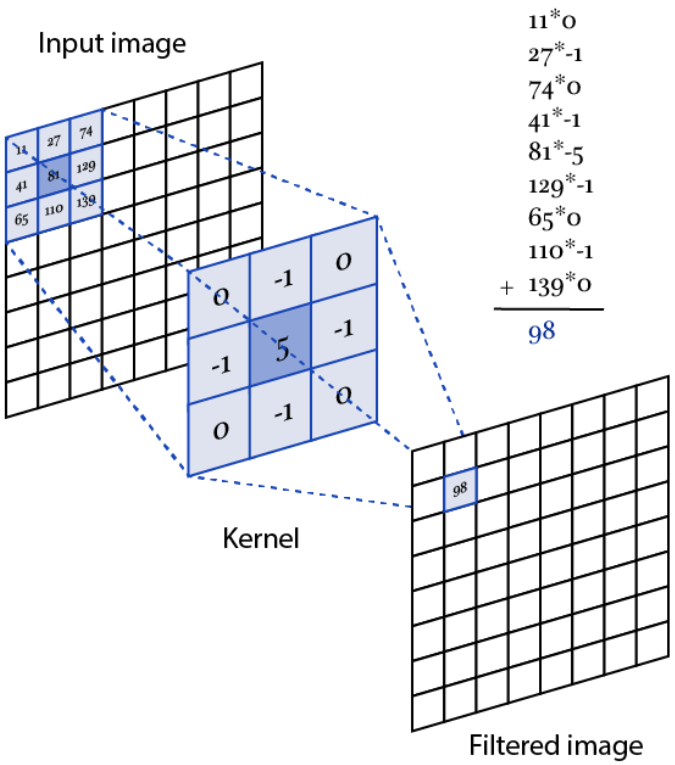
---

[29] Flagel, Lex et al. "The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference." *Molecular biology and evolution* vol. 36,2 (2019): 220-238. doi:10.1093/molbev/msy224

This algorithm for image filtering has been commonly used in image editing software like Adobe Photoshop using pre-created kernels to produce effects such as sharpening or blurring. However, in a CNN, randomly-initialized kernels will be trained to produce filtered images that are useful for a model to generalize image features. We will go into this in further detail, but first, there are some considerations to be made when constructing the convolutional layers in a CNN.

Selection of stride size is an important consideration. A stride of 1 is common, retaining the most information possible from an image. A stride of 1 keeps a model accurate by making it more translation-independent, i.e., able to recognize features regardless of their position in the image. However, a larger stride size is sometimes used to reduce computation especially with larger kernel sizes.

1- and 2-dimensional convolutions are possible, but since color images have red, green, and blue channels, the kernel is usually a 3-D matrix with a depth of 3. Each layer of the kernel is applied to its respective color channel in the input when working with RGB images.

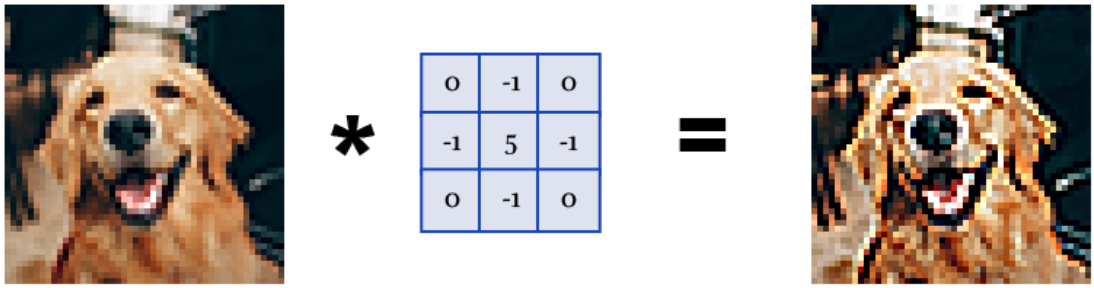


The calculation performed for a single pixel value by for a given kernel

Source: Nick Evans

Additionally, there is the issue of edge pixels. With kernel sizes larger than 1x1, part of the kernel will be overlaid outside the bounds of an image when working along the edges. There are a few techniques to address this. We could simply ignore the edge pixels, but this results in lost information. This can be acceptable for small models, but for deep CNNs with many layers, this becomes increasingly problematic as more data is lost in each layer. More commonly, many

implementations will pad images with zeroes, reflect the values around the edges, or wrap the image with pixels from the opposite edge.



Application of a "sharpening" kernel over a sample image

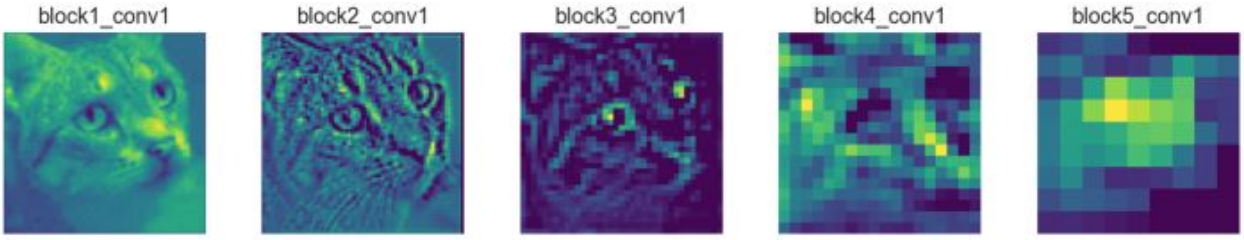
Source: Nick Evans

Different values in a kernel filter will produce different effects in convolution. For example, some kernels can sharpen an image. Other values can create a blur effect. Some kernels exaggerate certain features, such as horizontal or vertical lines. A convolutional layer in a CNN will typically contain many randomly initialized kernels. During training, the network will tweak these values as it does the weights and biases of neurons in dense layers. The goal is to learn filters that emphasize the high-level features that are useful to a particular model, such as edges, lines, or specific shapes.

Pooling

Convolutional layers are often followed by a pooling layer. Pooling layers reduce the size of the image, lowering the computational load on the model, and further extracting dominant image features.

Pooling works by passing a filter of a certain size over an image, creating a new image whose pixel values are either the maximum (in max pooling) or the mean (in average pooling) of the pixels covered by the pooling filter. For example, a max pooling layer with a filter size of (2, 2) will produce new images one-quarter the size of the original, with each pixel value being the maximum value of a 2x2 region in the original.



Source: Arden Dertat, Towards Data Science

Above we can see an example of an image processed over five convolutional and pooling layers, in order to visualize exactly what this kind of process looks like under-the-hood. Often,

convolutions produce strange effects that are not immediately intuitive to a human observer, but are in face useful abstractions for our model to understand the data it's working with.

Other Uses for CNNs

As an aside, CNNs are commonly used when working with images as data, but they have seen surprising success applied to tasks in other areas as well. CNNs have been used in speech recognition and voice synthesis. For example, the synthesized voice of Google Assistant is based on Deepmind's WaveNet CNN. Researchers in the field of biotechnology have employed CNNs to generate models for disease identification in genomics, and to infer genetic and evolutionary history from population-scale genomic data sets.

However, CNNs are typically most effective for problems like image classification, image improvement, and object detection.

Overfitting

A common challenge faced in machine learning is the issue of *overfitting* the training set. Over many training epochs, a model faces the risk of becoming so well-attuned to its training set that it no longer generalizes well to new data. Rather than learning the general features we want

our network to recognize, it begins to memorize the specific examples in its training set. At this point, while accuracy on the training set may continue to increase, the model's performance on new data may begin to degrade, until the model becomes practically useless.

```
# MODEL DEFINITION
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(2)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Total params: 3,453,634
Trainable params: 3,453,634
Non-trainable params: 0
_________________________________________________________________
```

Sample model using the Keras API for TensorFlow

The following graph illustrates overfitting of a neural network implementation designed to classify images into two classes: "cat" or "dog." This simple CNN consists of four blocks of convolutional layers followed by max pooling layers, then a densely connected layer of 512 neurons, and finally a dense layer with two outputs for classification. This model was trained on 2000 examples from the "Dogs vs. Cats" dataset provided by Kaggle over 50 training epochs.

Demonstration of overfitting on a simple CNN implementation

Overfitting can be spotted during training by utilizing a *validation* set, separated from the training set, to test the network's performance after each training epoch, as seen above. It is important that the model is not trained based on the validation set, so that it remains a useful estimation for how the model will perform on new data. If the accuracies on the training set and the validation set begin to diverge, we know that overfitting is occurring, and we can stop the training. This technique, *early stopping*, ensures we reach our model's potential without overfitting. There are many other techniques we can employ for our purposes to address the issue of overfitting.

Image Augmentation

Perhaps the most obvious way to address overfitting is to simply use a larger training set. A model that can train against a larger variety of examples will be able to generalize better and make more useful predictions. However, when no further examples can be naturally acquired (and indeed, we are quite limited in the training examples available to us for this particular application), the technique of image augmentation can be employed to artificially enlarge the training set.

By applying random image transformations, such as rotations, stretches, zooms, or reflections, any training set can be significantly expanded to provide more data to a model. By applying these transformations during training, we can in fact ensure that the model never sees the exact same image twice.

Dropout

Another simple but effective technique to delay overfitting is *dropout*. This technique involves giving every neuron a set probability of randomly disabling. A disabled neuron does not

participate in the forward pass and it is not involved in back propagation. In effect, it is removed from the model altogether. Therefore, each training example is passed through a different subset of the overall network architecture each epoch. This prevents neurons from relying on other, overly weighted neurons and encourages the development of a more flexible and robust model since the network cannot rely on specific neurons in combination.

Regularization

Regularization is another method to reduce model complexity and address overfitting. Regularization involves adding an additional penalty to our model's loss function for large weights.

The most common are known the L1 and L2 penalties. The L1 penalty adds a linear value related to the absolute value of the weights in our model. The L2 penalty penalizes our model based on the squared magnitude of our weights. The idea is to disallow our model from making weights arbitrarily large, therefore making it more general and less prone to overfitting, similarly to the dropout method.

The previous network with randomized image augmentation and 50% dropout employed

By employing the some of the above techniques on our example model, we can see a dramatic improvement. First, we use the image data generator class provided by TensorFlow's Keras API to add random image rotations and zooms. Then, we add a 50% dropout probability to all the neurons in our hidden dense layer. Running the training again, we can see the model's performance on the validation set continue to mirror that of the training set. In our original model, we experienced overfitting after just 10 training epochs, but now performance remains good over 50 epochs without any changes to our overall network architecture.

Transfer Learning

Designing and training new neural network models, even for intuitively simple tasks, can be complex and time-consuming. Effective models require large amounts of training data and training these networks can take a considerable time (days or weeks). *Transfer learning* is a technique where a pre-trained model can be re-used for another task.

Using existing models as a jumping point for new networks can produce fantastic results in a short time frame. Many models available for this purpose from resources such as TensorFlow Hub have been built by large teams of researches over many years with specialized equipment and massive amounts of data. Transfer learning allows us to build off their knowledge and work to solve new problems.

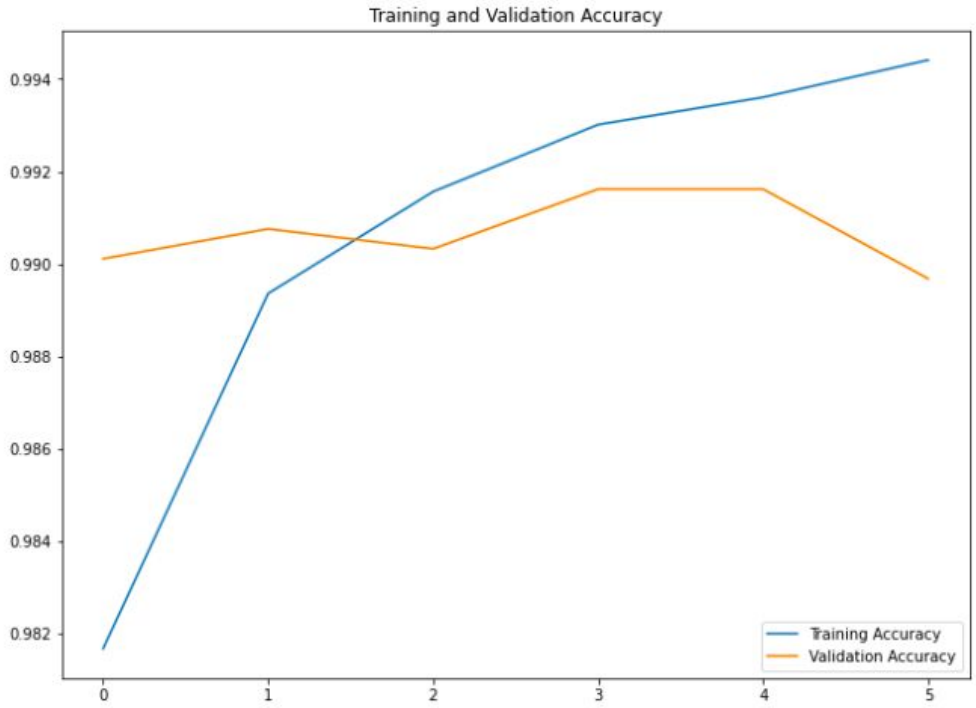
Google's Inception model and Microsoft's ResNet model are examples of pre-trained networks highly suited to problems that expect images as inputs. These networks, trained to classify images into thousands of classes, are already well trained at image feature extraction.

```
URL = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/2"
feature_extractor = hub.KerasLayer(URL,
                                   input_shape=(IMAGE_RES, IMAGE_RES,3))

feature_extractor.trainable = False

model = tf.keras.Sequential([
  feature_extractor,
  layers.Dense(2)
])
```

Simple example of importing and preparing a pre-trained model from TensorFlow Hub, with a new two-node classification head attached

TensorFlow's Keras API makes it simple to import existing models into our own project. By setting the model's *trainable* field to *False*, we can "freeze" the model, and attach new layers to the end specific to our task. We freeze the weights on the model so they are not altered during training, because at first the error in training will be very large, which will cause sweeping changes to our imported model. Instead, we will use what the model has learned, and train only our extension to build off it.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```



Here, our model utilizing transfer learning achieves over 99% accuracy on the same cats and dogs data as our earlier example model after only 5 training epochs.

Object Detection

A more specific problem under the umbrella of image classification with CNNs is the issue of *object detection*.[30] Classifying images is a solved problem in the field of computer vision, with modern neural network implementations able to meet or exceed human

[30] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017.

performance. However, the goal here is to identify regions in photographs were objects are present, and then to classify them.

An example of object detection, with objects labeled and bounding boxes determined

Source: TensorFlow Blog

This problem proves to be more challenging for a variety of reasons. First, we need not only to identify objects, but determine their size and location (a bounding box.) A classic "sliding window" approach proves to be very inefficient. Objects may take up any portion of an overall image, and the aspect ratio of their bounding boxes can vary infinitely as well.

An additional challenge is the variable number of objects possible in an image. The need to accurately identify all occurrences of an object in an image is obvious for many applications

of object detection, such as crowd counting. It is especially important for our purposes here as well. The unknown number of outputs needed for the model presents additional complexity.

Finally, there is the challenge of combining the two seemingly different problems, classification and localization, into one model. There have been several increasingly effective techniques proposed in recent years.

Region-Based CNNs

Many methods of object detection rely on a two-step process. The first step is to propose regions of interest in the image. In the second step, these regions are then fed to a traditional CNN architecture for classification.

An early major example of this was in the Region-based CNN (R-CNN) model developed by Ross Girshick in 2014. This model extracted possible object regions using the Selective Search algorithm, and then extracted the features and classified the images with a CNN. While this approach was effective, it was very slow.

The following year, Fast R-CNN and Faster R-CNN were developed, with tremendous (over 100x) increases in speed and performance. Faster R-CNN replaces the computationally expensive Selection Search algorithm for finding object regions with a Region Proposal Network (RPN) which is trained to identify object regions that are pooled into Regions of Interest and passed to a classification network.

The TensorFlow 2.0 Object Detection API uses an implementation of Faster R-CNN. Since our project is using TensorFlow and its APIs, this implementation is of special interest.

The first step of Faster R-CNN uses a network pre-trained in image recognition (such as ResNet). Instead of using the final output of this model, however, we use the convoluted output of an intermediate layer in the network. A set of fixed bounding boxes, known as anchors, are overlaid on each pixel of this output. The RPN then adjusts these anchors and assigns each one an "objectness" score. The RPN ultimately outputs a set of proposals for object bounding boxes. These boxes are then pooled and passed to a R-CNN which has the task of classifying each proposal as a specific object (or a background) and further adjusting the bounding boxes to better fit the object.

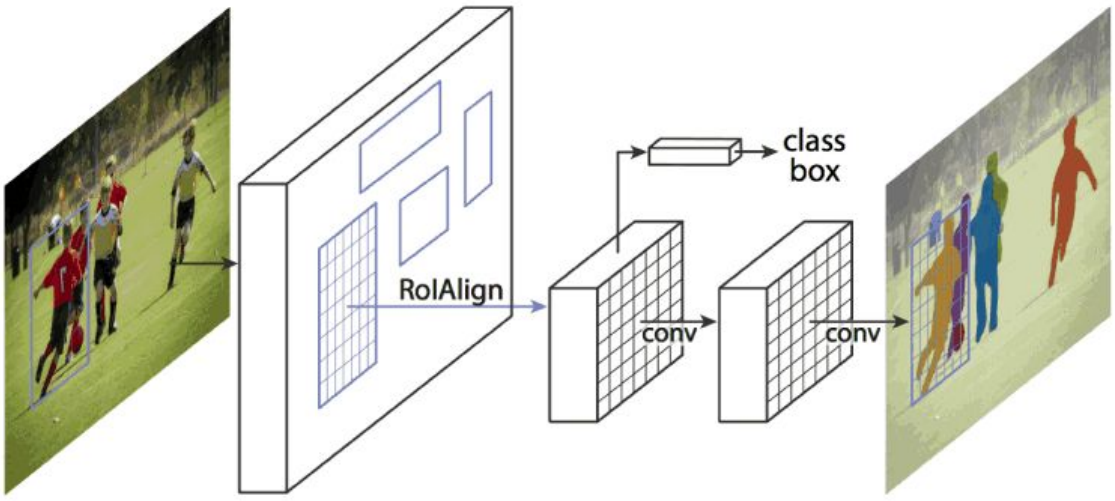
Going Further With Mask R-CNN

Example polygon from Dr. Giroux's initial pitch

Source: Dr. Amy Giroux

Our task is more than a simple bounding box, however. The goal here is to define a polygonal shape that outlines the tombstones seen the aerial photographs. The shapes can have four or more sides, depending on the shape of the tombstone. Most are rectangular, but some are of more irregular design. Since a bounding box is defined only by a width, height, and center coordinate, it will not suffice.

Mask R-CNNs are a model that is an extension of Faster R-CNN. These networks go one step further, and produce a pixelwise mask for the identified objects. In order to train a Mask R-CNN, we need training data that has been tagged not merely with a bounding box, but with a mask for the object of interest. Our tool for highlighting the tombstones should be able to be modified to fulfill this task.



The Mask R-CNN framework for instance segmentation

Source: Mask R-CNN, Facebook AI Research

So, in our final model, a Faster R-CNN uses a convolutional neural network to extract feature maps from the images we need to work with. These feature maps are passed to a Region Proposal Network that decides on good candidates for image bounding boxes. These bounding boxes are passed onto a Region of Interest pooling network that consolidates the bounding boxes using intersection over union. Finally, in the step novel to Mask R-CNN, the model generates a bitwise mask for each of the identified objects.

Provided we can tag enough data sufficiently, this method and model will allow us to train a network to give us precise segmentation[31]s for defining polygons that outline each tombstone in our data.

Our Training Process

Now to summarize the outline for our training process this summer. (Note that this section will be subject to change and will be updated.)

We will be using TensorFlow 2 as the framework for our project. Its Keras API simplifies a great deal of work on our end. TensorFlow Hub will also allow us to easily import various CNNs pre-trained for feature extraction to utilize for our purposes. We will be able to use

---

[31] Girshick, Ross, et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014.

multiple different models and select the one that offers the best tradeoff between speed and performance. TensorFlow 2 also has a fairly new but robust Object Detection API from Google that utilizes Faster R-CNN and Mask R-CNN models to jumpstart our own image segmentation.

Some sample training examples we will be working with

Source: Dr. Amy Giroux

Our training data will be taken from aerial photographs of national cemeteries taken by Dr. Giroux herself. We will be segmenting these images into a set of images with a dimension of 256 by 256 pixels each. We will then be tagging and masking the individual tombstones in each of our training examples.

Tagging of our data presents a challenge: we will end up with hundreds or thousands of examples that will need to be tagged by our six-person team. Often times in large machine learning projects, this menial work is outsourced. We are likely to employ additional help for the tedious task of image labelling.

LabelImg interface

Source: LabelImg GitHub

Our plan is to use the popular LabelImg program, built in Python, for image tagging. LabelImg creates XML files containing bounding boxes for each image. These files can be bundled into a .tfrecord file for use by TensorFlow in training.

If we implement a Mask R-CNN for instance segmentation of the tombstones, we will need to create PNG images for the masks of each image. We can repurpose our own tool or explore additional tools for this purpose.

Once our data is tagged and properly segmented into training, validation, and test sets, we can build our model. We will use and compare various models from TensorFlow Hub as a base for our model. Currently, we are looking into getting access to UCF campus labs for the actual training process, but COVID-19 presents additional challenges to getting this access. If this is not feasible, we will use our own hardware set up for training.

After training, if all goes well, we will export our model for use by our front-end application. For our app, we will be using Python with the Qt library to allow Dr. Giroux to import new aerial photographs for analysis, view the analyzed data, and tweak it in the interface before exporting it.

Special Difficulties

Obviously, COVID-19 presents unique challenges to our team. Indeed, it presents a challenge to all the Senior Design teams across all disciplines in the 2019 and 2020 years (and possibly beyond, depending on how this global situation plays out over the next several months).

All communication has been virtual to respect social distancing. Our team has created a Discord server we have used for communication during our project and research. Communication with Dr. Giroux has been over email or in virtual Zoom meetings.

The lack of face-to-face interaction make some aspects of collaboration more difficult. It definitely presents us with a challenge unique to our class with comparison to past teams in this class. In-person meetings offer a better opportunity to ensure a mutual understanding the project and everyone's roles within it. There's a greater chance of miscommunication when things are digital, and it has the possibility to derail entire projects, as I've personally experience in other classes that have made the necessary transition to remote learning. Hopefully, our team can make use of consistent communication to prevent any disasters.

Additionally, the use of on-campus resources would be beneficial to our team. However, getting access to campus facilities is now a more complicated procedure than it otherwise would be. Many labs are closed or require special permissions to access even with the campus as whole planning for a reopening in the fall. If we have to use our own hardware for training purposes, it has the possibility of slowing us down a little, which may affect how much training we are able to do, how many different models we can test our, etc. It will slow down our process of debugging our model if it runs into problems or doesn't perform as we expect.

And unfortunately, it is likely we will have to continue with online-only communication into the Fall semester. Even though this is a two-semester class reaching into the Fall semester, and the University plans to begin reopening this August, it remains unlikely that our particular

class will be returning to in-person learning. The administration has made clear their intention to prioritize freshman classes for face-to-face instruction. Therefore, we will have to do the difficult and hands-on portion of our project virtually. That includes enlisting the help of others with tagging data over the internet.

However, I'm confident in the capability of our team to make things work in these extraordinary circumstances given our performance and communication so far. So far I think all team members have demonstrated initiative and good communication. Also, with the help of various online resources, I've personally learned a great deal about neural networks, machine learning, and the TensorFlow framework and have faith in our team's ability in the execution of this project.

Database

Database Design

Dr. Giroux only needs a local database that she can access on her computer, with no internet access required for use. Thus, we determined that a SQLite database would be sufficient in meeting this requirement. As per our discussions with our sponsor and the project pitch presentation, the information the database will need to store is as follows:

- Cemetery name
- Stone row

- Stone column
- Latitude/longitude of each corner of the polygon
- Centroid of the polygon

Because the headstones will not necessarily be in neat-and-tidy grids, stone row and column id will be in as close to row and column order as possible. Figure #16 represents this information in a database diagram.

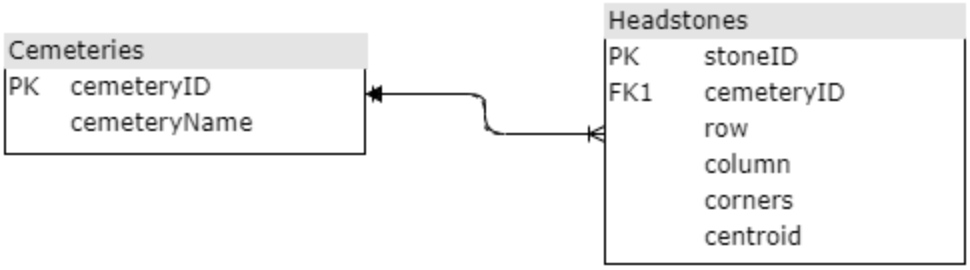


Figure #16: Database diagram 1 for the SQLite portion of the project. In this prototype, the corner coordinates are stored as a string to parse through on export.

In figure #16, the Headstones table has a single field for corner coordinates (entitled corners). This is due to the fact that we do not know how many corners a headstone polygon will have, as not all headstones are rectangular.

Also, an additional requirement of the database is that Dr. Giroux is able to export information by cemetery to a GeoJSON file. So, instead of having a maximum number of corners hard-coded as fields in the Headstones table, it may be easier to just store the corner coordinates as a string. Because of the necessary export to GeoJSON, we could store these

coordinates as they would be exported into a GeoJSON file (see figure #16 above for an example of this).

However with more familiarity with SQLite it may be easier to store corner coordinates in a separate Corners table. The maximum number of columns within SQLite is 2000, which is way more than we would ever need to store all the coordinates of a headstone polygon. Thus this option is the better way to proceed. See figure #17 for the database diagram of this concept.

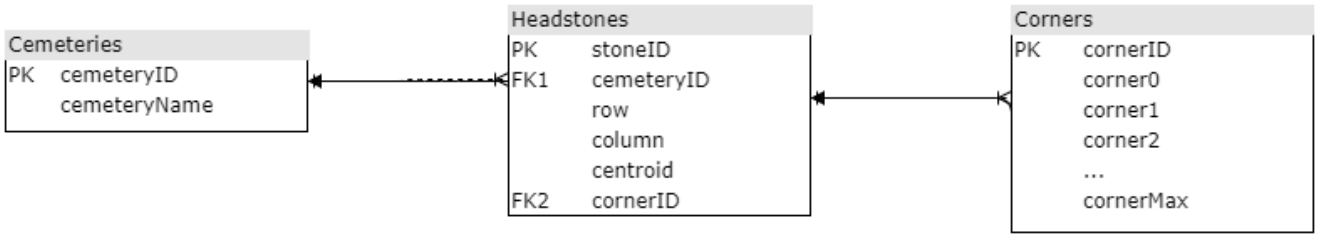


Figure #17: Database diagram 2, where coordinates are stored in a separate table. Here it may be easier for cornerID and stoneID to be the same value.

Figure #17 (database diagram 2) would be a cleaner way to store the corner coordinate information. It's less obtuse, and allows for easier editing later by a different group if necessary. Here I also considered adding object types from the GeoJSON format, to possibly make it easier to export cemeteries to GeoJSON files. However, all of the headstones are polygons, which we know, so it would be extraneous to store this information in the database. We could just add this programmatically when we choose to export.

Because cemeteries may have different naming conventions for headstone row and column, these values should be changeable by Dr. Giroux in the data entry screen. She should

also be able to change the corner coordinate values and thus the centroid of the polygon also. The rest of the values should remain static.

Python and the GeoJSON Format

There are libraries that incorporate GeoJSON features into Python. This library is known as the geojson library. There is an additional python library created called GeoPandas, based on multiple other libraries and used for geospatial data manipulation.

The GeoPandas python library is a powerful one, but it's toolset does not necessarily fit what we would want a library for. GeoPandas was created to manipulate already existing geospatial data, so it uses already created GeoJSON files, and edits the data from those. To my understanding, it does not create the GeoJSON files from scratch. It can take other geometric files (like a Shapefile or GeoPackage) but it does not create GeoJSON from scratch[32], which is what we are attempting to do.

The geojson Python library allows us to create GeoJSON objects within Python[33]. This is more in line with what we need to do for our project. The most important geometry type for this project is polygon, specifically polygons with no holes. Polygon coordinates consist of an array

---

[32] GeoPandas developers. "Reading and Writing Files" GeoPandas. https://geopandas.org/io.html (accessed July 27th, 2020).

[33] Gillies, S., Russell, M., Farwell, C., and Blake G., "Python-GeoJson", https://python-geojson.readthedocs.io/en/latest/# (accessed July 27th, 2020).

of linear ring coordinate arrays[34], but because we are working with aerial images of headstones and memorials, they are solid and will only be an array of a single linear ring coordinate array. A linear ring is just the boundary of a surface and is a closed LineString (another geometry type) with 4 or more coordinates.

```
{
 "type": "Feature",
 "geometry": {
     "type": "Polygon",
     "coordinates": [
         [
             [100.0, 0.0],
             [101.0, 0.0],
             [101.0, 1.0],
             [100.0, 1.0],
             [100.0, 0.0]
         ]
     ]
 },
```

Figure #18: Example polygon coordinate formatting within a GeoJSON file, from the GeoJSON documentation.

This just means that it's an array of all the points of the polygon, enclosed (meaning the first and last coordinate are the same). See figure #18 (example polygon) for an example of a solid polygon in a GeoJson file. Notice how the beginning and end coordinates are the same here, this is very important as the polygon will not be complete without this remaining true.

[34] Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., and T. Schaub, "The GeoJSON Format", RFC 7946, DOI 10.17487/RFC7946, August 2016, https://www.rfc-editor.org/info/rfc7946. (accessed July 08, 2020).

When there are multiple objects described within a GeoJSON file, it is necessary to include these in a collection. There are two kinds of collections in the GeoJSON format: FeatureCollection and GeometryCollection. Currently, GeometryCollections are somewhat obsolete, as they are used more for multiple different kinds of geometries that refer to the same object. What we are discussing here is a bunch of different headstones within a cemetery, so a FeatureCollection is much better suited for the task.

A FeatureCollection is a GeoJSON object that contains an array of Feature objects. A Feature consists of a single geometry and its properties. It is within the properties type that we can define things like the name of a polygon (a.k.a a headstone) and other attributes associated with these. Also within a Feature is the type and coordinate values. These describe what kind of geometry the feature is, and its location respectively. Also within Figure #18, you can see that the polygon geometry is described within a Feature. FeatureCollection is usually defined at the top of a GeoJSON file, with the rest of the file being the Features within the collection.

SQLite

Upon recommendation from our sponsor, we chose SQLite for our database development. Dr. Giroux stated that previous senior design groups had found success using it for their projects, and after researching it, its features and strengths seem to line up well with what our database needs to do. We only need a local database, just for Dr. Giroux to reference and edit on her own computer. Also according to the SQLite website, SQLite is backwards compatible

and will continue to be until at least 2050[35], thus our work will not become obsolete quickly like it would with using a more obscure and unmaintained library (like SpatiaLite).

An interesting fact we learned about SQLite during our research was that it is a recommended storage format for datasets according to the US Library of Congress[36], which according to preservationists at the Library of Congress means that SQLite has a maximum chance of being around and compatible for a long period of time.

SQLite databases are also easy to edit using Python, which is the language we are using for the machine learning aspect of the project. It's part of the Python Standard Library, so we will not have to download or install any extraneous libraries or functions to do this. This helps to prevent some of the conflating issues that come with using multiple languages or non-standard libraries on different aspects on a project.

Implementing the Database

To implement the SQLite database in python we will be using the sqlite3 package found in the python standard library. Using this, implementation of the database is very simple, especially considering we don't have to worry about any form of SQL injection attack as there

[35] "SQLite". https://www.sqlite.org/ (accessed July 21st, 2020).

[36] Sustainability of Digital Formats: Planning for Library of Congress Collections". Library of Congress. July 27, 2017. https://www.loc.gov/preservation/digital/formats/fdd/fdd000461.shtml#local (accessed July 27th, 2020).

are no foreign users accessing the database as our application is all local, removing the need for any form of prepared statements. Setting up access to a simple database is as simple as doing

```
import os
import sqlite3

DEFAULT_PATH = os.path.join(os.path.dirname(__file__), 'database.sqlite3')

con = sqlite3.connect(db_path)
cur = con.cursor() # instantiate a cursor obj
customers_sql = """
CREATE TABLE customers (
    id integer PRIMARY KEY,
    first_name text NOT NULL,
    last_name text NOT NULL)"""
cur.execute(customers_sql)
```

The application can be tightly coupled to the database as well as it is the primary form of export, leading to all database operations just being straightforward sql commands as we do not have to deal with any complex data structures in this project, mostly just writing coordinates and graves to a database file, with the ability to make edits in the future.

[37]

[37] Sqlite3 - DB-API 2.0 interface for SQLite databases¶. (n.d.). Retrieved June 25, 2020, from https://docs.python.org/3/library/sqlite3.html

McQuistan, A. (n.d.). A SQLite Tutorial with Python. Retrieved July 20, 2020, from https://stackabuse.com/a-sqlite-tutorial-with-python/

Frontend

GUI Development

During our very first meeting with Dr. Giroux, we brought up the user interface/data entry screen she requested and asked her about her expectations for this. Because of the large data set we are working with on the machine learning side, we need to dedicate a lot of time towards creating and annotating the training images for our machine learning algorithm. Thus, we decided the data entry screen to be more of a stretch goal, ideally completing the visual aspect along with the editor but at the very least having a screen that will allow Dr. Giroux to edit the coordinates textually. See figures #, #, and # below for the wireframes for these screens.

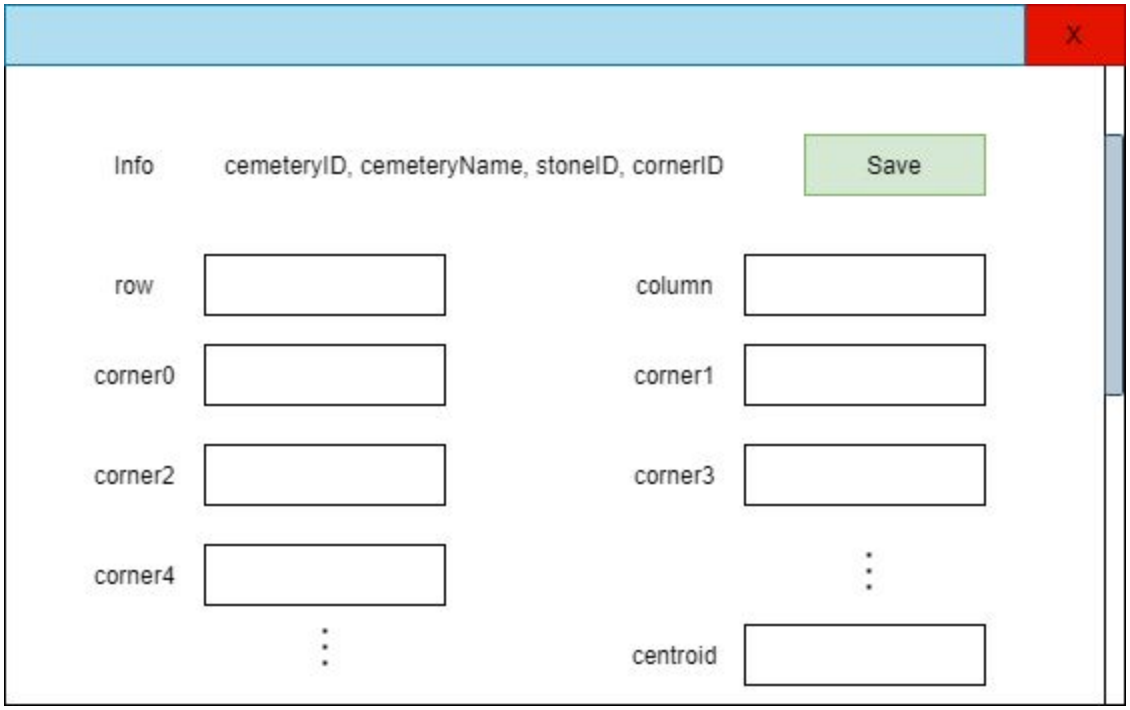


Figure #19: Sample GUI screen. This is the version we definitely want to have completed, with no headstone images. See figure #20 for the stretch goal version of the data entry screen.

Figure #19 (above) displays our goal for the data entry screen requirement. Because image annotation and training of the machine learning algorithm is a very large task involving manipulating hundreds of images, and the much more important task within this project, we are focusing more on that aspect. This data entry screen features an info square with cemeteryID, cemeteryName, stoneID, and cornerID. They are strictly informational and cannot be changed in the screen (as they are keys used in the database and must be unique).

The row and column variables can be renamed, in accordance to different cemetery practices. Each individual corner coordinate can also be changed within the text boxes next to the

label. The “Save” button would save the user’s changes and update the tables within the database to match those changes.

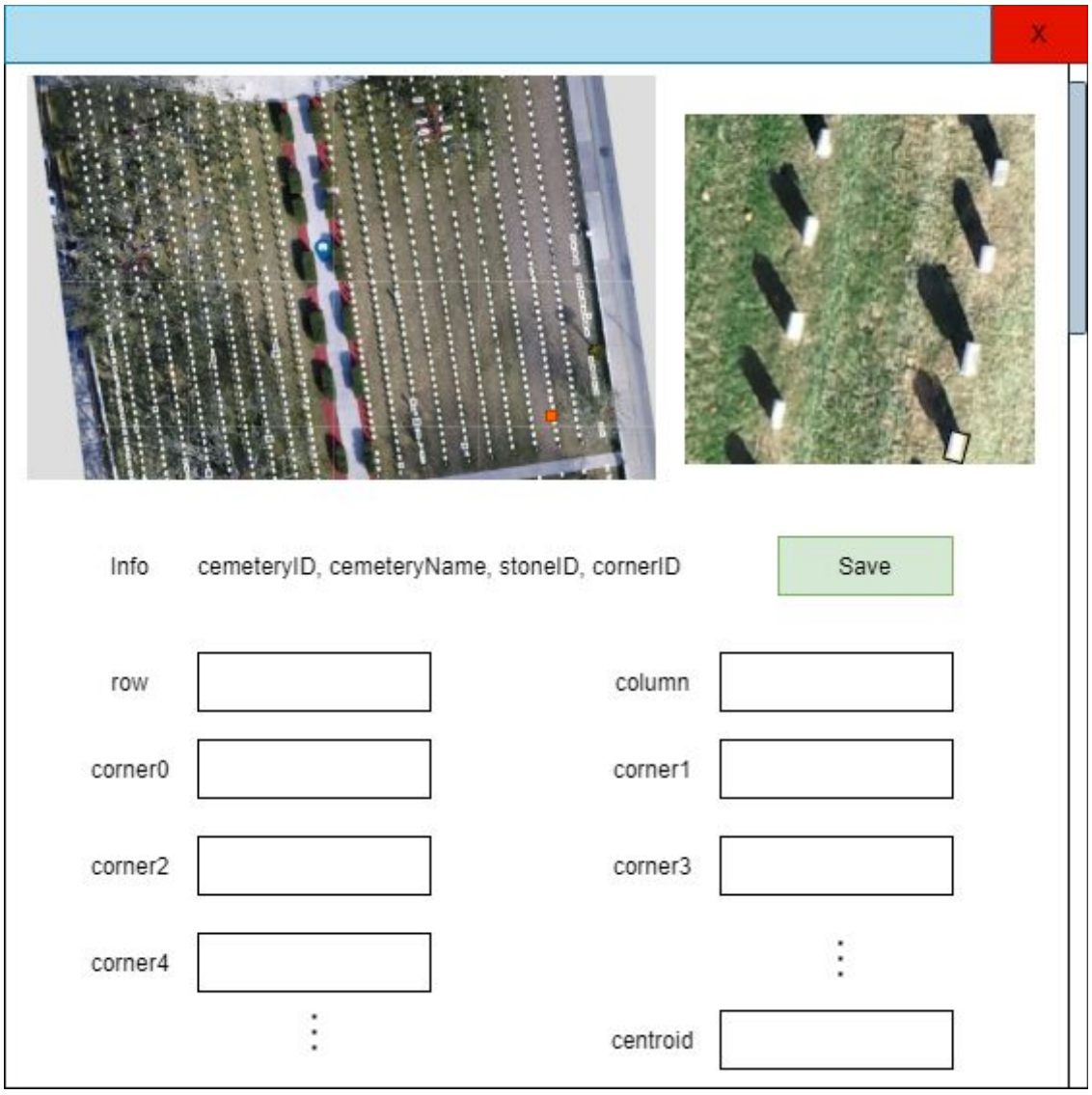


Figure #20: First Full GUI prototype for the data entry screen. This is the beginning of the stretch goal version, consisting of images highlighting the headstone being edited as well as text boxes to edit the coordinates and row/column names.

Figure #20 (above) features our first attempt at the stretch goal version of the data entry screen. It contains the same information as the previous prototype, but also includes the images of the headstone in question. Ideally, it would outline the headstone both in the image of the entire cemetery, and in a smaller, more zoomed-in image to check for accuracy.

Editing the coordinates would change the image to show those coordinates instead, without needing to hit the “Save” button. A user would also be able to edit the coordinates visually by moving the corners of the polygon, which would update the values in the text boxes below.

Because of the features listed above, we came up with a more suitable wireframe for the data entry screen, with more helpful features. See figure #21 below for this version.

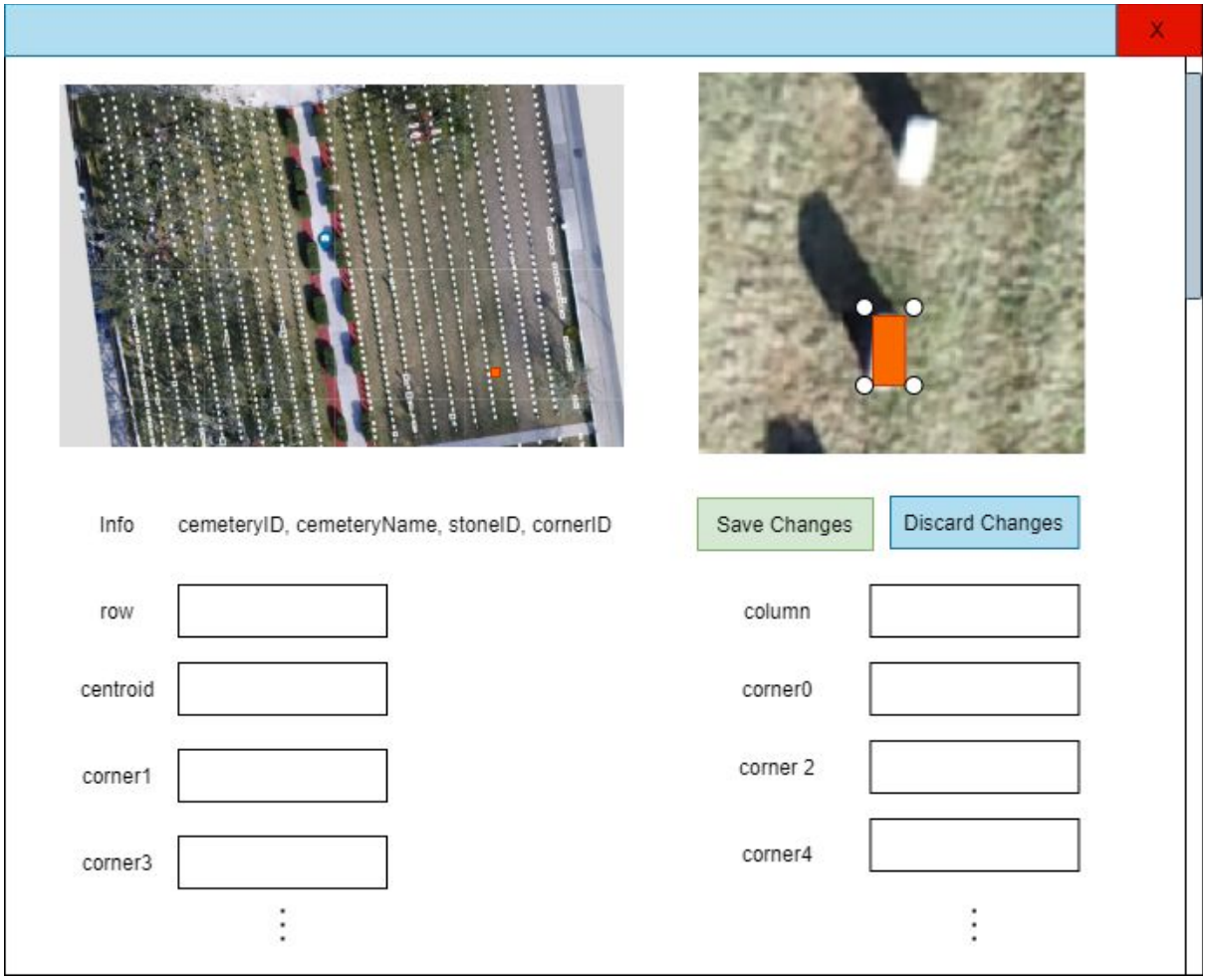


Figure #21: Final Full GUI prototype for the data entry screen. This is the stretch goal version, adding a discard button and being able to edit the polygons visually with markers.

The biggest changes between the final wireframe and the initial one were the discard changes button, and the ability to change the coordinates within the image of the headstone. In creating this version, it was noted that the zoom level for the image the user is "fixing" should be at least 24 (in GDAL2Tiles terms) from the original image. This gives a close enough view to see any disparity and edit it accordingly.

It may be prudent to include an export screen in order to select which cemetery to export to the GeoJSON file as well. A wireframe of this can be seen in figure #22.

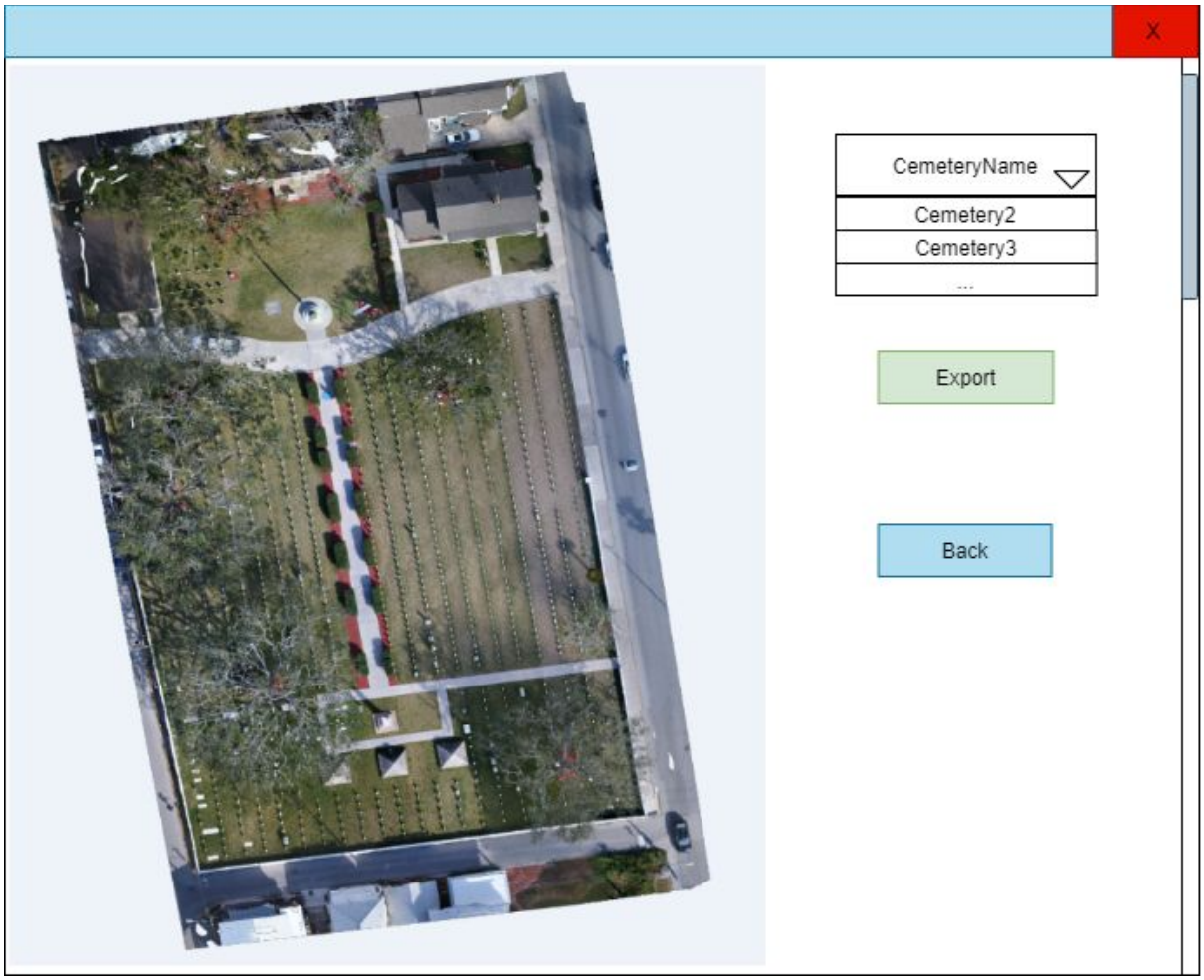


Figure #22: Wireframe for the export screen (stretch goal).

This adds the export functionality to the user interface, making it easier to select which cemetery and verify before export. Because adding this to the data entry screen was not a requirement, it is also a stretch goal. It displays the image of the cemetery and the name so that the user can verify that this is the cemetery that they would like to export. The user can also use

the drop down to select a different cemetery from the list, which gets its names from the database.

Implementing the UI

For the user interface of the application we've chosen QT as it is very popular and easy to use, and is well supported in general along with the python bindings for it. It is perfect for building functional minimalist applications that focus more on ease of use and functionality than they do on presentation, letting us get to right to what we need. The QT Python library we've chosen is PyQt5.[38]

To install PyQt5: > pip install PyQt5

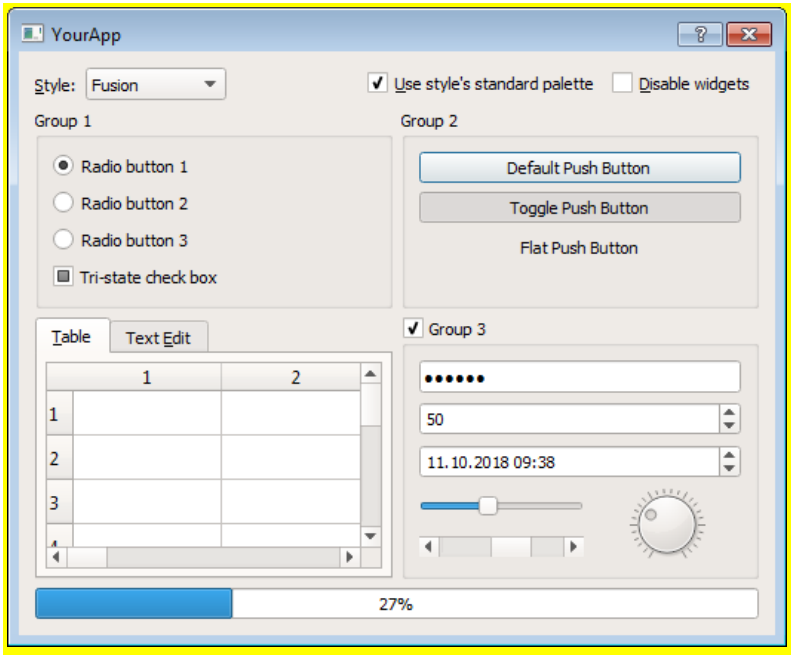


Figure #23: Example UI built with QT

[38] Machine Koder. (2019, June 12). PyQt vs Qt for Python (PySide2). Retrieved July 28, 2020, from https://machinekoder.com/pyqt-vs-qt-for-python-pyside2-pyside/

Getting started with QT is very simple. Fundamentally, everything is a widget, and any application window is just a collection of them arranged in whatever way is appropriate. Just about every common interactive element used in modern ui design has a widget cognate in QT, and associating them with actions is very simple too.[39]

The following code is sufficient enough to create a window with a small button that performs an action when you click it:

```
from PyQt5.QtWidgets import *
app = QApplication([])
button = QPushButton('Click')
def on_button_clicked():
    alert = QMessageBox()
    alert.setText('You clicked the button!')
    alert.exec_()

button.clicked.connect(on_button_clicked)
button.show()
app.exec_()
```

Example code from https://build-system.fman.io/pyqt5-tutorial[40]

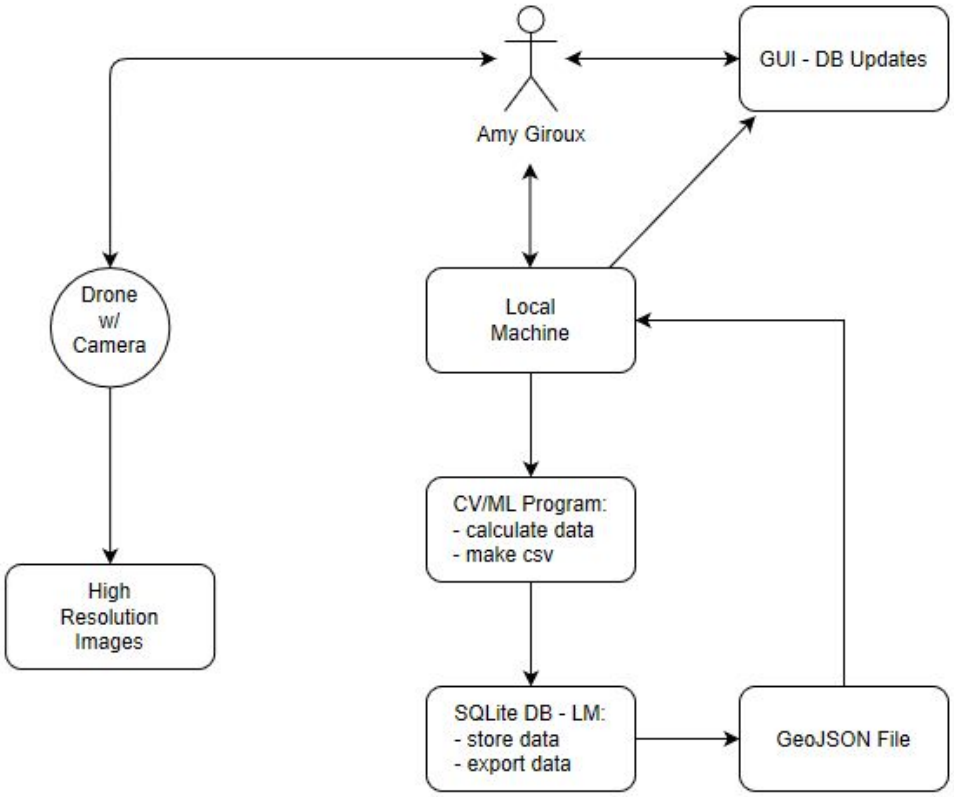
[39] PyQt5 tutorial 2020: Create a GUI with Python and Qt. (n.d.). Retrieved July 29, 2020, from https://build-system.fman.io/pyqt5-tutorial

[40] Pokhrel, S. (2020, March 11). Image Data Labelling and Annotation - Everything you need to know. Retrieved July 14, 2020, from https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1

Technical Specifications

Block Diagram

Legend:
LM - Local Machine
DB - Database
CV - Computer Vision
ML - Machine Learning
GUI - Graphic User Interface



Specifications and Requirements

Overview

This guide will show you how to install all the software and dependencies needed in order to run a Tensorflow machine learning model. A machine learning model, especially one with multiple layers and big data sets, requires heavy CPU/GPU usage. Most of the time, CPU alone does not have enough processing power in order to efficiently train a model. This is why in this guide we recommend Tensorflow-GPU. Dedicated graphics cards, such as NVIDIA, are compatible with Tensorflow and offer the processing power needed in order for a machine learning model to train and learn effectively and efficiently. Getting your machine set up for Tensorflow-GPU can be quite the headache and overly complicated, however, through the use of an Anaconda virtual environment, most of the headaches and complications can be avoided.

Getting started

In order to begin running a machine learning model on a computer, we must first have all the proper hardware, software, and dependencies in place. For this project in particular, we will be running the machine learning model on a Windows based computer, and everything from the Tensorflow machine learning model to the SQLite database will be run locally. Therefore, this guide will serve well for those hoping to get up and running on a Windows machine. While running this setup on a Mac or Linux machine will be very similar, the way in which some

programs and dependencies can or will be installed will vary. Hence, I recommend cross referencing this guide with official documentation from Anaconda, Tensorflow, and Python in order to adjust the following instructions for your preferred machine. Overall, if using an Anaconda virtual environment as this guide recommends, the process to run the machine learning model should be nearly identical, no matter which operating system you use.

Keeping Up to Date

This section of the documentation will guide you step-by-step in order to successfully install and run all software programs and dependencies needed in order to fully build and run a machine learning model in your own environment. However, it is very important to keep in mind that software updates occur very frequently, and some dependencies can become out of date and require upgrading.

Some library functionalities may become deprecated and require some change. Therefore, this guide will go into detail on what each major component of the program does, and what to do if you run into a situation where something has become deprecated or requires an update. It is also important to note that some dependency versions don't work well with others, and in the event of an update causing incompatibility, you can always downgrade or upgrade a particular dependency with ease (with the user of a virtual environment). This is why we recommend using a virtual environment.

Windows, Mac, and Linux

In order to begin the installation process, we must first install Anaconda onto our machines. In this guide, we will be installing the 64-bit Graphical installer for Python 3.7. In order to follow along with this guide and have a hassle-free installation process, we recommend the following specifications for your machine:

- Windows 10 - 64-bit (recommended)
- 8th Gen Intel i7 Processor (minimum recommended)
- 16GB RAM (minimum recommended)
- 5GB free disk space (allow extra space if storing datasets locally) (minimum)
- NVIDIA GPU with CUDA compute compatibility (minimum)

It is important to remember that this is only recommended, and the guide can be followed on any machine with similar or better specifications. However, the installation of Anaconda itself is crucial, as well as having a graphics card that can handle the computations of a machine learning model (more on this explained below). Anaconda offers installations for Windows, MacOS, and Linux. The 64-bit installer for Python 3.7 is recommended, no matter the machine.

Tensorflow-GPU

As briefly mentioned earlier, a machine learning model heavily depends on a computer's CPU/GPU processing power. This is because through Tensoflow's Keras API, there is a lot of complex mathematics and calculations happening in the background. The heavy mathematics and calculations can exhaust a computer's processing power very quickly and easily if not setup with the proper equipment, which is why we recommend the specifications above. Without a good GPU, a well defined machine learning model is as good as useless if it doesn't have the processing power to train and learn.

Hence, in this guide we will be working with Tensorflow-GPU which requires a dedicated graphics card with CUDA compute compatibilities. The good news is, if you are equipped with an NVIDIA GPU, chances are it is ready for Tensorflow-GPU.

For an extensive list of which GPUs are CUDA-enabled, checkout the link below:

https://developer.nvidia.com/cuda-gpus

Installation - Directions and Instructions

Anaconda

Anaconda is one of the most popular Python distribution platforms around the world, and is an open-source platform. Anaconda offers various tools and utilities ranging from an IDE to its

very own command prompt. Anaconda also makes it easy to manage multiple data environments that can be maintained and run separately without interference from each other.

Anaconda is built to support machine learning and data science. The Individual Edition they offer, and the eidition we will be using, is built with plenty of developer support and contains thousands of data science and machine learning packages that will make working with machine learning models very simple . We won't be using ALL of its features, but you may feel free to explore all it has to offer. One of the features that will come in handy is Anaconda's Jupyter Notebook.

Jupyter Notebook

Jupyter Notebook is a neat open-source tool that allows the creating and sharing of documents that contain live code, equations, visualizations, and narrative text. Although it isn't necessary to use this tool for the purposes of this project, it is very useful in the fact that since Windows WSL does not support visualizations, we can use this tool to bypass that issue.

Jupyter Notebook also allows us to run segments of code in a virtual-environment-like setting. That is, for every new "notebook" we can import and install certain dependencies for that particular instance. We can run segments of code as if they were running in a terminal, and it also displays any graphic interfaces or visualizations that may arise.

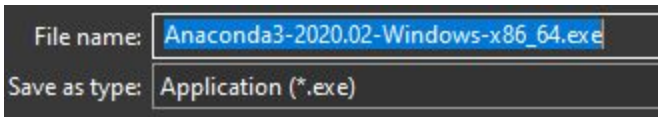
One useful way of using the Jupyter Notebook is to extract blocks of code from any Python file and post it into a Jupyter Notebook cell in order to test the functionality of that code without messing with the original content. The extracted block of code can then be changed, altered, or experimented with, using all the same dependencies as the original program.

The Anaconda download and installation file may be found at the very bottom of:

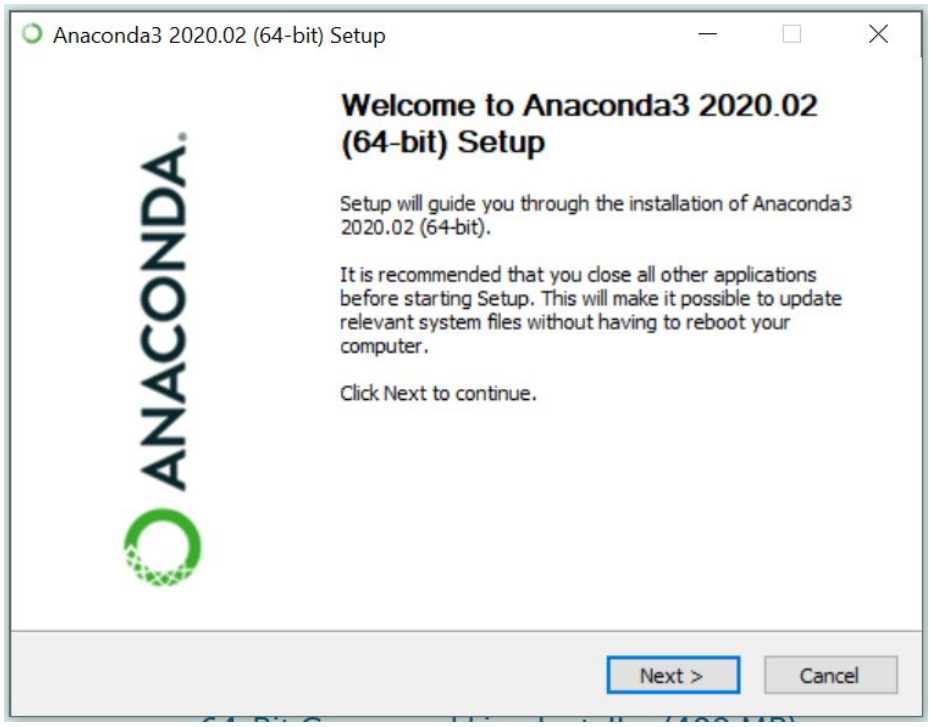
https://www.anaconda.com/products/individual

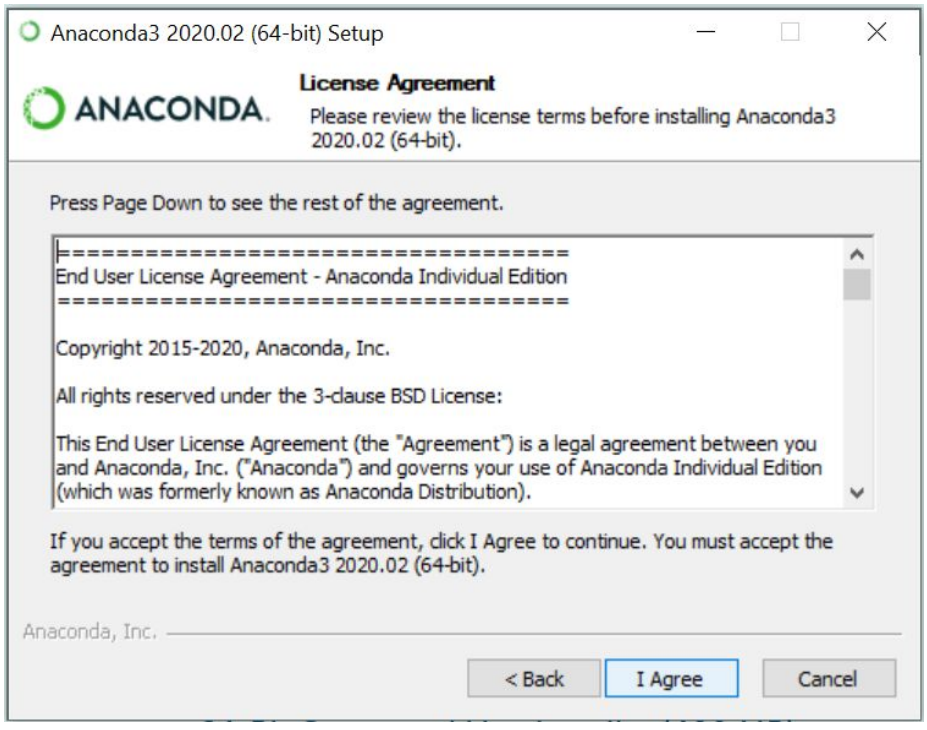
- First, we must find the appropriate install for our machines, in our case, Windows:

- Once the download is finished, if a prompt did not automatically pop up to continue installation, run the .exe file, which can be found in the file path you chose when you downloaded the installer.

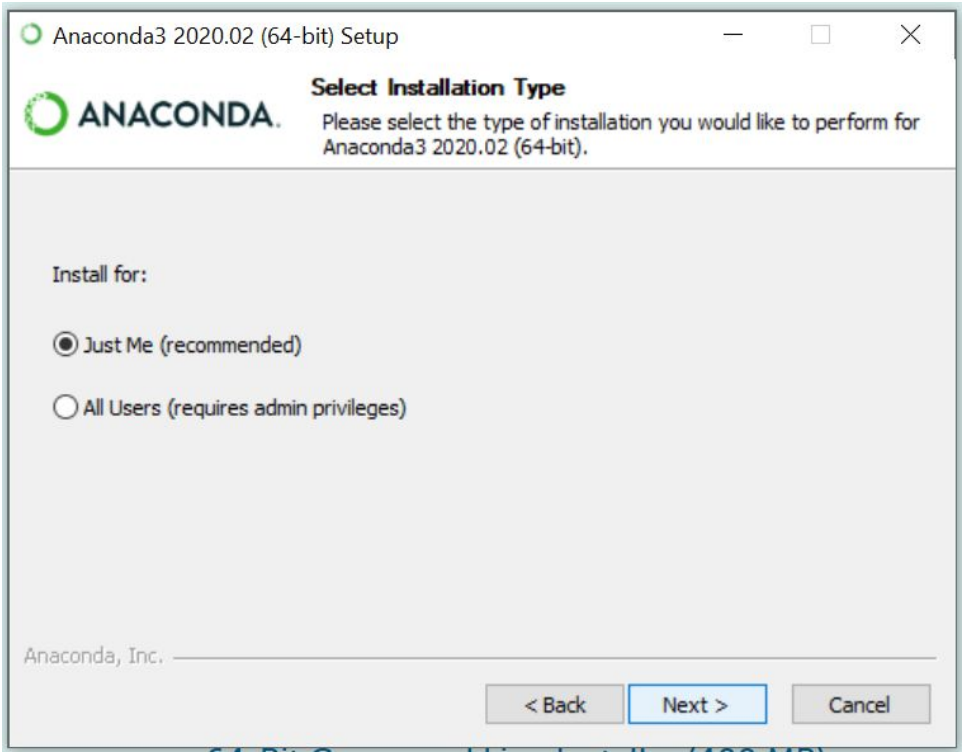


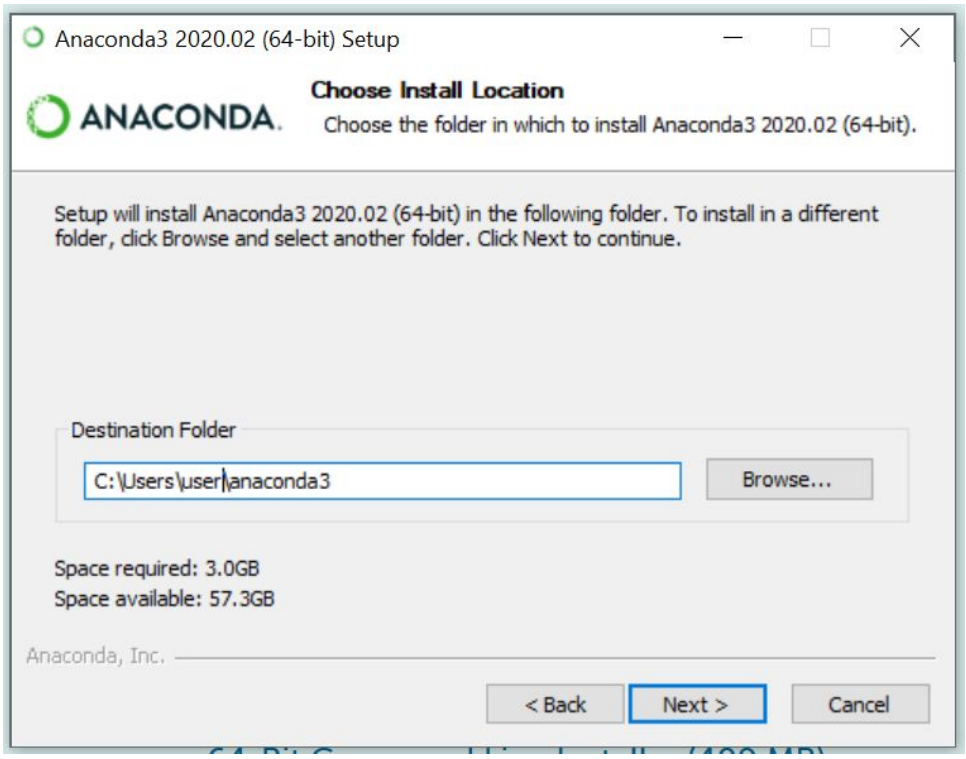
- Follow the prompts to continue installation:


Anaconda3 2020.02 (64-bit) Setup
Welcome to Anaconda3 2020.02 (64-bit) Setup
Setup will guide you through the installation of Anaconda3 2020.02 (64-bit).
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
ANACONDA.
Next >
Cancel

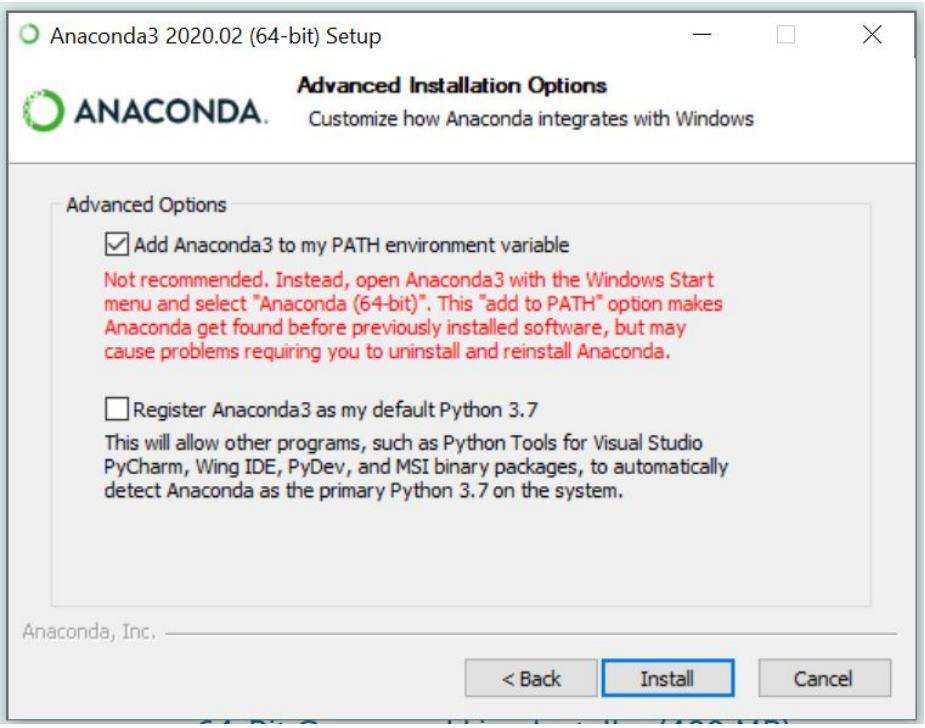

Anaconda3 2020.02 (64-bit) Setup
ANACONDA.
License Agreement
Please review the license terms before installing Anaconda3 2020.02 (64-bit).
Press Page Down to see the rest of the agreement.
End User License Agreement - Anaconda Individual Edition
Copyright 2015-2020, Anaconda, Inc.
All rights reserved under the 3-clause BSD License:
This End User License Agreement (the "Agreement") is a legal agreement between you and Anaconda, Inc. ("Anaconda") and governs your use of Anaconda Individual Edition (which was formerly known as Anaconda Distribution).
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2020.02 (64-bit).
Anaconda, Inc.
< Back
I Agree
Cancel

continued....

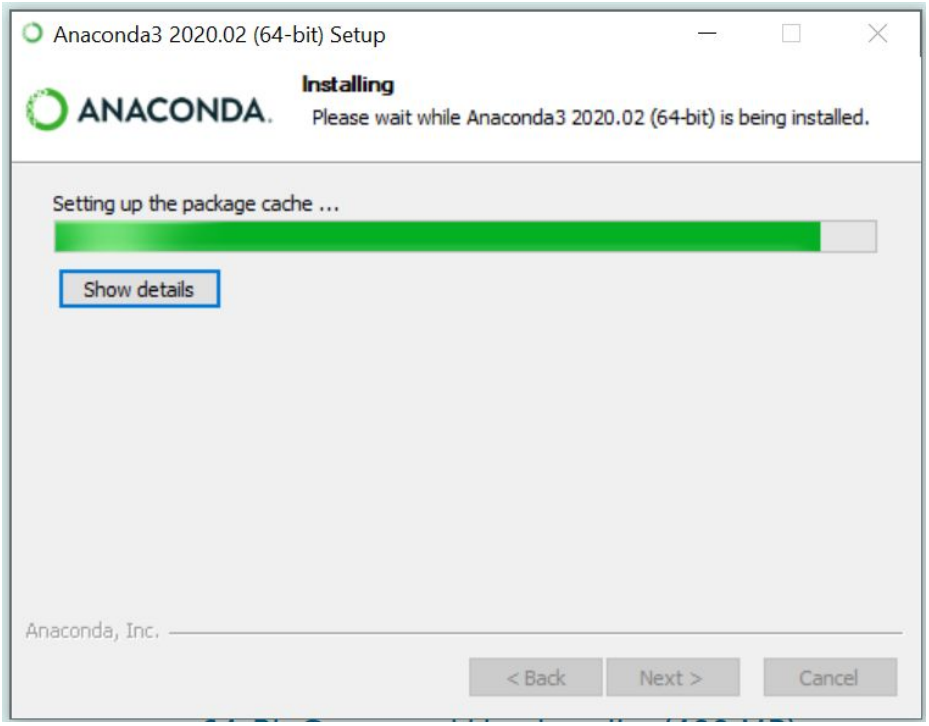

Anaconda3 2020.02 (64-bit) Setup
ANACONDA.
Select Installation Type
Please select the type of installation you would like to perform for Anaconda3 2020.02 (64-bit).
Install for:
Just Me (recommended)
All Users (requires admin privileges)
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2020.02 (64-bit) Setup
ANACONDA.
Choose Install Location
Choose the folder in which to install Anaconda3 2020.02 (64-bit).
Setup will install Anaconda3 2020.02 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Users\user\anaconda3
Browse...
Space required: 3.0GB
Space available: 57.3GB
Anaconda, Inc.
< Back
Next >
Cancel

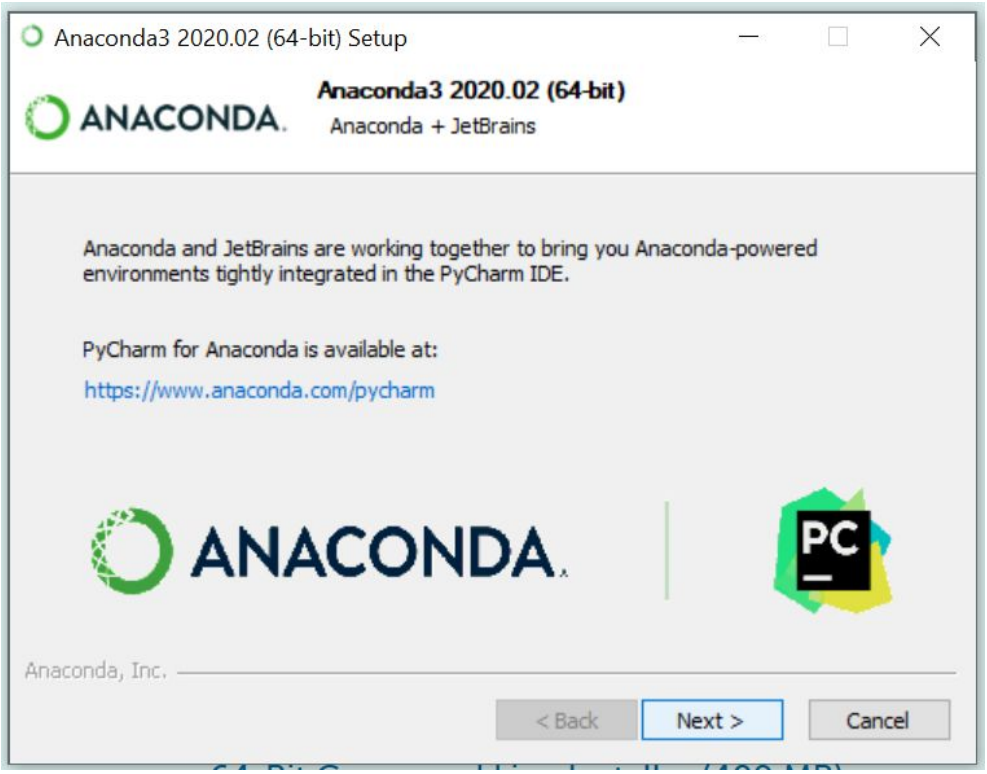
- Important! when prompted, select the first option (to avoid any issues):
- Note: the second option is completely optional.



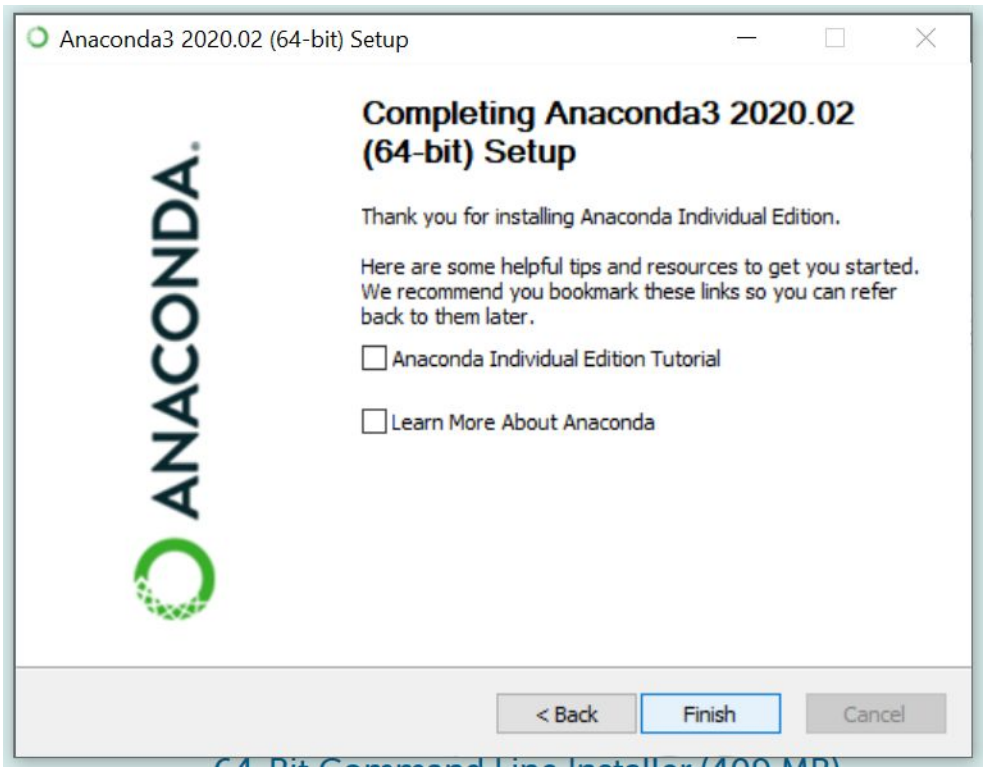
- It may take a few minutes for Anaconda to finish configuration and installation…


Anaconda3 2020.02 (64-bit) Setup
ANACONDA.
Installing
Please wait while Anaconda3 2020.02 (64-bit) is being installed.
Setting up the package cache ...
Show details
Anaconda, Inc.
< Back
Next >
Cancel
u

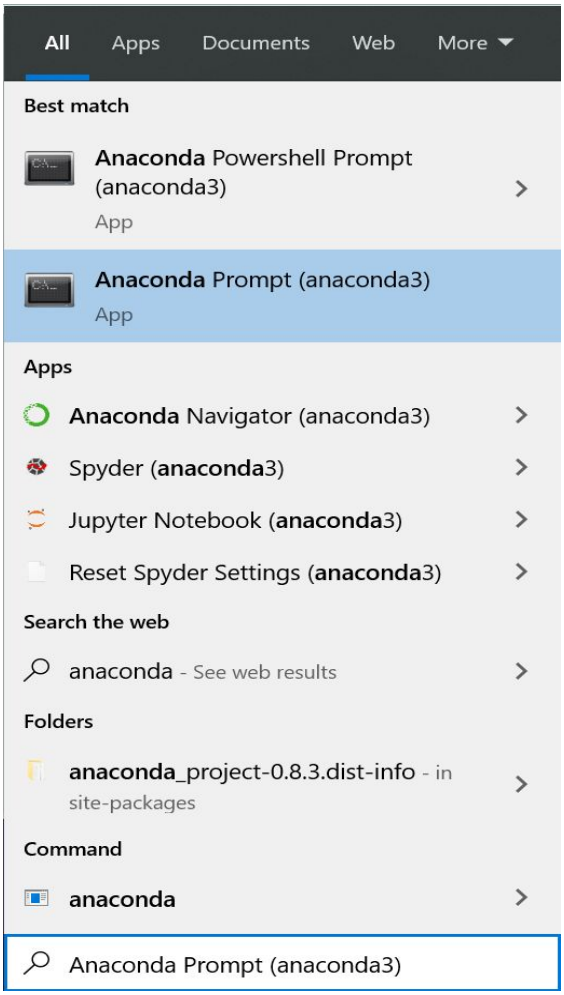
continued...



- The next options are optional:



- Once the installation is successful, we should have the following (Anaconda Prompt, etc.) when typing “Anaconda” in the Windows search bar:



Congratulations! You have completed your first steps, and are well on your way to building and training a machine learning model.

Anaconda Virtual Environment

Now that we have Anaconda installed on our machines, we will be mainly utilizing Anaconda Prompt. Anaconda Prompt will be used to create the virtual environment, install the specific Python version, along with Tensorflow, Keras, and all other dependencies. As mentioned earlier, keeping all of our frameworks, libraries, and dependencies isolated into a virtual environment will make working with Tensorflow a lot easier. We can easily downgrade or upgrade certain dependencies when one becomes deprecated, or when they have updates made available.

Disclaimer: At the time this document was written, July 9th of 2020, these instructions are up to date and all dependencies installed work seamlessly together. However, in the case that one dependency needs downgrading our upgrading in the future, this can easily be fixed with the following commands (variation of these commands may have to be used on case-by-case basis):

- To install the latest version (using numpy as an example):

  > pip install numpy

- To downgrade to a previous version (specify the version, using numpy as an example):

  > pip install numpy==1.15.4

To create and set-up a virtual environment on your machine:

1. Open the Anaconda Prompt.

Python3

2. Once the prompt is open, we're gonna create a new environment, running on Python 3.7:

   Note: "tfgpu" is the name I chose for my environment, your environment name can be whatever you would like it to be. You will then be asked if you want to install new packages, say yes.

   > conda create -n tfgpu python=3.7

3. Now we want to activate the environment, this will allow us to install all frameworks and dependencies to this environment only:

   > conda activate tfgpu

Ipykernel

4. Now we will install the tool needed to install a kernel to our environment:

   > pip install ipykernel

5. Proceed to install the kernel (I suggest you keep the display name the same as the env).

   > python -m ipykernel install --user --name tfgpu --display-name "tfgpu"

Tensorflow

6. Once the kernel is installed, it is time to install Tensorflow-GPU. It is important to install the GPU version of tensorflow. Be warned, this part of the installation may take a while, as many packages will be installed:

   Note: when prompted to proceed, say yes.

   > conda install tensorflow-gpu

Keras

7. Now that Tensorflow-GPU is installed, we will install the Keras API.

   Note: it is important to use pip instead of conda to install keras, because conda can possibly downgrade your Tensorflow GPU.

   > pip install keras

Jupyter Notebook

8. It is recommended to install Jupyter Notebook so that you can run segments of code without the use of an IDE or terminal and have display capabilities.

   > conda install jupyter

Other Dependencies Through Virtual Environment

NumPy

To install numpy: > pip install numpy

NumPy offers a way to provide powerful N-dimensional array objects. NumPy can also be used to reshape arrays into the needed shape required for machine learning models. For example, if we have a one-dimensional data set, but the machine learning model requires a 3-dimensional data set, NumPy can transform the data set into a 3-dimensional object very easily. NumPy also provides useful linear algebra, Fourier transform, and random number capabilities that can be used on machine learning model data sets. Overall, NumPy can be used as an efficient multi-dimensional container of generic data, where arbitrary data types can be defined.

Pandas

To install pandas: > pip install pandas

Pandas is a library for data manipulation and analysis that is built over NumPy. Pandas offers data structures and operations for manipulating numerical tables and time series. It provides fast, flexible, and expressive data structures designed to make working with structured and time series data both easy and intuitive.

Most importantly, Pandas offers a Data Frame data structure, a very flexible data structure where it can handle missing data, columns can be inserted and deleted, provides automatic data alignment, can merge and join data sets, reshape datasets, and much more. The Pandas data frame can also be used with SQLite, which will be used to store our data from processed images, to export data from the database table and into another file type. In our case, it can help export from SQLite database into a GeoJSON file.

Matplotlib

To install matplotlib: > pip install matplotlib

Matplotlib is a very comprehensive library that offers static, animated, and interactive visualizations in Python. Matplotlib is also very versatile, it produces publication-quality figures in various hardcopy formats and interactive environments across platforms. It is especially useful when working with machine learning models because it allows us to plot our data sets with labels, predictions, accuracy, loss, etc.

This allows us to visualize our model and understand how we can fine tune it for training. Machine learning models use advanced mathematical formulas to train and the training can be tracked by plotting our training data sets against our validation data sets, letting us examine things such as linear regression, etc. Matplotlib can also be used to plot our data with labels, letting us have a quick glance at our data sets, as well as augmented data.

Sklearn

To install sklearn: > pip install sklearn

Sklearn provides Scikit-learn functionalities that are sometimes used in conjunction with Tensorflow. It provides a range of supervised and unsupervised learning algorithms in an easy-to-use interface for Tensorflow. For example, Sklearn can provide functions for fitting and making predictions on models, provides various other mathematical functions which includes logistic and linear regression, and even a confusion matrix.

The confusion matrix can be used to describe the performance of a classification model. Although Sklearn can stand alone as a machine learning library, Tensorflow is more low-level and provides a way for developers to be more hands on and have more flexibility. Hence, we use parts of Sklearn with Tensorflow.

Itertools

To install itertools: > pip install itertools

Itertools is a library that implements a number of iterator building blocks from other languages. This makes it possible for Python to operate on iterators to produce more complex

iterators. This can be very useful for working with data sets, as we will have plenty of data to iterate through, and even generate from. The library is specifically made to be fast and efficient, so it is the preferred method for many developers that require using iterators and generators.

Pillow

To install pillow: > pip install pillow

Pillow is a Python Imaging Library, also known as PIL. It adds the support for opening, manipulating, and saving various different image file formats. Being that our machine learning model will be using image classification, this library will be very useful in our image processing. We can use this library to open image files, display them, resize the images, and even alter the images in many different ways, similar to a standard image editor program.

How To

Run Program

In order to run the machine learning python program, we will be using the Anaconda command prompt. We know that the virtual environment we set up is running on Python 3, and usually when running a Python program written in Python 3 through WSL or terminal, we run the program by typing in the command:

> python3 myprogram.py

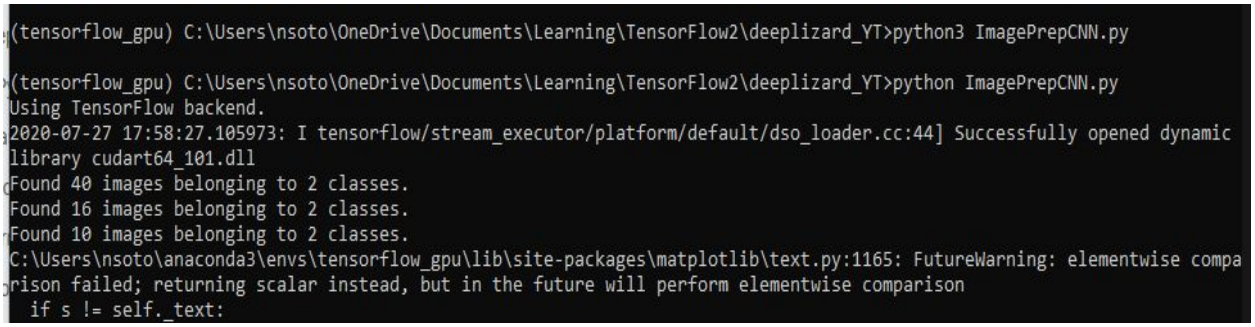
However, when running a Python program through the Anaconda command prompt, we can run the Python program as such:

> python myprogram.py

As you can see in the example below, running the program with the “python3” command will not yield any output, but if ran with “python” as mentioned, it will compile and execute the program. The first few lines, i.e. “2020-07-27 18:05:06.417774: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] ...” (many of the same type may pop up) can be ignored, as it is essentially just the prompt telling you that your GPU is being used in order to run the program.

The following lines after that, i.e. “Found 40 images belonging to 2 classes...” is output from the actual program. Finally, you may be greeted by a message of the type “FutureWarning” telling you that in the near future a functionality from a library or dependency may become out

of date. If this is the case, refer to the section titled “Keeping up to date” under “Specifications and Requirements” in the document.



Understand/Update Code

In this section you will learn how to generalize the source code into their own individual segments according to loading in data required for training the machine learning model, building a machine learning model, training a machine learning model, evaluating the model, making predictions, as well as much more. This will help you identify what parts of the source code does what, and how it can be changed to accommodate your needs.

The examples that will be shown and explained below are general examples and may not match the actual source code for this project. However, the idea here is that from understanding the basic and general structure of a machine learning model and understanding how the Keras API works, you will be able to identify and understand each segment of code in this project's source code. As a result, you will gain a fundamental understanding of how a machine learning model works, how to maintain it, and how to change it should you need to.

One major component in building and training a machine learning model is supplying it data to train from. Supplying the machine learning model adequate and quality data is essential

in the efficiency and accuracy of the models learning. In order to succeed, we have split this portion of the workload from the actual building and training of the model. In this section, as far as data, we will simply explain how data is loaded into the program and how it can be used. In order to understand how the data was processed for this project, please refer to section "…".

Load In The Data

Before you continue, please be sure to read and understand how we are using our data and how it is being processed, as mentioned above. CNNs expect data in the shape of N x H x W x C. That is, N x Height x Width x Color. Not all data will come in this shape, so we use functions from libraries such as numpy or pandas to reshape it into the expected form. For example, a 1D array can be an array of length n, and is equivalent to a 2D array of length n x 1, etc. This means, a 1D array can be reshaped into a 2D array and vice versa, as needed. We can also use functions such as ImageDataGenerator() to generate the data from images for us.

In the example below, we have multiple sets of images stored in three directories, we then use ImageDataGenerator() to split the images into three different batches of images for training, validation, and testing:

```
17  # varaibles that contain path to training, validation, and testing data.
18  train_path = 'cats-and-dogs/train'
19  valid_path = 'cats-and-dogs/valid'
20  test_path = 'cats-and-dogs/test'
21
22  # create batches for data sets, generate batches of tensor image data (format for keras).
23  train_batches = ImageDataGenerator().flow_from_directory(
24      train_path, target_size = (224, 224), classes = ['dog', 'cat'], batch_size = 10)
25  valid_batches = ImageDataGenerator().flow_from_directory(
26      valid_path, target_size = (224, 224), classes = ['dog', 'cat'], batch_size = 4)
27  test_batches = ImageDataGenerator().flow_from_directory(
28      test_path, target_size = (224, 224), classes = ['dog', 'cat'], batch_size = 10)
```

As you can thee, the directories of the three data sets are stored in variables for our convenience. The flow_from_directory() function will grab all the images from the specified directory for us and feed them into ImageDataGenerator(). Using ImageDataGenerator() we can control how many batches we want to split the images into for training, as well as augment the data (more on data augmentation later). To view all parameters that ImageDataGenerator() can take, and understand their functionalities, please refer to the Tensorflow official documentation below:

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Build The Model

Many machine learning models are not the same. Some can be very simple with very little layers, and many can be overly complex with multitudes of layers. Each layer can be of a different type, or there can be a pattern of layers through the multitude of layers. Deciding how many layers to add, or what kind of layers to add all depends on the use of the machine learning model. This is where fine-tuning of the model takes place.

Oftentimes, if we are training the model on new data, it would require removing the output layer of the CNN because that output layer is accustomed and trained to the old data.

Similarly, if the new data is similar, sometimes not many layers will need changing. Below is an example of how we can structure the layers of a model, as well as getting a taste for what kind of layers we can add.

Note: The parameters that are taken into these functions play a very important role and shouldnt be taken lightly. Every parameter in each layer plays a crucial part in the machine learning model. One of the most important features is the activation function. The activation function of a node defines the output of that node given an input or set of inputs. Each layer in the neural network is connected to another, and ultimately, they all connect in the very last layer. One change in output can create a chain reaction of changes to the final output layer. There are many different types of activation function available, and each has its own advantages or disadvantages based on the particular problem in question.

There are binary step functions which are threshold based, linear activation functions, and non-linear activation functions. Non-linear activation functions are the most popular because they allow the machine learning model to create complex mappings between the network's inputs and outputs. Some of the most useful non-linear activation functions are the Sigmoid/Logistic, TanH/Hyperbolic Tangent, ReLU (Rectified Linear Unit), Leaky ReLU, Parametric ReLU, Softmax, and Swish. Some activation functions are used more often in particular layers. For example, the Softmax function is often used in the output layers only.

Other functions such as the Sigmoid/Logistic function are slower and computationally expensive. Overall, choosing the right activation functions for our machine learning model will

require heavy theoretical research and lots of trial and error. Many of these functions can be stacked together, mixed and matched.

```
# Build the model using the functional API
i = Input(shape=x_train[0].shape)
# x = Conv2D(32, (3, 3), strides=2, activation='relu')(i)
# x = Conv2D(64, (3, 3), strides=2, activation='relu')(x)
# x = Conv2D(128, (3, 3), strides=2, activation='relu')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
# x = Dropout(0.2)(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
# x = Dropout(0.2)(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
# x = Dropout(0.2)(x)

# x = GlobalMaxPooling2D()(x)
x = Flatten()(x)
x = Dropout(0.2)(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# Compile
# Note: make sure you are using the GPU for this!
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Fit
r = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=50)
```

To get a full list of the types of layers we can use and what parameters they can take, please refer to the Tensorflow official documentation below:

https://www.tensorflow.org/api_docs/python/tf/keras/layers/

Train The Model

Training the model is more simplistic than creating the model. Once we have our model created and compiled, we run the fit() or fit_generator() function on the model in order to begin the training. In the example shown below, we pass the fit_generator() function a couple of parameters with the following significance: train_batches contains the batches of data that we collected using ImageDataGenerator() from the previous section. The steps_per_epoch is calculated by taking the number of batches we have and dividing it by the total number of images we are working with. Although it cant be seen in the image, we have 40 images contained in the train_batches and each batch is of 10 images.

Hence, our steps_per_epoch is 4. We can also pass this function a set of validation data in order to train the model so it understands what it is looking for. Epochs is how many steps the model will take in order to process all of the training data. Ideally, the more epochs the model takes to train, the more accurate it may become.

```
29  model.fit_generator(
30      train_batches,
31      steps_per_epoch = 4,
32      validation_data = valid_batches,
33      validation_steps = 4,
34      epochs = 5,
35      verbose = 2)
```

When we run the program, we can see the epochs being calculated from the Anaconda command prompt. Of course, the more epochs there are, the longer the model will take to train, the longer it will take for the program to finish running. Below is an example how the program looks when it is executing the epochs (in this case, 5 epochs, from a very basic model):

```
 - 3s - loss: 1228.5343 - accuracy: 0.4250 - val_loss: 1009.5422 - val_accuracy: 0.5000
Epoch 2/5
 - 0s - loss: 868.4305 - accuracy: 0.5000 - val_loss: 849.8314 - val_accuracy: 0.5000
Epoch 3/5
 - 0s - loss: 419.5427 - accuracy: 0.6250 - val_loss: 674.3995 - val_accuracy: 0.5000
Epoch 4/5
 - 0s - loss: 232.5603 - accuracy: 0.6750 - val_loss: 139.8742 - val_accuracy: 0.5625
Epoch 5/5
 - 0s - loss: 125.8120 - accuracy: 0.8250 - val_loss: 303.0119 - val_accuracy: 0.5000
```

As we can see, the program tells us how long each epoch took to calculate, as well as the loss and accuracy at each step, along with the validation loss and accuracy. We can also observe that the numbers seem pretty weak, that is because they are. In this example, We have used a very simple and basic machine learning model that consisted of very little layers.

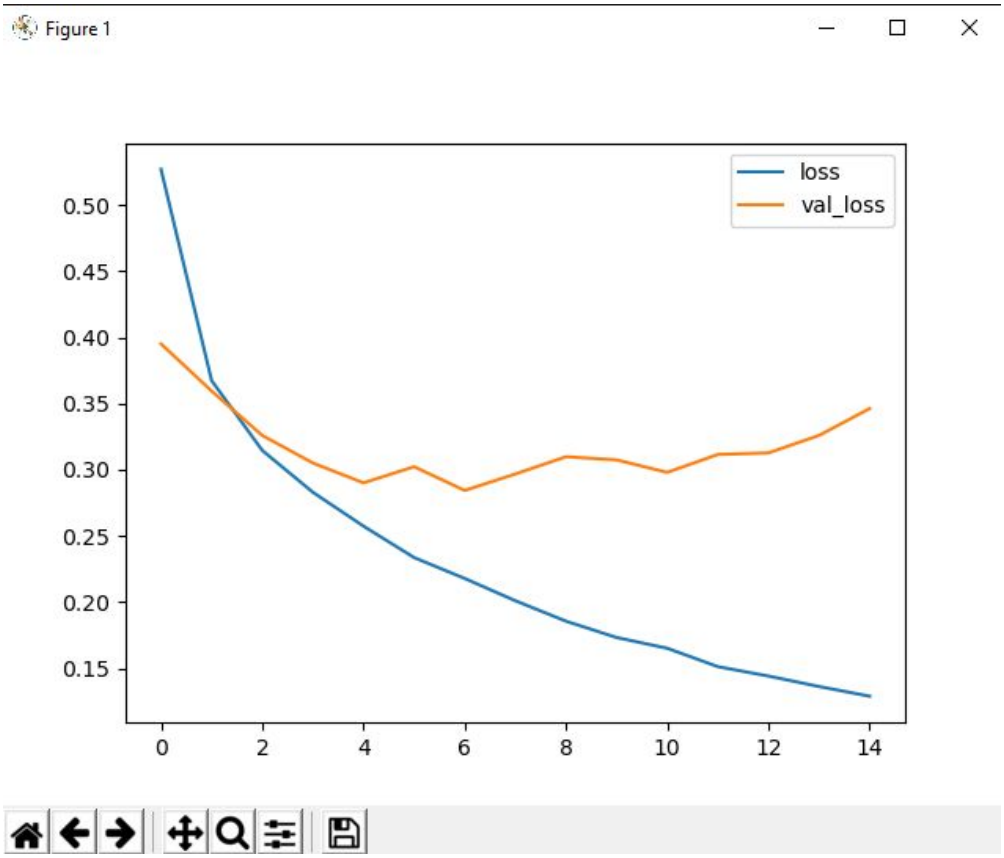
Evaluate The Model

Once we have built and trained the model, it is time to evaluate its performance in order to make some fine-tuning decisions. It is unlikely that you will achieve your desired results from the first build. Through trial and error, we can continuously adjust the layers until we achieve our desired results. In order to evaluate our model, we must be able to label our images and plot our results in a graph. This can be done through the use of matplotlib. Matplotlib offers various functions that let us label and plot our data into a visual graph so we can better understand how our data is being affected. Below is an example of how we can access the data from our training and plot it to be shown in an interface using matplotlib.pyplot:

```
44  # plot loss per iteration.
45  plt.plot(r.history['loss'], label = 'loss')
46  plt.plot(r.history['val_loss'], label = 'val_loss')
47  plt.legend()
48  plt.show()
49
50  # plot accuracy per iteration.
51  plt.plot(r.history['accuracy'], label = 'acc')
52  plt.plot(r.history['val_accuracy'], label = 'val_acc')
53  plt.legend()
54  plt.show()
```

Once the data is plotted, we can expect a graph such as the following:

*Picture is on the following page.

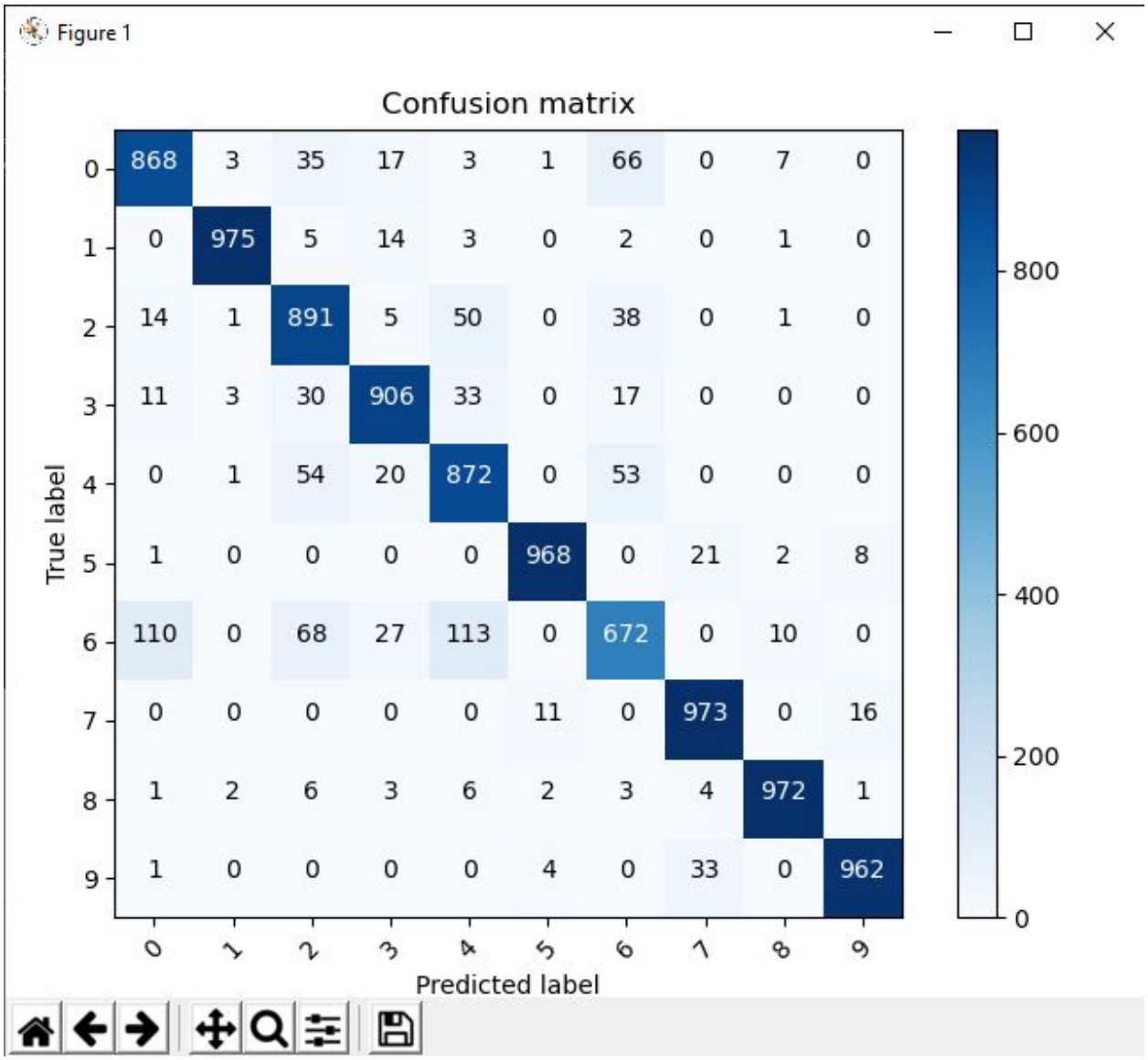


Make Predictions

After the model is built and trained, we can then use a testing data set in order to see how well our train model performs. As described earlier, we can split out data into three categories: training, validating, and testing data. We use the training and validation data sets during our training to get weights for our model. The weights are saved to the model so that it knows how to detect images that it was trained to detect, on datasets it has never seen before. This dataset is the testing dataset. The testing data set can be used to make predictions using the model and its saved weights.

Making predictions will allow us to see just how well the model will perform in a real world setting. When making predictions, it usually requires using a “confusion matrix” so we can identify images and their labels, and how they are being interpreted by the model. The confusion matrix can consist of a graph where it compares actual output with predicted output. This is done by supplying the model with images which we know what labels it belongs to and comparing it with labels that the model has given the images in the testing dataset. Below we can see just how the confusion matrix works.

Confusion Matrix



The confusion matrix is labeled with “True label” and “Predicted Label”. Here we can see how many times the machine learning model correctly predicted the label of an image when processing an image, compared to what the image was actually labeled as. For example, for a

label of "6", our model correctly predicted the label "6" for an image 672 times when it was in fact 6. However, it predicted the label "6" for an image 53 times when it was in fact labelled "4". Overall, in this example, the model incorrectly labeled an image as "6" 179 times. The confusion matrix also uses a standard color coding that can help you visualize how accurate it was.

In order to plot a confusion matrix, we can use any open-source code we want from the internet. There are various different source codes available to choose from. The confusion matrix used in the previous example is as followed: (keep in mind, this is a standard confusion matrix and it can be altered).

*Image is on the following page.

```python
# Plot confusion matrix.
from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()


p_test = model.predict(x_test).argmax(axis=1)
cm = confusion_matrix(y_test, p_test)
plot_confusion_matrix(cm, list(range(10)))
```

Data Augmentation

Sometimes the amount of data we have for our model isn't enough, or we would like to just enhance our current dataset with altered images to truly test the accuracy of our model. In this case, Tensorflow offers us this ability through the keras API through the ImageDataGenerator()

function. The ImageDataGenerator() function allows us to pass various parameters to it that can enhance or alter an image in various distinct ways. This works by breaking the image down into binary data, and stored into an array, or tensor. The data is then manipulated to mimic the movements or alterations of the image. For example, an image can be horizontally or vertically flipped, shifted up/down/left/right, zoomed in or out, rotated, and much more. We can further simplify this task on multiple images by passing a directory path that contains the images we would like augmented into a generator. The generator will then go through all the images and augment them according to the parameters passed to the function. Below is an example of what augmenting the data looks like:

```
# augment image.
gen = ImageDataGenerator(
    rotation_range = 10,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.15,
    zoom_range = 0.1,
    channel_shift_range = 10.,
    horizontal_flip = True
)

# get image path.
image_path = 'cats-and-dogs/train/dog/15.jpg'

# obtain image from image path.
image = np.expand_dims(plt.imread(image_path), 0)
plt.imshow(image[0])
plt.show()

# generate batches of augmented images from 'image'.
# aug_iter = gen.flow(image)
# to save augmented images...
aug_iter = gen.flow(image, save_to_dir = 'cats-and-dogs/augmented-images/dogs')

# get 10 samples of augmented images.
aug_images = [next(aug_iter)[0].astype(np.uint8) for i in range(10)]
```

Save/Load Model/Weights

One of the most important features of a machine learning model is the weights it holds after training. A model can be saved and loaded in order to be reused or changed. Along with the model, we can also save and load the weights. We can even load different weights into different models and vice versa. The weights and biases of a machine learning model are the learnable parameters. As the input data goes through each layer in the neural network, the weights can change its output value, or pass it along to the next layer.

This is how a machine learning model "learns" and understands how to process or label data, based on the weights as it passes through layers. Overall, if we have a model built and trained, and would like to save the weights to use on an updated model, or maybe transfer the model to another machine that does not have access to the old weights or the dataset it trained on, we can simply save the weights. We can even save both the model and the weights and load them in whenever we please. Through Python, we can even save this information in JSON or yaml format. Below is an example of how we can achieve these things:

```
# save: the architecture, weights, training configuration, and state of optimizer, of model.
# in this instance, we are saving a model from CreateSequentialANN.py.
model.save('medical_trial_model.h5')

# create a new model using the saved model.
new_model = load_model('medical_trial_model.h5')
new_model.summary()
new_model.get_weights()
new_model.optimizer

# save: only the architecture.
# save as JSON or yaml with .to_yaml()
json_string = model.to_json()
print(json_string)
model_architecture = model_from_json(json_string)
model_architecture.summary()

# save: only the weights.
model.save_weights('my_model_weights.h5')

# to load weights into a new model.
model2 = Sequential([
    Dense(16, input_shape = (1, ), activation = 'relu'),
    Dense(32, activation = 'relu'),
    Dense(2, activation = 'softmax')
])

# Load the weights into another model.
model2.load_weights('my_model_weights.h5')
```