ArachnID:

An Android App for the Identification of Household Spiders

Nicholas Whitehead | nwhitehe@unca.edu

B.S. Computer Science, Computer Systems

Instructors & Advisors:

Adam Whitley, Ph.D. | awhitley@unca.edu

Marietta Cameron, Ph.D. | mcameron@unca.edu

November 10, 2021

Abstract

This project provides a tool for identifying spiders found within homes or residential buildings. The software produced includes an Android application with which the user can photograph a spider and receive immediate information about that spider's species, venom status, and habitat. The identification software is based on a Convolutional Neural Network trained on images of North Carolinian spiders, allowing for automatic species classification. This classification model is hosted on a web server, with a Flask application communicating between the user's device and the model. The final product is a tool useful for ensuring the safe handling of venomous species, as well as providing an educational opportunity for learning about a small part of North Carolina's wildlife.

Introduction

Spiders frequently seek shelter in the warmth and safety of human residences. While the majority of these spiders may be harmless, western North Carolina is home to dozens of species, including some that pose real danger. Identifying a spider's species can be challenging for anyone without prior training, and having an identification tool immediately at hand at all times eliminates the uncertainty. The goal for this project is to provide such a tool. *ArachnID*, pronounced "arachnid", is an application that allows the user to photograph a spider with their phone and receive immediate information about the spider's species, potential for harm, and habitat. Properly identifying this information allows the user to make a more informed judgment when handling a spider. It is important to recognize when a spider is venomous, so that it may be treated carefully and, if necessary, removed from areas where people frequent. The identification tool also provides value beyond safety precautions. This app is useful for educational purposes as well, giving the user an opportunity to learn more about local North Carolinian fauna.

There are only two considerable bounds for the target audience of this project. The first of which is the user's location; the app will only provide fully accurate identification for species local to the western North Carolina region, with decreased efficacy outside of the southeastern United States. The other limitation is the user's device: *ArachnID* is developed for Android only, meaning that those with iPhones or phones of any other operating system will not be able to use it.

The Android application is the main component of the project that the user will interact with. The image classification model runs on a separate machine, with a web server hosted on *PythonAnywhere*[1] handling the communication between the user's phone and the model. The app provides the user interface, allowing users to take a photo with their phone's camera and send it to the server via a Flask web application. The server runs the photo on the classifier, and sends a response to the user. The server's response includes both the species classification and information about that species, such as its venom status and habitat. The app then displays this information to the user. The classification model is a Convolutional Neural Network built with the TensorFlow framework, trained on images of North Carolina native spider species. These three parts work together to provide a complete spider identification tool.

Literature Review

Computer vision is a rapidly growing field, with many independently developed tools for identifying certain objects in the world. Some of these are incredibly vast in their approach. For example, the largest product in this category, and the primary existing competitor to this project, is Google Lens[2]. Google Lens is a product for general-application vision-based classification. Features include live classification of objects through the viewfinder, a wide range of classifiable categories, optical character recognition for text transcription, and built-in translation tools. The massive scope of this software is backed by Google's access to a near-infinite amount of training data. Despite the large scale of the product, however, there are certain categories that the software works better on than

others. The product boasts its highly accurate plant recognition capabilities, but is less accurate in other categories like food dishes or clothing. This reflects the opportunity for improvement in certain niches. Google Lens does not nullify the improvements this project will look to provide towards identifying North Carolinian common spider species.

Google's use of the abundant data it has access to as training data for recognition models has instigated a debate regarding copyright and fair use in machine learning. This question is relevant to this project because the spider classification model requires a similar collection of training images, which may be of varying copyright status. The *Author's Guild v. Google* District Court case set an important precedent regarding the legality of using copyrighted material in discriminative deep learning models[3]. The Google Book Search algorithm was trained on a large database of copyrighted material, which the Author's Guild alleged to be an infringement of copyright for millions of books. The Court ruled in Google's favor, accepting the reasoning that Google's Book Search poses no market substitution for the original works and therefore, the use of copyrighted materials was covered by fair use. This case established legal precedent ensuring the safe application of fair use for discriminative deep learning models.

For this project, a large amount of the training data will come from the website *Spider ID*[4]. This is a community-supported database of spider images, including a set specifically of North Carolinian spiders. Most species in the North Carolina list have dozens, if not hundreds, of user-submitted images. Some relevant information is also given for each species, including things like typical sighting locations. There is limited information on any potential danger to humans, and as such, this information would need to come from other sources. An interesting result of the user-submitted nature of the website is that the quantity of images for any given species loosely correlates with the frequency of appearance in real life, making it a helpful tool for selecting which species to prioritize training for.

Published in 2012, *Image Processing for Spider Classification* was a project led by Jaime R. Ticay-Rivas, et al.[5], that explored the idea of using image recognition software to monitor and preserve biodiversity in spider species. The paper outlines the process involved in creating a classification model based on spider web recognition. The authors describe the database of spider web images used to train the model, as well as the preprocessing techniques used to isolate the most important parts of the images. These techniques are useful as a model to study when designing the preprocessing stage of the final classification model for this project. The authors utilized a Support Vector Machine (SVM) for the image classification model, as opposed to the Convolutional Neural Network (CNN) planned for *ArachnID*. CNNs have proven more effective for most image classification models than SVMs, although they were less popular at the time of this paper's publication. Despite this difference in approach, the paper gives a useful description of cross validation techniques, namely K-fold cross validation and Hold-out validation.

Cross validation is a tool for measuring the accuracy of a classification model. An article by Sanjay M., "Why and How to Cross Validate a Model?",[6] gives an introductory overview of cross validation. A Hold-out approach involves separating the training data into two sets from the start; one is used to train the model, the other is used to test the accuracy of the model. This approach is only preferable for problems with massive sets of images, because it is very easy for the Hold-out technique to overfit to the training data if the data set is too small. In this case, the model would perform less effectively when facing real-world, unseen images because of the biases created by the small training set. K-fold cross validation splits the data randomly into *K* subsections, or folds, with *K*-1 folds being used to train the model and the *K*th fold used for testing. This is repeated until every randomly generated fold acts as that *K*th fold, the test fold. K-fold cross validation is better for smaller data sets, and likely more applicable for the spider classification model.

The article "Understanding 8 Types of Cross-Validation", by Satyam Kumar[7], gives another overview of a few different types of cross validation, including Leave-p-out, Leave-one-out,

Hold-out, Repeated Random Subsampling, K-fold, Stratified K-fold, Time Series, and Nested cross validation. The article points towards K-fold and Stratified K-fold as the most popular systems, both being generally applicable to image recognition problems. Stratified K-fold cross validation addresses the issues that can stem from imbalanced data, which can be troublesome when using simple K-fold validation. For example, If there are *x* images for one class, and 4*x* images for another, the second class may be overrepresented in the model and may introduce biases. Stratified K-fold ensures that one class isn't overrepresented in the validation model by selecting an equal number of instances from each class for the validation data. These two approaches are very similar, and the tradeoffs of each will need to be considered when the final training data set for this project is compiled.

A Review of Deep Learning in Image Recognition, written by Myeongsuk Pak and Sanghoon Kim[8], compares the performance and complexity of four different Convolutional Neural Network architectures: AlexNet, VGG, GoogLeNet, and ResNet. Respectively, they have error rates of 16.4%, 7.3%, 6.7%, and 3.6% in the ImageNet Large Scale Visual Recognition Challenge, the most comprehensive annual test of cutting-edge image recognition models. The takeaway is not, however, that these error rates are indicative of an objective tier-list. Each model has different complexity measures and implementation requirements. What performs best on a general data set is not necessarily what performs best on specialized niches such as spider identification.

Building the spider classification model patterned after any of these four CNNs would likely result in a fairly accurate model. The article "VGG16 – Convolutional Network for Classification and Detection", by Muneeb ul Hassan[9], goes into detail about the VGG architecture. This model is relatively large in its size and training times, but this is made up for by the simplicity of its implementation and use. VGG will be the primary target for this project's model architecture.

Android development is much simpler than CNN development. The Android development documentation[10] provides ample information about each design component. However, the network

code requires more involved consideration, as there are many different network libraries that provide similar services. The paper *Evaluation and Identification of Android Networking Libraries*, by Jaydevi Bhade and Himanshu Yadav[11], provides exactly this. The authors compare the libraries Retrofit, Volley, OkHTTP, Fast Android Networking Library (FANL), Jus, and Robo Spice. Retrofit, OkHTTP, and FANL are the three that provide the most relevant tools for this project. Image transfer is the main consideration, and each of these three provide similarly effective functionality for this. While Retrofit and FANL are both built on top of OkHTTP and provide more elaborate features, OkHTTP was chosen for this project for its simplicity and low performance impact.

The final consideration for this project's tool set is in its server hosting. Ankur Tiwari provides a justification for the use of *PythonAnywhere* for this project in the article, "In-Depth Review: Should You Host Python Websites on PythonAnywhere?"[12]. Among other benefits, the author specifically mentions the ease of deployment for Flask servers on *PythonAnywhere*. A Flask application is used in this project to receive user image uploads, so this is very useful. The author also describes the CPU usage limitations of the *PythonAnywhere* servers; these limitations will not be an issue for a pre-trained TensorFlow model.

Minimum Viable Product Deliverables

There are three main goals for this project to be met by the end of the first capstone semester. The first of these requires that the Android application must have a fully functional user interface. This includes the button layouts, application views, and user navigation; this is where all user interaction is handled within the app. This also includes the relaying of any information delivered from the server to the user, displaying the species classification and all relevant information effectively.

The next requirement for this semester is that the image capture process within the app needs to be fully formed. What this means is that the app must be able to access the camera and allow the user to take a picture. The image file then needs to be stored and held by the program in

preparation for uploading it to the server, at which point it will be passed along to the classification model.

These server/client communications make up the last deliverable for the MVP. The server, running through the *PythonAnywhere* service, will use a Flask web application for input and output handling. For this stage of the project, the server has three tasks: receive image files sent from the user's phone to the server, handle prediction information being sent from the server to the phone, and interact with a dummy classification model. This dummy model takes an image as input and returns text data, allowing it to fit into the same project space as the final, real image classification model, which will have the same input and output formats. The dummy model's only relevance to the project is as a placeholder, allowing for more extensive testing of the server architecture while the final model is still in development. Each of these three requirements for the MVP come with extensive testing to ensure that they are as bug-free and fully developed as possible. This testing is imperative to the success of the project, and must be done before moving on.

With this understood, there are other parts of the final project that could reasonably be approached in the first semester as well. One of these stretch goals is to begin the initial development of a TensorFlow model and slot it into the dummy model's place. Another important requirement of the project that needs to be addressed is the sourcing of training data for the classification model. The project will need to identify the list of spider species it will be training for, and it will need to find a sufficient amount of images for each of these species to train the model. The initial target for this data is at least 100 images per species.

Final Project Deliverables

Upon completion, this project will provide a fully functional Android application, an accurate image classification model (including training data), and the server code for communicating between these two components.

The Android app will handle all user interaction with the project. This will include three application activities (or screens): the main page with a prompt for the user to open the camera (fig.1), the internal camera view (fig.2), and the results returned by classification model (fig.3). The final project will have an updated, more attractive design; the figures below are preliminary UI mockups and are not representative of the final design. These mockups show the functionality provided by the app and the process the user will go through when using the app. Internally, the app will be able to capture and save images from the phone's camera, upload an image to the web server, receive the classification prediction from the server, and present the results to the user effectively.

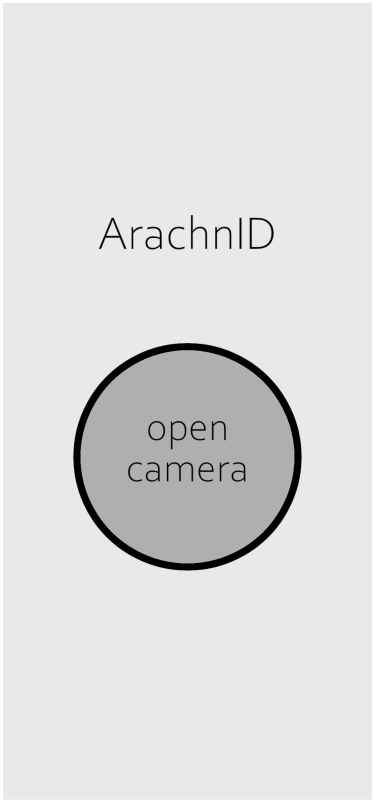

fig. 1

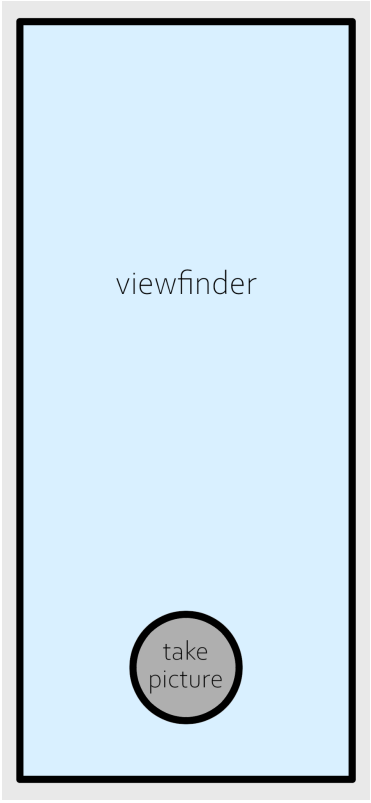

fig. 2

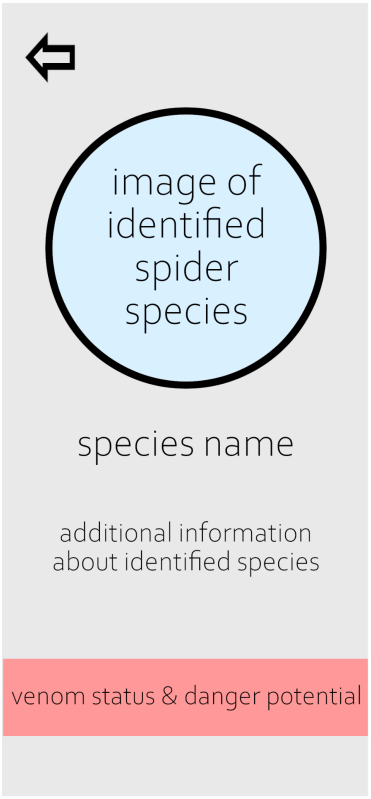

fig. 3

The final image classification model will use a Convolutional Neural Network built with the TensorFlow library. Training data for the model will be gathered from the website *Spider ID* and other online sources. This data will be in the form of close-up, clear photographs of each spider species. The classification model will be developed in Python, the language most supported by the

TensorFlow platform and offering the greatest extent of data science and computer vision tools for a project such as this. The server code will also be written in Python, using a Flask web application to communicate between the model and the user's phone. These two programs will be hosted on *PythonAnywhere*, a web hosting and development service that facilitates the deployment of Python code. All of these software components will work together to create a cohesive user experience and useful species classification tool.

Methodology

This project involves multiple distinct pieces working together, with each having their own development process. Each of these unique software environments has its own learning curve and workflow acclimation. For these to not cause delays, the project has to keep to a coordinated development schedule. For the first semester, this schedule gives a timeline of five development focuses.

First semester targets and the MVP

Before anything else could begin, the project needed an inventory of each tool that would be used. Android Studio is used for programming the app itself, which is in Java. GIMP is used for some of the design of UI components. Python code, particularly the TensorFlow platform, is used for the development of the image classification model. This list of tools is largely immutable; these are the tools necessary to complete this project. The component that had the most varying options was the question of where and how to run the image classification model. After considering multiple options for this, including running the model on the user's phone itself, using an FTP server hosted from a personal computer, multiple web application frameworks, and using UNCA campus computers for hosting, it was decided that running a Flask application hosted on *PythonAnywhere* provided the

best efficiency and capability. A rudimentary Flask application was developed and the *PythonAnywhere* hosting was set up by the end of September.

The project needed to ensure that a classification model could be run on this server configuration. For the sake of testing this capability, as well as for later testing of the server communications, a dummy classification model was developed. This program is written in Python and has the same input/output functionality that the final TensorFlow model will have. Instead of performing any image recognition and prediction, the program simply reads in an image file and calculates the dominant color value across all of the image's pixels. It returns a text output classification of either "Red", "Blue", or "Green". This simulates the final model's output of species names as strings, which lets us test the app's capability of handling different responses from the server. The dummy model was developed very quickly. Total development time for this part was limited to a single day, with the bulk of this time being spent reviewing OpenCV documentation[13] - review that will be useful for the final model's development as well.

By mid-October, the project shifted focus to Android development. To begin work on the server communications, namely the image upload process, there needed to be an image on the Android device to upload. As such, the image capture functionality of the Android app was the next bottleneck of development. A week was spent learning the Android Studio development process, and the next three weeks were spent building the foundations of the app. This includes both the initial home screen as well as a viewfinder screen with access to the device's camera (figures 1 & 2 above). The latter activity is complete with image capture and file saving functionality.

Beginning in the second week of November, the focus of the project is to develop and finalize the server/client communication between the server and the phone. This entails sending image files from the phone to the server, passing these files to the classification model, and sending the resulting text data from the server to the phone. On the Android side, the project uses the OkHTTP library to send and receive data from the server. The Flask application hosted on the server

manages incoming and outgoing data, as well as the server-side file system. All server communication is to be finalized by the fourth week of November.

The last week of development before the submission of the MVP will be spent finalizing the functionality of the Android app. The only part of the MVP left to implement at this point is the final screen of the app, used to display information to the user about the species identified in the user's photo (fig. 3). Final design considerations are not included in this stage; the sole focus is to display all of the relevant information to the user in an easily understandable way. This stage should not take more than a day or two of work, leaving the rest of the time before submission available to test the project and prepare it for demonstration.

Testing the MVP

While each component of the MVP receives adequate unit testing alongside the initial development stages, there is a significant amount of testing to be done after each part is functional. Integration testing between the Flask application and the dummy classification model, both running within the *PythonAnywhere* environment, received focus during the development of the dummy model. Integration testing between the entire project information flow will take up the majority of the remaining time before the MVP deadline. This testing will involve ensuring images captured by the Android app are uploaded to the server properly, with connections and subprocesses being terminated at the end of transmission. The server has to receive the image, send it to the classification model, and send text back to the user's phone. When this data is received in full by the phone, testing will focus on reproducibility; the app needs to complete the same process over and over for any arbitrary number of uses.

Beyond the MVP

Between the end of the Fall semester and the beginning of the Spring semester, the primary focus of the project will be to further design the UI elements of the Android app. Making the app more attractive to the users is a part of designing a product worth using. This is, however, an aspect of the project that resides quite low on the priority list; functionality will always take precedence. The winter break presents a good opportunity to address this task without impeding on the development schedule. This time will also be spent collecting training images for the TensorFlow model.

Beginning in the second semester, the project will focus entirely on the TensorFlow model. There are three stages for the development of the final classification model. The first stage is establishing the database of training and testing images. K-fold cross validation will likely be used to train and validate the model, so this will consist of a single data set. Some of this collection process is delegated to spare time in the previous semester and throughout the break, so this process should be limited to the first couple of weeks of the semester. The next stage is image preprocessing. This will include formatting every image to a uniform nature: centered subjects (i.e. the spiders) and common dimensions (i.e. size and ratio). This stage also includes any contrast modification, denoising, binarization, or other more advanced image processing techniques that may follow. Determining the best image format for the training data is a consideration that cannot be completed entirely independently of the development of the model itself; these stages will have significant overlap. Regardless, the preprocessing stage should be at a substantially developed point by late February. Development of the final TensorFlow model will have the following 4 to 6 weeks to reach an acceptable product. The remaining weeks before the final deadline will be spent testing the integration of the TensorFlow model into the larger project. This testing process will have been made much simpler by the testing done in the first semester, enabled by the dummy model filling the same project space as the final model.

Hardware and software limitations

The project takes into account the hardware and software requirements for running the software. Training the TensorFlow model is the most computationally intensive piece of the project. This will only need to be done once for the final product (excluding any testing) and will not need to be done on any user's own hardware. The other straining aspect of the software is the prediction process, when the model is used to classify a user's image. This process will be run on the *PythonAnywhere* servers, using server-grade hardware, and therefore imposes no minimum hardware limitation on the user. The Android app itself is relatively lightweight, with little computation occurring on the user's phone.

Despite there being practically no hardware requirements for the user, the software is developed with specific environments in mind. The app will only be runnable on Android devices; this includes phone manufacturers like Samsung, Google, etc. This is an intentional limitation of project scope, as well as a limitation brought about by the prohibitive development costs associated with developing for iOS devices. Software support targets Android version 8.0 and above, although the specific versions supported are liable to change by the end of the project. Android version 8.0, "Oreo", and above have a market share of 84.9% of Android devices as of November, 2021[14]. Version support is a balance between maximizing the potential user base while still utilizing modern libraries and functionality, notably the full *CameraX* API and the *OkHTTP* library.

The app requires an internet connection to function. The image classification is not run locally on the user's device, so the web server needs to be able to communicate between both the model and the user's phone over the internet. There are two primary justifications for this architecture: guaranteeing hardware capabilities for the computation, as mentioned previously, and allowing for future expansion and advancement of the project. Should the classification model be upgraded in the future, it would only need to be updated on the server, rather than on every installation of the app. This also allows for the implementation of new models trained on other

regions besides western North Carolina without having to increase the size of the app's installation. The app's upload to the server would then simply include a location tag to determine the right model to use. The dependency on internet access is not seen as a major limitation of the project's capabilities due to the nature of the problem being solved; this tool is primarily developed for the identification of spiders within a home, rather than in nature. It can be reasonably assumed that an internet connection will be available in the majority of these spaces. This is the extent of the hardware and software requirements for this project.

References

1. *PythonAnywhere*, https://www.pythonanywhere.com/.
2. "How Lens Works." *Google Lens*, https://lens.google/howlensworks/.
3. Stewart, Matthew. "The Most Important Supreme Court Decision for Data Science and Machine Learning." *Medium*, Towards Data Science, 29 July 2020, https://towardsdatascience.com/the-most-important-supreme-court-decision-for-data-science-and-machine-learning-44cfc1c1bcaf.
4. "Spiders in North Carolina." *Spider ID*, https://spiderid.com/locations/united-states/north-carolina/.
5. Ticay-Rivas, Jaime R, et al. "Image Processing for Spider Classification." Biodiversity Conservation and Utilization in a Diverse World, 2012, https://doi.org/10.5772/50341.
6. M., Sanjay. "Why and How to Cross Validate a Model?" *Medium*, Towards Data Science, 19 Aug. 2020, https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f.
7. Kumar, Satyam. "Understanding 8 Types of Cross-Validation." Medium, Towards Data Science, 13 Sept. 2020,

https://towardsdatascience.com/understanding-8-types-of-cross-validation-80c935a4976d

8. Pak, Myeongsuk, and Sanghoon Kim. "A Review of Deep Learning in Image Recognition." *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, 2017, https://doi.org/10.1109/caipt.2017.8320684.
9. ul Hassan, Muneeb. "VGG16 - Convolutional Network for Classification and Detection." *VGG16 - Convolutional Network for Classification and Detection*, 24 Feb. 2021, https://neurohive.io/en/popular-networks/vgg16/.
10. "Developer Guides." *Android Developers*, https://developer.android.com/guide.
11. Bhade, Jaydevi, and Himanshu Yadav. "Evaluation and Identification of Android Networking Libraries." *International Journal of Scientific Research & Engineering Trends*, vol. 5, no. 3, 2019.
12. Tiwari, Ankur. "In-Depth Review: Should You Host Python Website on Pythonanywhere?" *Medium*, Thoughtlytics, 30 Aug. 2020, https://medium.com/thoughtlytics/in-depth-review-should-you-host-python-website-on-pythonanywhere-594c31428336.
13. "OpenCV Modules." *OpenCV*, https://docs.opencv.org/4.x/.
14. "Android OS Version Market Share Over Time." *AppBrain*, Nov. 2021, https://www.appbrain.com/stats/top-android-sdk-versions.