Western Governors University

Downtown Botanical Garden Plant ID

WGU BSCS Capstone Project

Misty Hurley

6-28-2021

CONTENTS

JUNE 26, 2021

Misty Hurley
[redacted]

[redacted]
Downtown Botanical Garden
San Antonio, TX 78212

To [redacted] and [redacted]

I am writing in response to the RFP issued by Downtown Botanical Garden regarding the volume of plant identification inquiries becoming too cumbersome for staff to respond to in a timely manner. I understand guests have become frustrated by what they see as lack of engagement, while your organization struggles to serve guest interests.

Alamo Software Solutions has considered a number of strategies to address your organization's needs, and we would like to propose a plant identification application that may be used by guests to automatically classify plants without the need for staff interaction.

We believe the majority of the plant identification inquiries may be self-served through this application without the need for interaction with Downtown Botanical Garden staff, allowing staff to focus on more productive duties in the workplace. To return a fast deliverable, we propose a small dataset of commonly requested plants be included in the initial version of the application, with later expansions to include more species and subspecies in the future. Estimated funding for this project is $25,000. This includes application development, testing, and deployment over a period of one month.

Attached to this letter, you will find our proposal, executive summary, and other application details. I am confident that this project will significantly improve staff productivity, guest satisfaction, and sales. Please do not hesitate to contact me with any questions or concerns. We thank you for your consideration and are eager to work with you on this exciting project.

Regards

Misty Hurley

[redacted]

[redacted]

A. PROPOSAL

SUMMARY

Downtown Botanical Garden is a 40 acre establishment featuring a rotating selection of thousands of native, cultivated, and exotic species and subspecies. Guests often express interest in cultivating many of these plant varieties. While the more exotic, ecologically important, and popular species are labeled with signs and plaques, a variety of rotating and supporting species are not labeled. Signage for each species would clutter the arrangements and detract from the natural beauty of the environment. Thus, staff are often inundated by inquiries requesting identification of plant varieties, questions that are not easy to answer considering the extremely large volume of data that would need to be frequently accessed and updated.

The solution for this problem can be solved through a machine learning application that utilizes deep learning to classify flowers by image. Guests may simply photograph the desired plant, and the product will return the most likely species or subspecies as well as a confidence level. The app will automatically answer most customer identification questions, freeing up staff to focus on more productive duties in the workplace.

Our team aims to deliver a user-friendly, functional application that can correctly identify the supported flower varieties with above 80% accuracy, as well as a confidence score that may signify the need for staff intervention. For a fast deliverable, we will initially support a small dataset of the most commonly requested species and subspecies. Support for more species will require further negotiation.

BENEFITS

- Improve company reputation: Guests will enjoy immediate responses to plant identification inquiries.
- Increase productivity: The app will filter recurrent inquiries from staff workload, allowing personnel to focus on more productive duties in the workplace.
- Scalable: A small dataset will be initially supported for a fast deliverable, with more species to be added incrementally as needs arise.
- Reusable: Deep learning models can be trained on additional data without the need to rewrite code.

APPLICATION OUTLINE

There are two parts of the proposed prototype: a *frontend* application that a user interacts with, and a *backend* program that runs the logic behind the scenes. The backend will consist of a multilayer machine learning model trained to identify the five initial plant varieties with at least 80% accuracy. The frontend will consist of a simple, standalone application that allows users to upload a photo, which the application will then classify as one of the five supported plant varieties.

DATA DESCRIPTION

This prototype will utilize an open source dataset containing 4242 total images, split into five classes: daisy, dandelion, rose, sunflower, and tulip (Mamaev, 2018). This dataset will be split into training, validation, and testing sets in order to train the model, validate, and test for accuracy. Data cleaning methods will be utilized to remove outliers in order to produce an accurate model. Images will be transformed with a variety of rotations, light adjustments, and angles during training to adjust for the amateur photography skills of the average consumer.

Expansions to the program will mine data from a variety of sources. Our team will utilize primarily open-license and unrestricted images. Downtown Botanical Garden is a nonprofit organization with a mission to educate and inform the public, thus free, unrestricted use of images is indicated in the image use policies of many nonprofit and educational institutions, which will significantly reduce data collection costs. One such example is Wildflower.org, which provides a gallery of 81,144 plant images, many of which are verified by botanists and horticultural experts (Lady Bird Johnson Wildflower Center, n.d.). Potential limitations of this data collection method include disproportionate data subsets for certain plant varieties and a higher incidence of mislabeled data from nonprofit institutions. These limitations may be mitigated with a small budget set aside for image license fees, which our team estimates can be covered with $1,200 per 100 species.

OBJECTIVE, HYPOTHESES

The primary objective of the proposed solution is to deliver a user-friendly, functional application that can correctly identify the supported plant varieties. Successfully meeting this objective is expected to:

- Immediately answer most customer identification queries
- Improve Downtown Botanical Garden's reputation with guests
- Improve season pass and ticket sales
- Increase guest recommendation rates
- Free up staff to focus on more productive duties

METHODOLOGY

Our team utilizes an Agile methodology of software development in order to deliver working product quickly, while communicating with our clients transparently and collaboratively (See Figure 1). We believe this methodology adapts perfectly to the proposed design of this project, as delivering visible results quickly through our initial prototype keeps stakeholders excited for this project and looking forward to further improvements. When applied to machine learning projects, agile methodologies have been shown to reduce development timelines from months to weeks (Dougherty, 2019).

Fig 1: The Agile Manifesto (Lichtenberger, 2014).

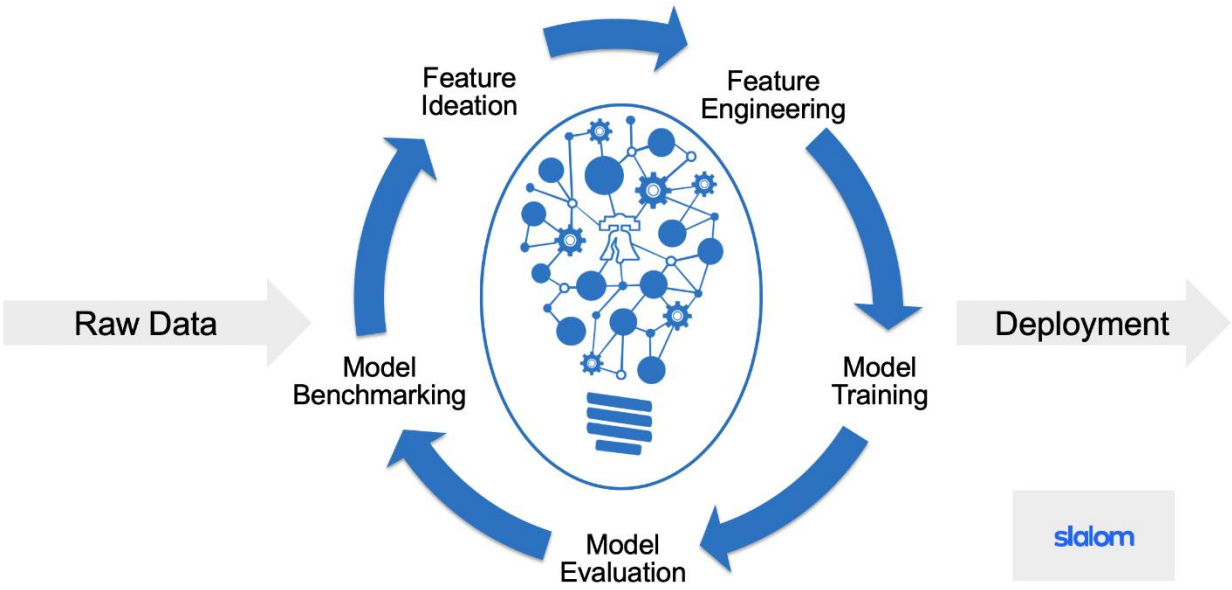


Fig 2: Agile for Machine Learning (Dougherty, 2019)

FUNDING

The proposed solution will require initial funding of $19,000 for our working prototype developed in our in-house facilities over a period of one month. Additional expansions are estimated to cost $25,000 per month and will cover mobile app development, hardware, servers, labor, image licensing, and model training per 100 species and subspecies. Following completed development and deployment, servers, maintenance, and support will cost approximately $20,000 per year.

STAKEHOLDER IMPACT

Positive public sentiment is crucial for a nonprofit organization to operate smoothly and successfully, and it is imperative that visitors continue to see Downtown Botanical Garden in a positive light. Providing a free application to the public is a form of advertising that can put Downtown Botanical Garden in the spotlight. Encouraging visitors and supporters to

identify and learn about various plant species and subspecies aligns with Downtown Botanical Garden's commitment to biodiversity and connection to the natural world.

PRECAUTIONS (SENSITIVE)

This project is classified as low risk, as the data is primarily public and not easy to lose. The loss of confidentiality, integrity, or availability would have little or no adverse impact on the organization's mission, safety, finances, or reputation. User-based sensitivity will be handled by auto-scrubbing EXIF location data from and auto-blurring human faces in photos prior to uploading to avoid risk of sharing personally identifiable information.

DEVELOPERS QUALIFICATIONS

Alamo Software Solutions is a rapidly expanding company that has been serving our clients for over 5 years, providing our clients with web and mobile products in AI, ML, DL, and data science services. Each member of our development team holds a bachelor's or master's degree in computer science or a related field and several years of experience.

B. TECHNICAL EXECUTIVE SUMMARY

PROBLEM STATEMENT

Visitors to the Downtown Botanical Garden often find themselves interested in cultivating some of the displayed species. While many popular species are labeled with signs and plaques, there are many species that are not and cannot feasibly be labeled. Staff complain of the volume of inquiries congesting workflow, while guests complain of the extended inquiry turnaround time. A technical solution involving deep learning may be utilized to automatically classify plant images to automatically answer many such inquiries, satisfying guests and freeing staff to work on more productive duties.

USER SUMMARY

Userbase for this application will consist of all visitors to the Downtown Botanical Garden, including educational tours, event attendees, and members of the public. Under the current system, visitors intercept personnel or send inquiries requesting identification of various plant species and subspecies. The proposed solution will be a simple, intuitive application that allows visitors to self-serve these inquiries by uploading an image that the application will classify as the appropriate species or subspecies.

EXISTING SYSTEM ANALYSIS

The current system includes signage for a number of the more popular, ecologically important, and exotic species. Signage for all species is not a suitable solution, as this will detract from the beauty of the natural environment and become too arduous to maintain due the varietal offerings of rotating species and companion plants. Under the current system, visitors intercept personnel or send inquiries requesting identification of various plant species and subspecies. A DBMS was proposed as a possible solution; however, this database would be expensive and laborious to maintain due to the thousands of species and subspecies offered, coupled with the frequent changes due to display adjustments, availability issues, and fluctuating seasonal offerings. The proposed solution will allow

visitors to self-serve inquiries, satisfying guests and freeing personnel to focus on more productive duties.

PROJECT METHODOLOGY

The proposed solution will utilize an Agile Scrum framework of software development adapted to cover all machine learning steps. Each release cycle will be split into four 1-week sprints, each divided into exploratory data analysis (EDA), model tuning, and dashboard development. As indicated in Figure 3, data analysis requires an initial extra time investment that is later fine-tuned and realigned in subsequent sprints. As analysis and model training is refined, more time is spent developing the dashboard. Within each sprint, various meetings will occur to generate a healthy flow of communication between client, project management, and the development team (See Table 1).

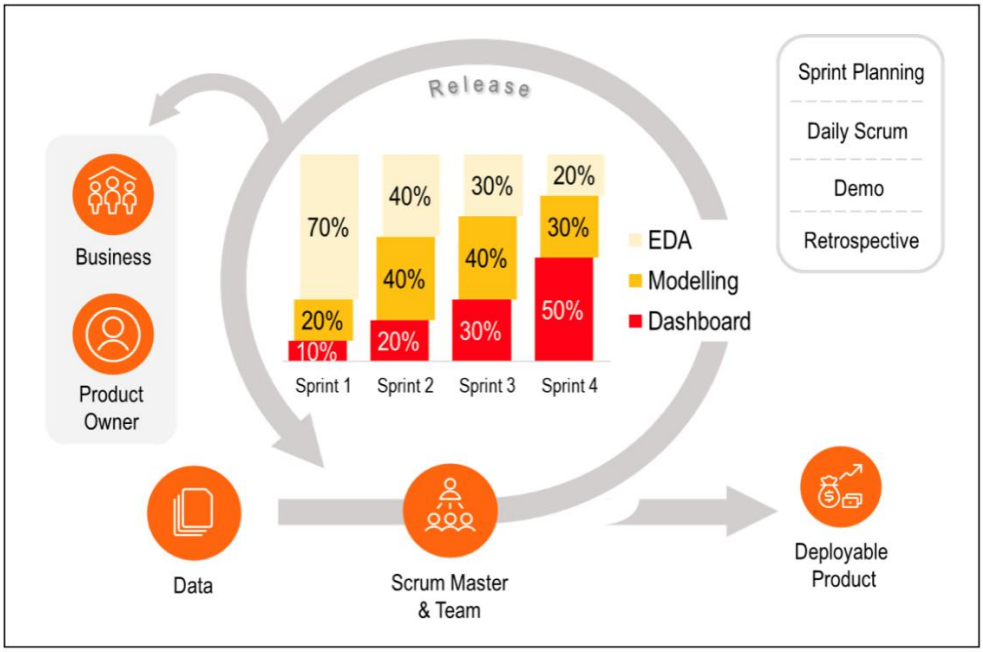


Fig 3: A simple release schedule of 4 sprints (SAURABH9745, 2020).

	Attendees	Purpose	Frequency (per sprint)
Sprint planning	- Client reps - Project manager - Development team	1 hour meeting to discuss goals for the sprint	Once, at the beginning of each sprint
Daily scrum	- Project manager - Development team	15 minute meeting to discuss daily goals and collaboration	Daily
Sprint review	- Client reps - Project manager - Development team (partial)	1 hour meeting to showcase completed work and discuss possible adaptations	Once, at the end of each sprint
Sprint retrospective	- Client (optional) - Project manager - Development team	30-60 minute meeting to reflect on work and discuss improvements	Once, after each sprint review

Table 1: Communications per sprint

PROJECT DELIVERABLES

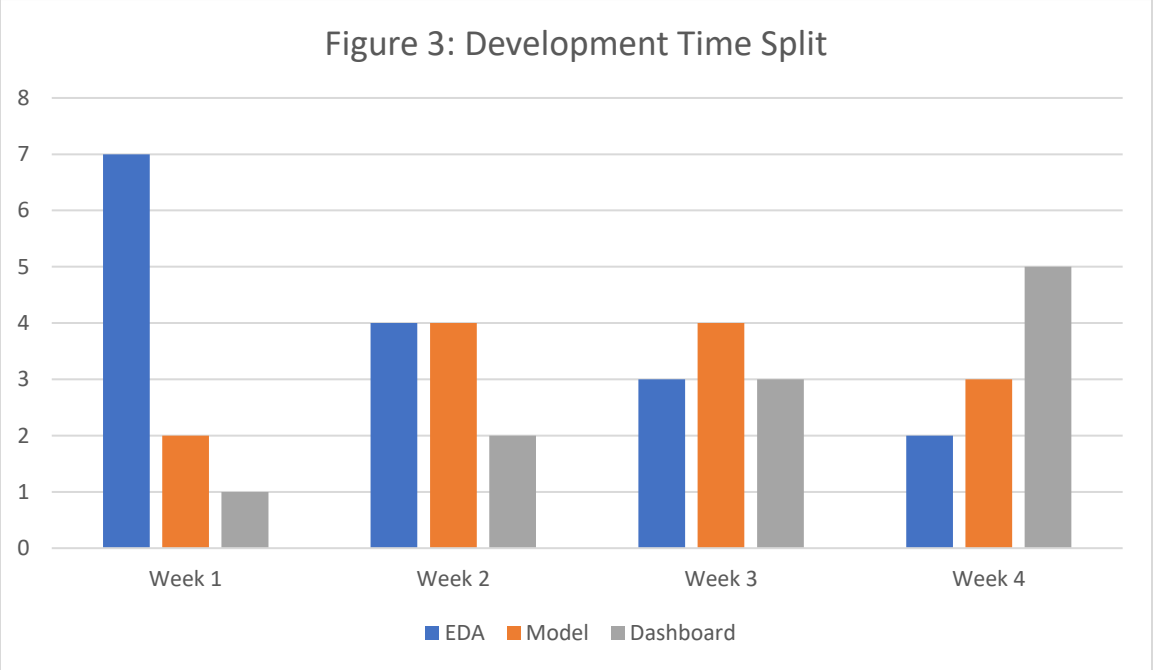
- User-friendly, functional application
- Correctly identify supported varieties reduce the need for extensive engineering hours.
- Simple, end-to-end trainable models reduce the need for extensive engineering hours.
- Scalable: A small dataset will be initially supported for a fast deliverable, with more species to be added incrementally as needs arise.
- Reusable: Deep learning models can be trained on additional data without the need to edit code for additional heuristics.

IMPLEMENTATION PLAN

Refer to Table 2 and Figure 3 for implementation plan details:

Week	Primary objectives and outcomes
0	Kickoff meeting
1	Dataset analysis, load and clean data
2	Develop and train model
3	Working prototype; begin testing
4	Testing phase, revisions
5	Deploy, review; discuss next steps

Table 2: Implementation plan



EVALUATION PLAN

The dataset will be split into training, validation, and testing sets. The validation set serves to automate validation during the model building phase. The model will be periodically validated with manual testing using the test set outside of model building. Week 4 will comprise the testing phases depicted in Figure 4. Prior to deployment, a final usability test will occur with a sampling of end users to evaluate the prototypes features and ease of use.

Figure 4: Levels of Testing (UTOR, 2020).

RESOURCES AND COSTS

The programming environment will consist of freely-available, open-source tools: $0

- Anaconda Individual Edition for Python machine learning
- Jupyter Notebook
- TensorFlow machine learning platform
- Keras deep learning API
- NumPy and other open-source Python libraries

Human resource requirements per 1 month prototype development:

Alamo Software Solutions project manager and in-house development team: $19,000

Total estimated costs: $19,000

The proposed solution will require initial funding of $19,000 for our working prototype developed in our in-house facilities over a period of one month. Additional expansions are estimated to cost $25,000 per month and will cover mobile app development, hardware, servers, labor, image licensing, and model training per 100 species and subspecies. Following completed development and deployment, servers, maintenance, and support will cost approximately $20,000 per year.

TIMELINE AND MILESTONES

Milestones	Notes
Kickoff meeting	Hold 1 hour meeting to kick off the project and plan for sprint 1
Sprint 1	Dataset analysis, load and clean data
Sprint 2	Develop and train model
Sprint 3	Working prototype; begin testing
Sprint 4	Testing phase, revisions
Deployment	Deployment phase to occur at 4 week end mark
Review	Hold review meeting between client, project manager, and senior engineer to discuss results and next steps

Table 3: Timeline and Milestones

C. APPLICATION DETAILS

DATA METHODS

A convolutional neural network was selected as the nondescriptive method. There are a number of built-in descriptive methods utilized within the implemented convolutional neural network model. ResNet50 was utilized as the base model with added cascading transformations via Dropout, GlobalAveragePooling2D, and Dense layers. The ResNet50 base model features 23,597,957 total parameters to classify image data. To freeze the convolutional base in order to prevent weights in each layer from being altered during training, applied layers were flagged as non-trainable, resulting in a total of 10,245 trainable parameters. This unsupervised learning approach negates the need for manually programmed descriptive methods (Bonaccorso, 2019).

DATASET

An open dataset was sourced from Kaggle.com and contained 4242 total images, split into five classes: daisy, dandelion, rose, sunflower, and tulip (Mamaev, 2018). This dataset may be accessed via the 'flowers_dataset' folder in the project files or downloaded from the source indicated in the References section.

ANALYTICS

The solution resulted in a successfully trained model that can classify the user's uploaded flower image into one of five currently supported plant varieties, allowing the user to self-serve plant identification queries without the need to request personal assistance from personnel.

DATA EXPLORATION, CLEANING, INSPECTION, AND WRANGLING

A variety of methods were used to explore, clean, inspect, and wrangle the data:

- The preprocessed dataset was split into training, validation, and testing sets with a script based on the split-folders Python module.
- Images were transformed (rotation, zoom, shear, flip) during model training via ImageDataGenerator from the Keras API in order to adjust for directionality.
- Images were normalized to 224x224 using ImageDataGenerator directory iterator to best fit the ResNet50 base model
- Manual inspection of data to clean outliers from the training set (e.g. women named 'Rose' in the 'rose' dataset).

DATA VISUALIZATION

There were 3 data visualizations utilized:

- Uploading and classifying an image – **bar chart** displaying relative confidence to other flower classes
- Confidence history – a **line plot** of the last 10 confidence percentages of identified images during the current session, along with an average confidence for the session
- Session statistics – a **bar graph** that tracks count of each flower identified per session

INTERACTIVE QUERIES

Users may view and interact with the data in the dashboard in a variety of ways:

- Uploading and classifying an image reveals a bar chart that displays relative confidence to other flower classes. In instances with relatively low confidence, this allows the user to discern whether the runner up species may be a more appropriate classification. In the case that confidence is distributed more evenly, this may indicate that the chosen flower is an unsupported variety in the current model.
- The user may click the 'Confidence History' tab in the application to open the confidence history graphic, a line plot of the last 10 confidence percentages of identified images during the session, along with an average confidence achieved during the session.
- The user may click the 'Statistics' tab in the application to track count of each flower classified during the session, displayed as a bar graph. This allows the user to easily visualize the distribution and number of flower classes scanned per session.

MACHINE LEARNING METHODS AND ALGORITHMS

The product utilizes a convolutional neural network (CNN) with a transfer learning framework. Transfer learning is an approach to deep learning that begins with a general base model that is pre-trained in computer vision, then tunes the model with a cascade of layers applied to the base model. The base model and layers are sourced from the Keras API.

Relevant methods utilized:

- Base model: **ResNet50**, a CNN model built into the Keras API with parameters customized for transfer learning and image classification
- **Dropout**: a layer that randomly sets inputs to 0 to ignore (drop out) neurons during training. The effect is that the model does not depend too much on the specific weights of neurons, improving generalization and reducing overfitting.
- **GlobalAveragePooling2D**: A layer that reduces overlapping in pooling in order to continuously reduce the spatial size (parameters and computation) of the model, thus reducing overfitting.
- **Dense**: The deeply connected neural network layer outputs the dot product between the input tensor and the weight kernel matrix.

OUTCOME ACCURACY

Model training was auto-validated using the validation set from the processed data and revealed an accuracy of 90.35%. The final accuracy was determined using the test set that was not introduced during model training and was calculated at 90.40%.

SECURITY FEATURES

The current version of this product is a stand-alone application that the user may run on a device that supports Python executable files and that does not require internet access. Users are expected to properly utilize the operating system's built-in security features and practice strong password techniques.

Future versions of the product will implement network features, require a username and strong password, scrub EXIF data from uploaded images, and auto-blur human faces from uploaded image files. However, this is beyond the scope of this prototype.

MONITORING AND MAINTENANCE

A print statement is used to dump classification data to the console to monitor the health of the model and ensure the program is performing as intended.

USER DASHBOARD

The following figures depict the user dashboard experience:

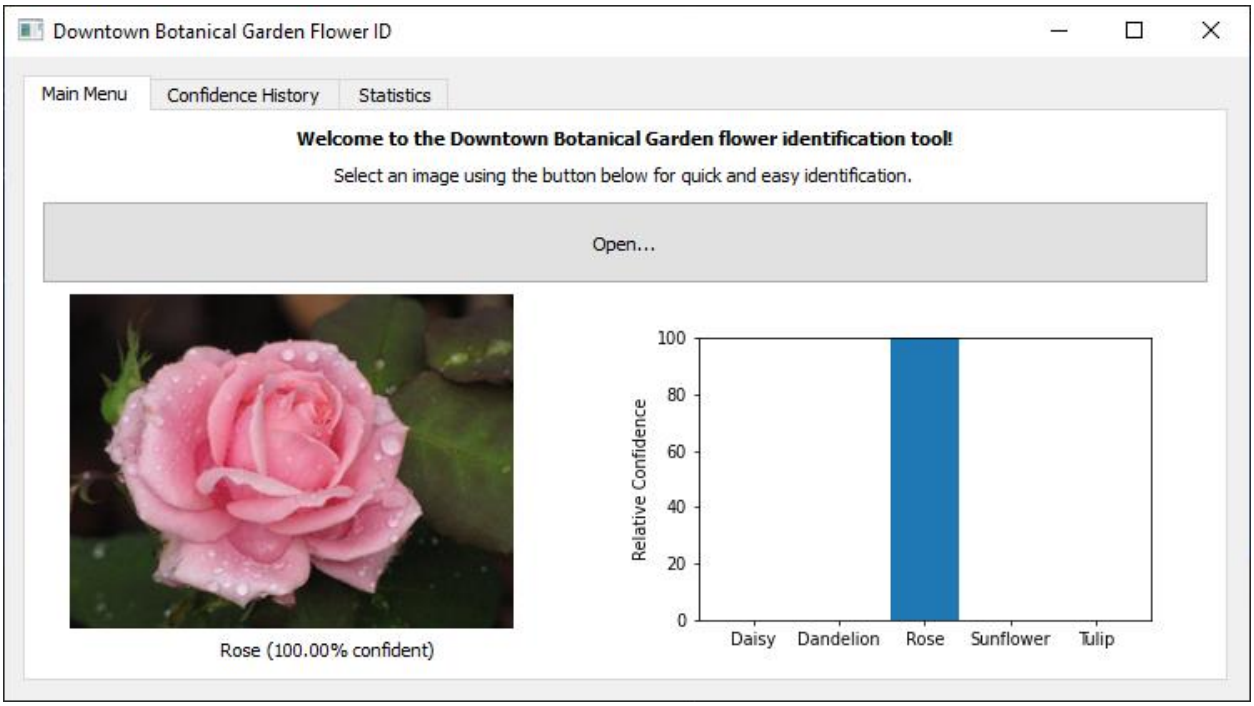


Figure 5: Main Menu with upload button and measure of confidence

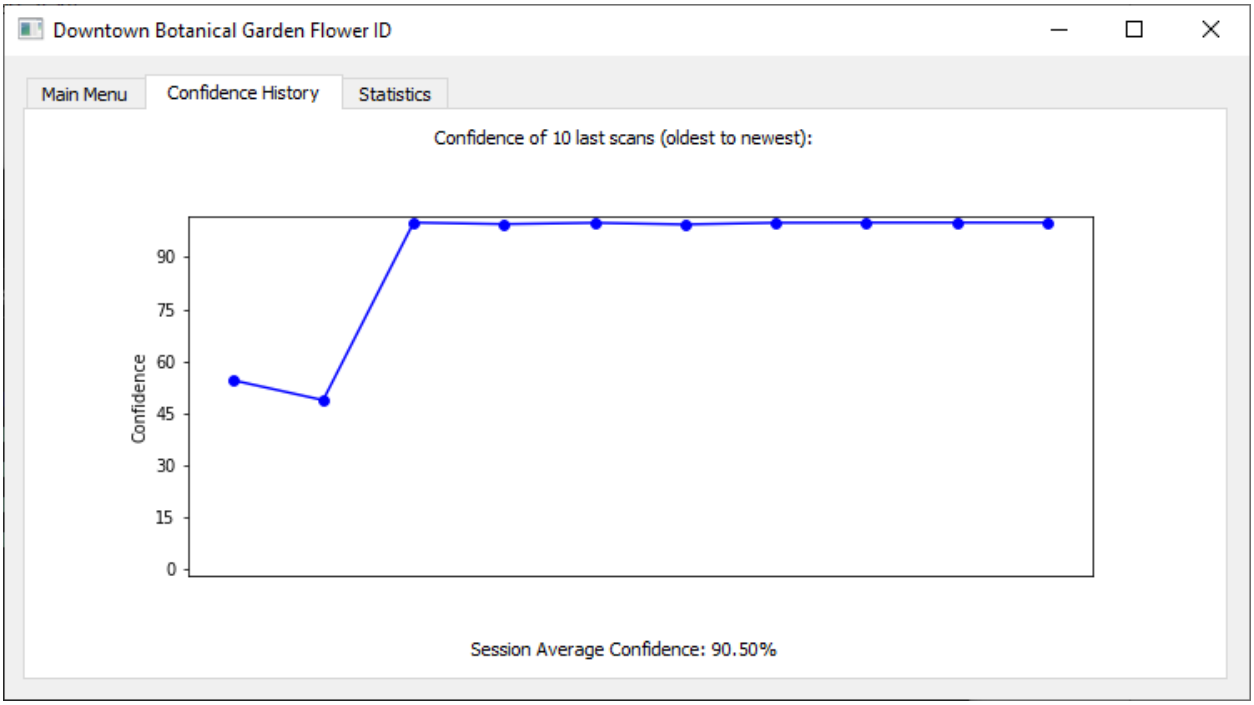


Figure 6: Confidence History tab with line plot visualization

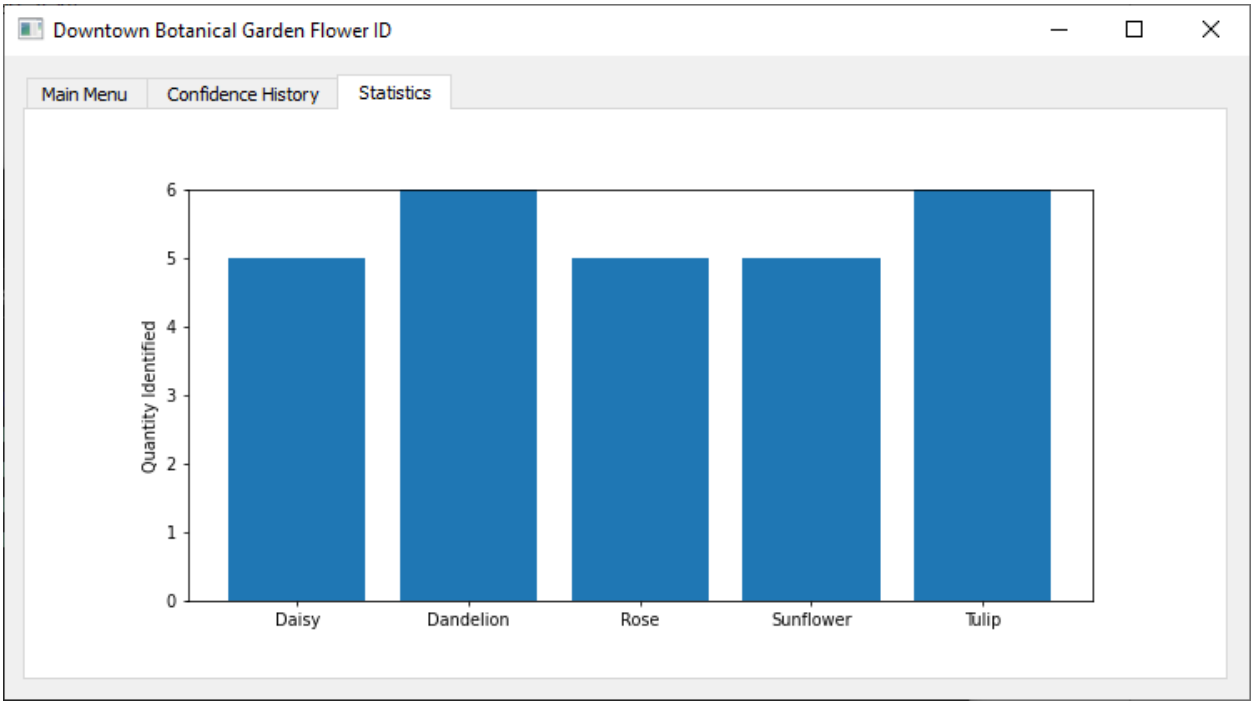


Figure 7: Statistics tab with Quantity Identified bar graph visualization

D. DOCUMENTATION

PURPOSE

The goal of this project was to build a product that would assist with plant identification inquiries from guests. A deep learning solution was implemented that satisfied this requirement by allowing users to upload flower images that the application would classify as one of the 5 supported classes in a simple, intuitive GUI format. The implemented solution successfully meets all outlined requirements.

DATASET

Note: Data is located in the 'dataset_flowers' directory. Test data may be accessed from the 'processed_data/test' subfolder.

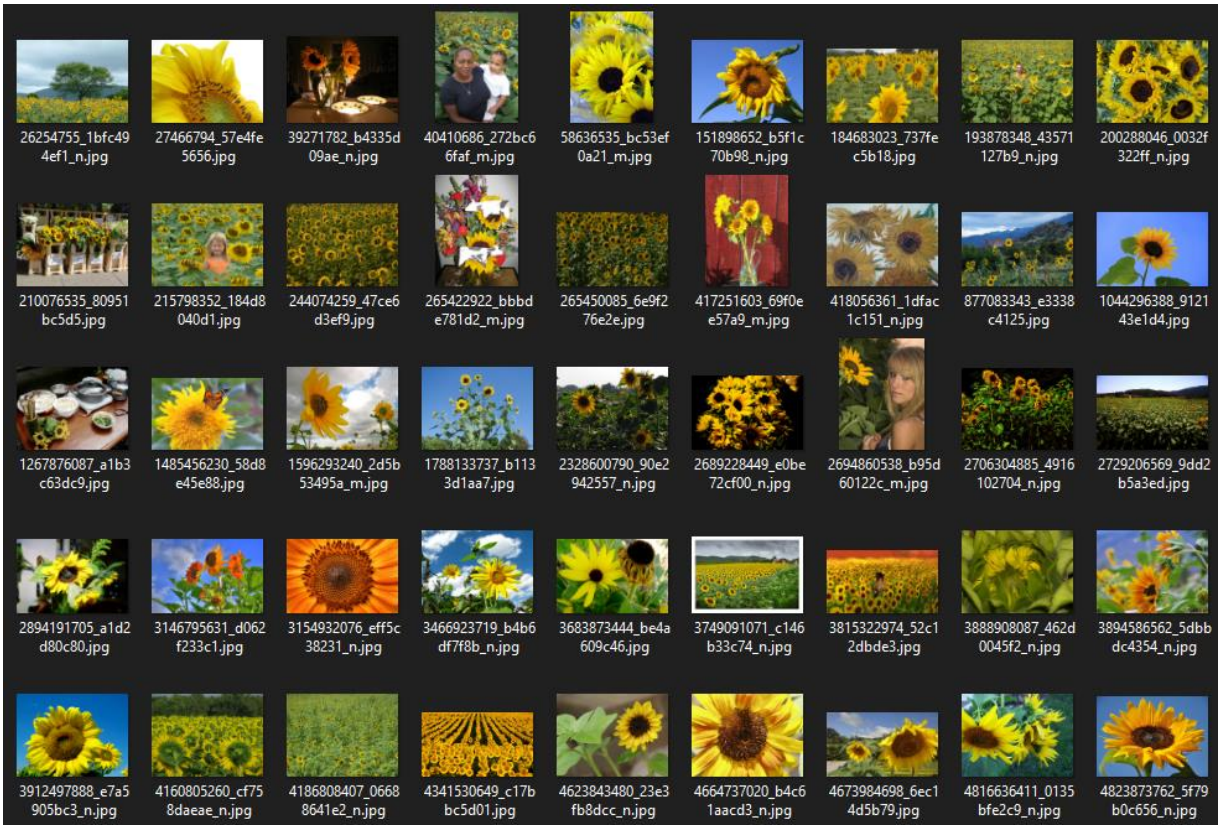


Figure 7: Sample of test dataset

DATA PRODUCT CODE

See attached code files.

Notes:

- 'preprocess.ipynb': pre-processing Jupyter script using split-folders to split the dataset into train, test, and validation subsets
- 'ModelGen.ipynb': Jupyter script used to train and build the model
- 'app/main.py': Dashboard application source code

HYPOTHESIS VERIFICATION

The hypothesis was that the model would achieve above 80% accuracy in identifying supported plant classes. Auto-calculated accuracy was over 90%, which met the proposed hypothesis. Final results tested over 90% confidence in the majority of randomly selected test images with near 100% accuracy.

EFFECTIVE VISUALIZATIONS AND REPORTING

Visualizations and reporting is covered in Section C above.

ACCURACY ANALYSIS

As stated above, auto-calculated accuracy was over 90%, which met the proposed hypothesis. Final results tested over 90% confidence in the majority of randomly selected test images with near 100% accuracy.

APPLICATION TESTING

During QA and user testing, the following revisions were made to the application:

- Results from the initially planned DenseNet-based model were unsatisfactory, so a new model was generated using ResNet50, which gave more consistent and correct results.
- "Open" file dialog would crash the application if the selection window was closed without selecting an image. This has been fixed.
- A "Relative Confidence" graph was added alongside the flower identification result to aid the user in identifying next best alternatives in the event the model is not able to provide an answer with a high degree of confidence.
- The "Confidence History" tab was added to provide a way monitor the model's confidence in its image classification as a measure of overall application performance.

```
In [17]: base_model = tf.keras.applications.resnet50.ResNet50(include_top=False, weights='imagenet')
         #base_model.trainable = False
         x = base_model.output
         x = Dropout(0.1)(x)
         x = GlobalAveragePooling2D()(x)
         predictions = Dense(train_generator.num_classes, activation='softmax')(x)
         model = Model(inputs=base_model.input, outputs=predictions)

         # mark precalculated base model layers as non-trainable; only our layers will remain trainable for future fitting
         for layer in base_model.layers:
             layer.trainable = False

         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['accuracy'])
         model.fit(train_generator, epochs = 10, validation_data = val_generator) # use separate validation generator

         Epoch 1/10
         82/82 [==============================] - 26s 291ms/step - loss: 0.7422 - accuracy: 0.7335 - val_loss: 0.4227 - val_accuracy: 0.
         8523
         Epoch 2/10
         82/82 [==============================] - 23s 283ms/step - loss: 0.3594 - accuracy: 0.8739 - val_loss: 0.3348 - val_accuracy: 0.
         8791
         Epoch 3/10
         82/82 [==============================] - 23s 283ms/step - loss: 0.2936 - accuracy: 0.8936 - val_loss: 0.3498 - val_accuracy: 0.
         8698
         Epoch 4/10
         82/82 [==============================] - 23s 283ms/step - loss: 0.2429 - accuracy: 0.9202 - val_loss: 0.3109 - val_accuracy: 0.
         8919
         Epoch 5/10
         82/82 [==============================] - 23s 283ms/step - loss: 0.2358 - accuracy: 0.9113 - val_loss: 0.2925 - val_accuracy: 0.
         9047
         Epoch 6/10
         82/82 [==============================] - 23s 282ms/step - loss: 0.1865 - accuracy: 0.9383 - val_loss: 0.3137 - val_accuracy: 0.
         8837
         Epoch 7/10
         82/82 [==============================] - 23s 284ms/step - loss: 0.1819 - accuracy: 0.9425 - val_loss: 0.2762 - val_accuracy: 0.
         9035
         Epoch 8/10
         82/82 [==============================] - 23s 285ms/step - loss: 0.1612 - accuracy: 0.9487 - val_loss: 0.2790 - val_accuracy: 0.
         9012
         Epoch 9/10
         82/82 [==============================] - 23s 283ms/step - loss: 0.1491 - accuracy: 0.9495 - val_loss: 0.2856 - val_accuracy: 0.
         9140
         Epoch 10/10
         82/82 [==============================] - 23s 282ms/step - loss: 0.1365 - accuracy: 0.9530 - val_loss: 0.2881 - val_accuracy: 0.
         9035

Out[17]: <tensorflow.python.keras.callbacks.History at 0x184a680a8b0>
```

Figure 8: Accuracy calculated via validation dataset

```
In [19]: test_result = model.evaluate(test_generator)
         print("loss, accuracy: ", test_result)

         865/865 [==============================] - 9s 11ms/step - loss: 0.2848 - accuracy: 0.9040
         loss, accuracy:  [0.2848333418369293, 0.9040462374687195]

In [23]: model.summary()
```

conv5_block3_3_conv (Conv2D)	(None, None, None, 2	1050624	conv5_block3_2_relu[0][0]	
conv5_block3_3_bn (BatchNormali	(None, None, None, 2	8192	conv5_block3_3_conv[0][0]	
conv5_block3_add (Add)	(None, None, None, 2	0	conv5_block2_out[0][0] conv5_block3_3_bn[0][0]	
conv5_block3_out (Activation)	(None, None, None, 2	0	conv5_block3_add[0][0]	
dropout_4 (Dropout)	(None, None, None, 2	0	conv5_block3_out[0][0]	
global_average_pooling2d_4 (Glo	(None, 2048)	0	dropout_4[0][0]	
dense_4 (Dense)	(None, 5)	10245	global_average_pooling2d_4[0][0]	

```
Total params: 23,597,957
Trainable params: 10,245
Non-trainable params: 23,587,712
```

Figure 9: Accuracy calculated via test dataset

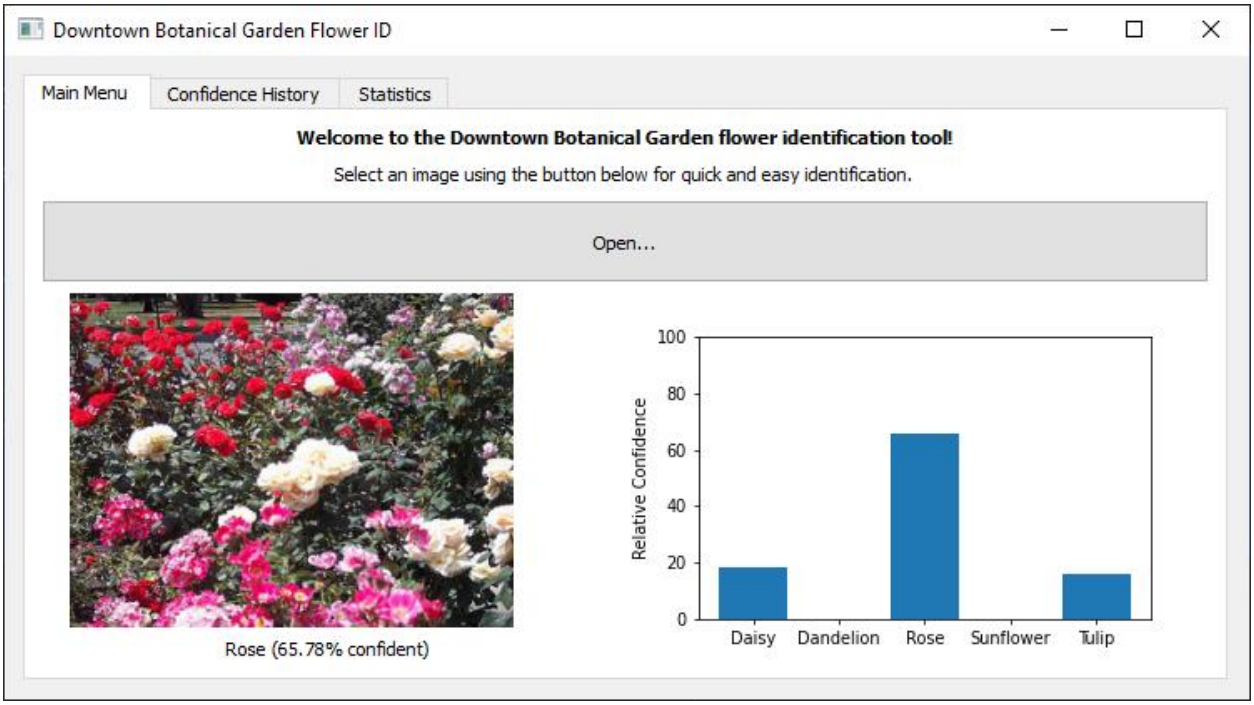


Figure 10: Sample depicting relative confidence metric

APPLICATION FILES

See attached code files.

Notes:

- 'preprocess.ipynb': pre-processing Jupyter script using split-folders to split the dataset into train, test, and validation subsets
- 'ModelGen.ipynb': Jupyter script used to train and build the model
- 'app/main.py': Dashboard application source code
- 'FlowerID.exe': Standalone executable version of application

QUICK START GUIDE

Instructions:

1. Download the program folders from Google Drive: https://drive.google.com/drive/folders/17K5BPWAoovBT2E4SV9JBFcW6yjYs1bzd
    a. 'FlowerIDApp.zip' contains the full program in .exe format (All dependencies included)
    b. 'mhurley flower id (full).zip' contains the full source code files, including 'app/main.py' but may require dependencies listed below to run
2. Unzip the file.
3. Open the folder
4. Run FlowerID.exe and follow installation instructions or run 'app/main.py'
5. Click the 'Open…' button and select an image using the file explorer.
    a. The test dataset in the 'processed_data/test' subfolder may be used to test the program.
    b. Alternatively, select an image from any source that depicts a sunflower, tulip, rose, dandelion, or daisy.
6. Click the other tabs to explore other data visualization results

Dependencies:

- Python 3.8
- NVIDIA GPU Computing Toolkit (CUDA v11.3)
- cuDNN 8.2.0 (for CUDA v11.3)
- TensorFlow 2.5.0
- Keras 2.5.0
- PyQt5
- Windows 10 (or equivalent)

REFERENCES

Bonaccorso, G. (2019, February). *Hands-On Unsupervised Learning with Python* [eBook edition]. Packt Publishing. https://learning.oreilly.com/library/view/hands-on-unsupervised-learning/9781789348279/

Dougherty, P. (2019, May 15). *How and Why to Use Agile for Machine Learning.* Medium. https://medium.com/qash/how-and-why-to-use-agile-for-machine-learning-384b030e67b6

Lady Bird Johnson Wildflower Center (n.d.) *Image Use Policy.* https://www.wildflower.org/plants-main/image-use-policy

Lichtenberger, A. (2014). *What IT Service Management Can Learn from the Agile Manifesto (And Vice Versa).* IT Infrastructure Library. https://blog.itil.org/2014/08/what-it-service-management-can-learn-from-the-agile-manifesto-and-vice-versa/

Mamaev, A. (2018) *Flowers Recognition*. Kaggle. https://www.kaggle.com/alxmamaev/flowers-recognition

SAURABH9745 (2020, November 28). *The Key Concept of Scrum in Machine Learning.* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/11/the-key-concept-of-scrum-in-machine-learning/

UTOR (2020, October 15). *Software Testing Stages of Agile and Traditional Methodologies.* UTOR. https://u-tor.com/topic/software-testing-stages-explained#list_title0