Plant Disease Detector

Eduardo Lemus, Muslim Razi, Robert Campbell, Ryan Choy

***Abstract*— Agriculture in the United States is a major driver of economic growth, accounting for 11% percent of total employment and $132.8 billion dollars to U.S. GDP in 2017 [1]. Increasing efficiency and reducing waste is a key research area in agricultural, particularly as climate change creates favorable conditions for pests and diseases to proliferate throughout a crop. Our project provides a tool to help farmers discover diseases within their crop by using IBM's visual recognition API to identify diseases of interest to farm owners. The tool provides farmers a platform to train a machine learning model customized to their problem while also providing a dashboard for visualizing past predictions, if needed. The model is trained and validated on the PlantVillage-Dataset.**

I. Introduction

The Plant Disease Detector was designed to help farmers monitor plant health. The application offers an easy to use user interface, while providing access to cutting edge technology. The program gives the user (1) ease of use when interacting with the application, (2) accurate identification of plant, and (3) accurate health status of the plant. The tools used to assure the best user experience include React on the frontend, Node.js on the backend, and Python along with IBM Watson Visual Recognition, among others. In all, the application provides farmers with a customized solution to their problems.

II. Project Design

A. Web Interface

The front end website is used with ReactJS as the base framework used. ReactJS is a component based framework developed by Facebook as a means of creating a website. The UI framework that i have used in conjunction to ReactJs is React Bootstrap as well as Ant Design frameworks. Pages that the users are able to access are the home page (Upload images and predictions), the about page (Get information about what our website is for), the contact page (List of people and their information who worked on the project), the login and signup page (Allows users to register and log into the website), and the user dashboard page (Allows users to store and view previously uploaded images).

On our website, we have also used axios APIs. Axios is a promise based HTTP client for the browser and node.js. It is used to send and retrieve information to our databases using GET and POST operations. When users want to send data to be processed, a GET (non-sensitive data) or POST (sensitive data) operation will be performed and either a response from the data , which will contain information about the data sent, or an error if something bad has occurred.

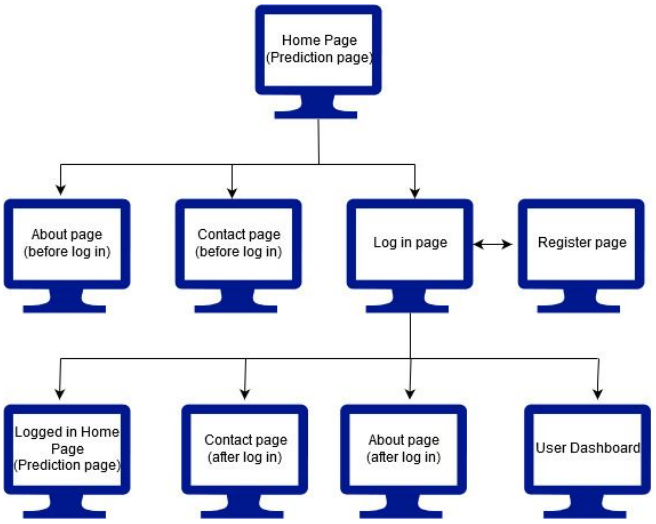

Figure 1 : GUI component diagram

Using the website, users are able to upload an image of a plant, then would get a result that shows whether the uploaded picture would be diseased or not, and the type of species it is. Users are also able to Log into the website on out Log in page after they signed up with us. The users will then be able to access the user dashboard page which the users are able to store images they have uploaded as well as the predictions of said pictures.

B. Data Science

In order to correctly classify user images, we used a dataset containing images of common farm crops with both healthy and unhealthy images for each species. The images are labelled with species and disease, and each image shows a single subject plant leaf against a varying background. The images are each 256x256 jpegs. As shown in figure X, the dataset is unbalanced in favor of the tomato class.

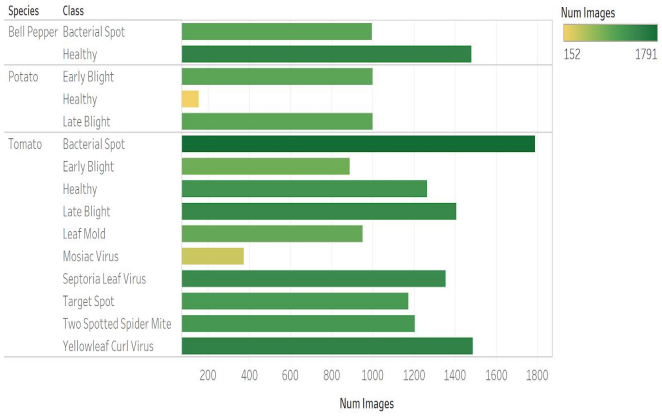

Figure 2. Number of images in training dataset per disease and species

However, since some diseases overlap across the species, we designed two approaches. The first approach treats the combined disease and species as a unique class. This approach yields a model with fifteen classes, one for each disease/species combination. This approach was tested by transferring learned features from MobilenetV2 pre trained on ImageNet. After training, the model was tested in deployment using tensorflow js.

The second approach separates disease and species into two models, one for classifying disease and the other for species. This approach was deployed on IBM's Visual Recognition using the custom classifiers option. Each image uploaded to our classification service is forwarded to both models. This approach is the one currently in deployment.

Species: Potato

Disease: Leaf Mold

Species: Bell Pepper
Disease: (None)Healthy

Species: Tomato
Disease: Septoria Leaf Spot

Figure 3. Sample images from training dataset and corresponding species and disease labels

C. *Backend*

The backend was written using Node.js enabling the user to manage files uploaded to the web application. Files may be uploaded, downloaded, or deleted. Images are organized based on the file name and stored into a folder pertaining to each user. User information and files are stored in Amazon's Simple Storage Service (Amazon S3) and retrieved using API calls.

Multiple images can be uploaded at the same time, andt have no size restrictions. Files must be uploaded in JPEG or PNG format. An algorithm was generated to determine what user a file corresponds to.

Each user is identified based on a unique JWT token generated by Amazon Cognito, once the user creates a new account. Users who do not wish to sign up may still use the web application with limited features. Without a personal account one can only upload images and view results. Account holders have no restrictions and are able to save files and review previously uploaded files in their folder.

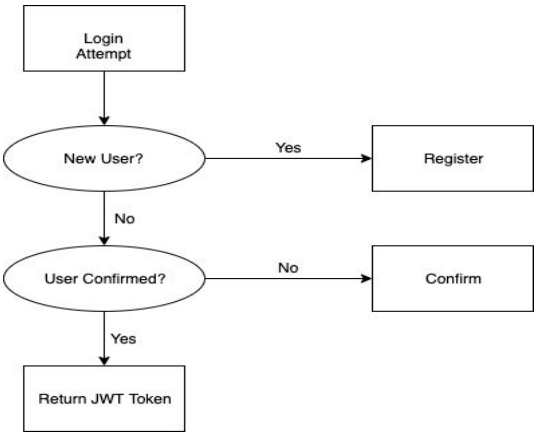

Fig. 4. User flowchart

D. *Deployment Architecture*

Two cloud platforms, AWS and IBM Cloud, were used to deploy the entire application. The web application was containerized using Docker and deployed on Kubernetes through IBM Cloud. For Continuous Deployment, IBM DevOps toolchain was used and deployed on Kubernetes as well. The toolchain managed AWS tokens by setting them as environment variables during deployment which allowed the web application to access AWS resources. An intermediate container was used to build the frontend and copy the artifacts to the backend container. Thus, only a single container containing the frontend and backend needs to be deployed.

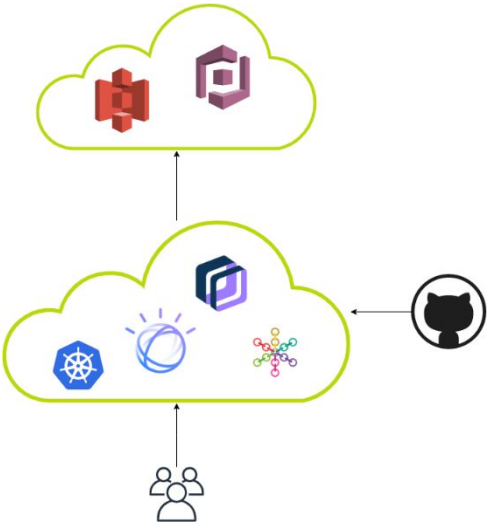

Figure 5. Multi-Cloud Architecture

III. Results

A. *Transfer Learning with Tensorflow*

As a baseline, we trained a model using keras and tensorflow 2.0 on Google's Colab platform. The model we choose was MobileNetV2, pretrained on ImageNet. MobileNetV2 was chosen because it provided a good balance of performance and classification speed.

Model training was conducted in two steps: training with a locked feature extractor and further training to fine tune features for task specific application. In the first step of training, the feature extractor of the model was locked to prevent training and a dense layer with 15 classes is added. Through training, this dense layer then takes features as input from the feature extractor portion of the network and maps this tensor into a vector with shape *<num_classes, 1>*.

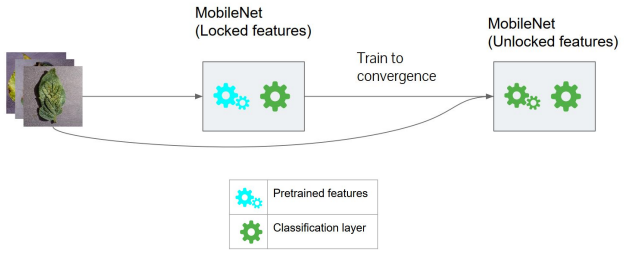

Figure 6. Training pipeline for transfer learning using pre trained MobilenetV2

As shown in figure 7, fine tuning the feature extract (~final 100 layers) leads increased performance even after just ten epochs of training.

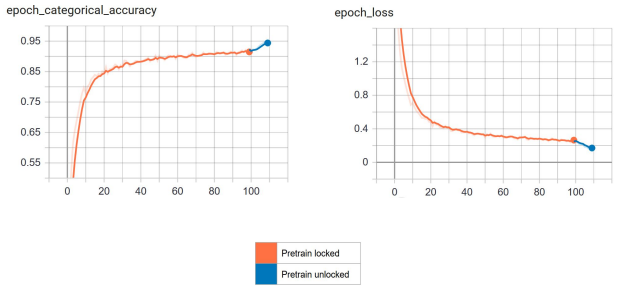

Figure 7. Training accuracy results of transfer learning using Mobilenet V2 with fine turning.

B. *IBM Watson Visual Recognition*

In addition to training a model with keras, we also used IBM's Visual Recognition service to predict species and disease from a leaf using the PlantVillage Dataset. Since Visual Recognition manages much of the machine learning pipeline, we split the prediction into two models: a species classifier and a disease classifier. This meant that there are three classes for the species classifier: potato, tomato, and bell pepper. The provides an advantage in extensibility since the model can be retrained with new classes and examples as needed by users with differing needs and crops.

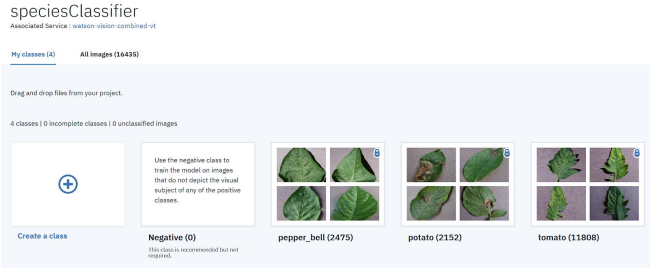

Figure 8. Species classification model on Visual Recognition

The disease classifier has one class per disease in the training dataset, as well as the negative class which is trained with healthy images. Similar to the species classifier, new diseases can be added and the model retrained as needed to adjust for user needs in the future. Both models were trained for approximately eight hours using IBM's Watson Studio service and the train models serve as the backend for prediction for the current iteration of our application.

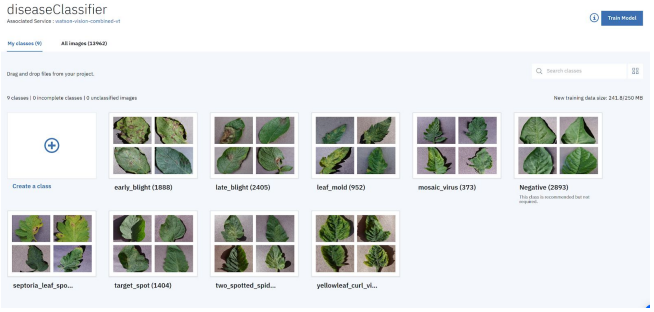

Figure 9. Disease classification model on Visual Recognition

IV. Conclusion

In sum, we developed an application that would help farmers use image recognition for the early detection of diseases in their crops which could lead to increased crop yields by putting more data in the farmers hands. The application uses a web interface that allows farmers with a login to classify images and review past insights, while also providing an interface for prospective users that shows the potential of the predictive model. For future work, we have considered additional features such as a mobile app, geotagging images, integrating multispectral overlays from Google Earth for additional insights into long term climate trends, and providing custom trained models per user.

V. References

[1] Chen, Ying, et al. "IBM Watson: How Cognitive Computing Can Be Applied to Big Data Challenges in Life Sciences Research." *Clinical Therapeutics*, vol. 38, no. 4, 2016, pp. 688–701., doi:10.1016/j.clinthera.2015.12.001.

[2] Mohanty, Sharada P., et al. "Using Deep Learning for Image-Based Plant Disease Detection." *Frontiers in Plant Science,* vol. 7, 2016, doi:10.3389/fpls.2016.01419.

[3] Sandler, Mark, et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520., doi:10.1109/cvpr.2018.00474.

[4] Emmanuel, Tairu Oluwafemi. “PlantVillage Dataset.” *Kaggle*, 30 Oct. 2018, https://www.kaggle.com/emmarex/plantdisease.

[5] https://reactjs.org/

[6] https://ant.design/

[7] https://react-bootstrap.github.io/