ECEN 404 Bi-Weekly Presentation
Team 57: Deep Learning for Hydroponic Soybean Growth

Team members: Samuel He, Mary Hughes
Sponsor: Sambandh Dahl, Krishna Gadepally

Project Summary

- **Problem Statement:**

  Researchers take time to track the solution and day of growth of a hydroponically grown plant

- **Solution**

  Deep learning model and user interface that tracks Day of Growth and outputs other growth data that may be useful to the user

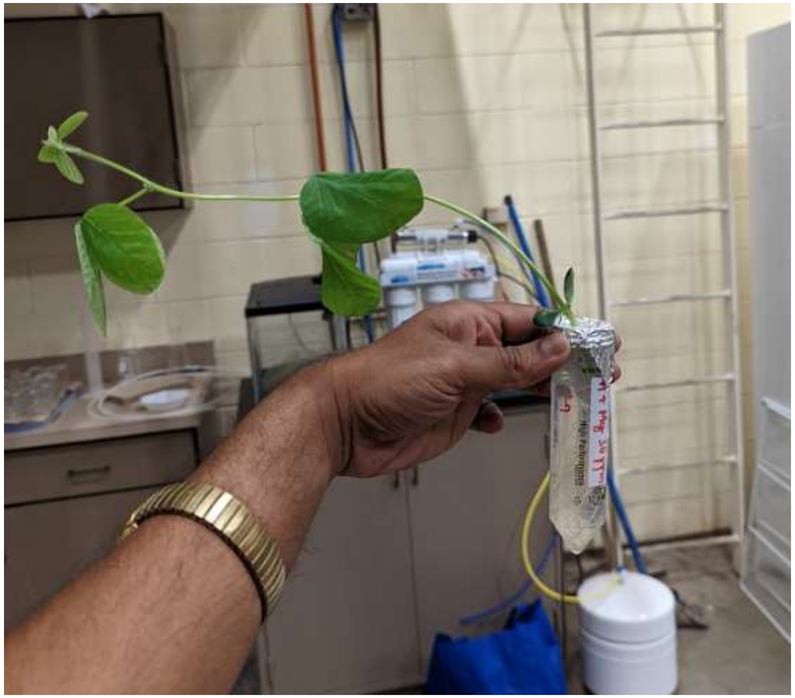
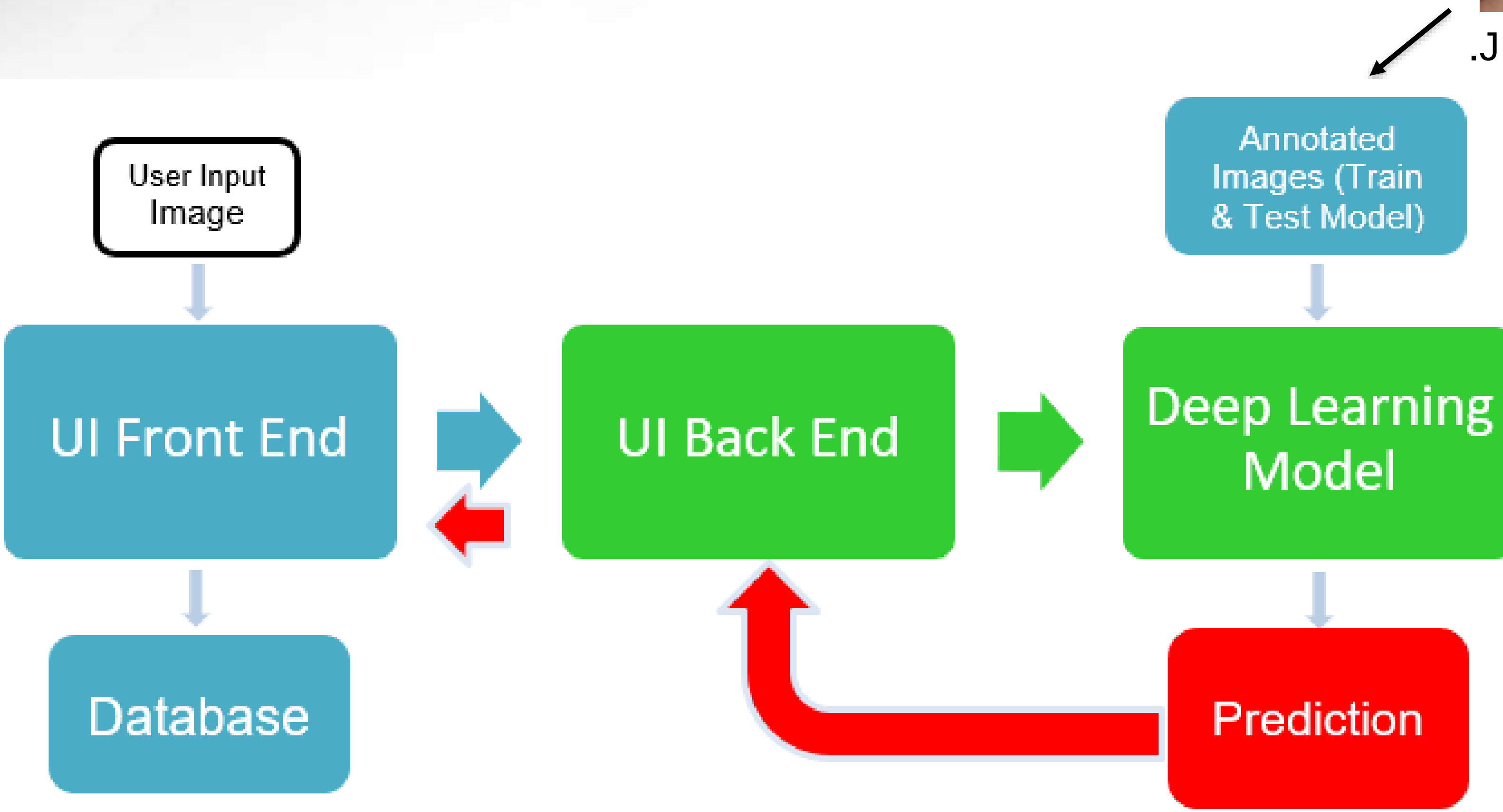


Image 1. Sample Data Image

System Diagram

.JPG Data



Samuel He
Mary Hughes

Major Project Changes (since update 1)

- Sponsor added more data and requirements - this data will be generated in the backend and displayed on the UI, and some of it will also be interacting with the database.
  - linear interpolation analysis of given data in backend
  - which nutrients are vital for plant growth
  - which samples have the largest water intake
  - percentage of biomass of each sample

Project Timeline

DL Model Complete

Database Built on Cloud Platform

AWS Up & Running

UI & Database Connected & Backend Data Implementation

Integration of UI with Model

Reformatting Frontend

Model & AWS Validation

UI & Database Validation

Full System Validation

Demo

Dwight Look College of ENGINEERING TEXAS A&M UNIVERSITY

Subsystem: Deep Learning Model

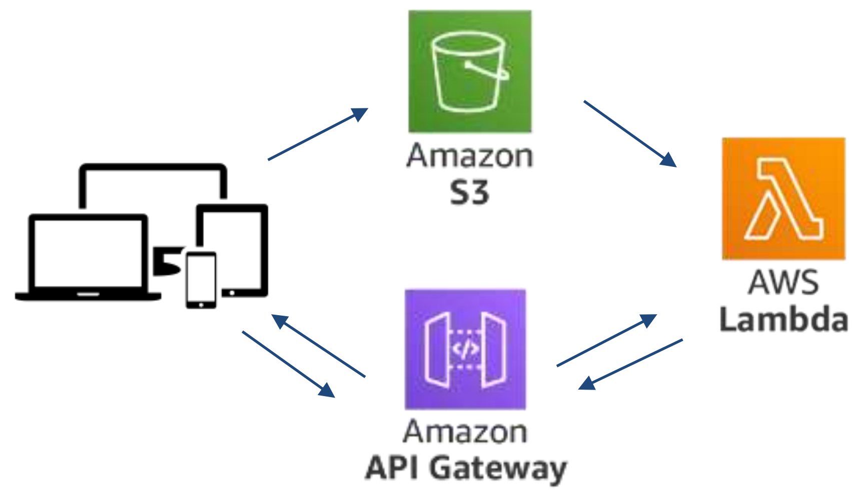
Accomplishments since 403 15 hours of effort	Ongoing progress/problems and plans until the next presentation
Completed DL model code Ran/debugged DL code Completed training of DL model Determined ideal method for using AWS for model deployment/hosting	Getting AWS to run smoothly Once there is a working API that calls the model, it can be provided to Sam. Then the DL backend will be integrated with the UI frontend. Need to determine how the analysis of the new data will be carried out.

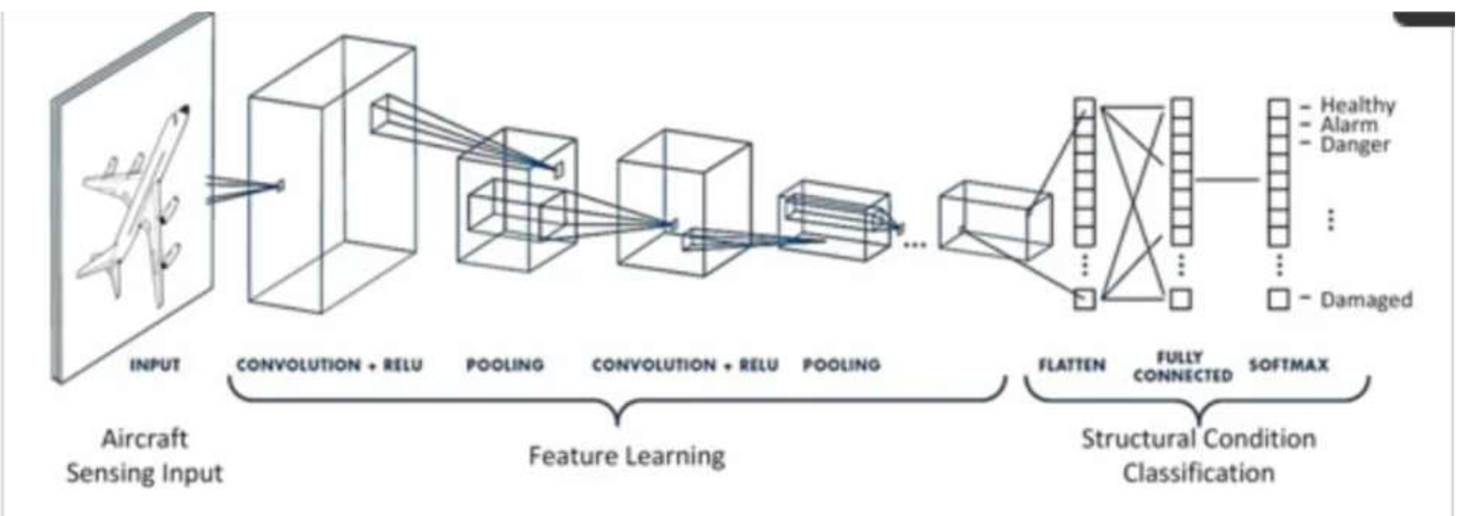
Mary Hughes

Subsystem: DL Model

```
#add convolution and pooling layers to model -> this par
model.add(layers.Conv2D(128, (3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(32, (3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))

#Flatten the last layer of pooling, so we can do a coupl
model.add(layers.Flatten())
model.add(layers.Dense(128, activation = 'relu'))
#Dense(x, activation = 'softmax') -> x = nodes on the la
model.add(layers.Dense(20, activation = 'softmax'))
```





Samuel He

Subsystem: User Interface

Accomplishments since last update 35 hours of effort	Ongoing progress/problems and plans until the next presentation
Deployed database to Heroku Cloud Services Updated Database Design Wrote code to connect Heroku Cloud application backend queries to Cloud database	Update database design to hold new information for analysis Dynamically display all database tables on frontend with finished styling Create data analysis algorithms on backend

Samuel He

Database deployed on application

SERVICE heroku-postgresql PLAN mini BILLING APP soy-api2

Empty database tables

```
soy-api2::DATABASE=> \dt
            List of relations
 Schema |     Name      | Type  |     Owner
--------+---------------+-------+----------------
 public | dry_weight    | table | tbptunbssokuks
 public | image_data    | table | tbptunbssokuks
 public | solution_data | table | tbptunbssokuks
 public | water_uptake  | table | tbptunbssokuks
(4 rows)


soy-api2::DATABASE=> SELECT * FROM solution_data;
 sample_id | solution | concentration | calcium | magnesium | sodium | potas
sium | boron | co_3 | hco_3 | so_4 | chlorine | no3_n | phosphorus | ph | co
nductivity | sar | iron | zinc | copper | manganese | arsenic | barium | nic
kel | cadmium | lead | chromium | fluorine | cb
-----------+----------+---------------+---------+-----------+--------+------
-----+-------+------+-------+------+----------+-------+------------+----+---
-----------+-----+------+------+--------+-----------+---------+--------+----
----+---------+------+----------+----------+----
(0 rows)


soy-api2::DATABASE=> SELECT * FROM image_data;
 image_id | image_name | day_prediction | image_data | segmented_image | sol
ution
----------+------------+----------------+------------+-----------------+----
------
(0 rows)


soy-api2::DATABASE=> SELECT * FROM dry_weight;
 sample_id | solution | dry_weight
-----------+----------+------------
(0 rows)


soy-api2::DATABASE=> SELECT * FROM water_uptake;
 sample_id | solution | uptake_amount | uptake_date
-----------+----------+---------------+-------------
(0 rows)
```

Database Design Update

Query Code

```
#INPUTS:
#solution: string (varchar)
#uptake_amount: float
#uptake_date: date     date format: yyyy-mm-dd
def insert_water_uptake(solution,uptake_amount,uptake_date):
    conn = psycopg2.connect(DATABASE_URL, sslmode='require')
    cursor = conn.cursor()

    cursor.execute("INSERT INTO water_uptake (solution, uptake_amount, uptake_date)" +
                   " VALUES(%s,%s,%s)",
                   (solution,uptake_amount,uptake_date))

    conn.commit()
    count = cursor.rowcount

    print(count, "Record inserted successfully into table")
    cursor.close()
    conn.close()
```

Execution Plan

	1/30/2023	2/6/2023	2/13/2023	2/20/2023	2/27/2023	3/6/2023	3/20/2023	3/27/2023	4/3/2023	4/10/2023	4/17/2023	4/26/2023
Create Database on Cloud Platform												
Link Backend with Database												
Display Database Info on UI												
Deploy Frontend												
Reformat Frontend												
UI Autoscaling												
Validate Frontend												
Format Dataset for Use in Model												
Finish DL Model												
Deploy Model												
Build AWS Lambda Function												
Proper Calls w/ AWS API Gateway												
Data Analysis in Backend												
Debugging AWS items												
Validate AWS fully functioning												
Validate Model Accuracy												
Integrate Frontend & AWS items												
Validate Integrated System												
Update Presentations												
Final Demo												
Engineering Project Showcase												

Samuel He
Mary Hughes
Shared Goals

Complete
In Progress
Not Yet Started
Behind Schedule

Validation plan

Paragraph #	Test Name	Success Criteria
3.2.1.3	UI Image Input	Users can upload up to 50MB of image data to website and receive a confirmation response within 1 second
3.2.1.3	Webpage Autoscaling	Webpage autoscales properly to mobile and desktop screens
3.2.1.3	Webpage Interactivity	Webpage navigation interactions are functional
3.2.1.3	Database Outputs on Frontend	Webpage frontend has all database prediction information displayed properly
3.2.1.3	Input Delivery to Back End	Image is successfully being delivered to the backend from the front end of the UI in <1sec
3.2.5.1.1	Application Failure Detection	Internal testing properly identifies when the application fails to communicate with the deep learning model.
3.2.5.1.1	Application Failure Response	Webpage gives user a correct error message when incorrect image formats are uploaded
3.2.5.1.1.1	Model Failure Detection	Application correctly detects if the model has given a valid input to the UI.
3.2.1.1	Day of Growth Identification	The deep learning model is correctly identifying the day of growth of an input.
3.2.1.2	Nutrient Solution Detection	The deep learning model is correctly identifying the nutrient solution of an input.
3.2.1.3	UI Delivers Input to AWS with API Calls	User Input images are successfully delivered to AWS using the APIs built in API Gateway.
3.2.1.3	AWS API Calls to Lambda	The User Interface Back End API calls work as expected, and can properly connect to AWS Lambda.
3.2.1.3	Lambda Properly Communicates with Model	AWS Lambda Function successfully delivers input to and recieves predictions from the DL Model.
3.2.3.2.1	UI Output Delivery	An output is being delivered to the UI in the correct format, including the prediction and the accuracy of prediction.
N/A	Full System Demo	The application and deep learning model process input as expected and deliver correct output to the UI.
3.2.1.3	UI Backend Communication with Model	The User Interface Back End API calls work as expected, and can return a prediction in a 3rd party testing platform.
3.2.1.3	UI Readability	UI design is clean and understandable, easy to use on multiple brightness levels

Validation plan

Methodology	Status	Responsible Engineers
Upload 20 different image sets to the User interface, starting at 1MB and incrementing by 5, up to 50MB	UNTESTED	Samuel He
Test the mobile view of the website on at least 10 different mobile views, using React Native Layout Tester. Compare results.	UNTESTED	Samuel He
Test button pressing functionalities of each button on navigation.	UNTESTED	Samuel He
Upload 50 images and monitor predictions for them both individually and altogether. Input images into model directly. Compare results.	UNTESTED	Samuel He
Monitor database to see if corresponding images and predictions are sent out and received. Send out time and retrieval time will be monitored by test cases in React.	UNTESTED	Samuel He
Restrict access from the application to the model. Attempt to upload an image to the model.	UNTESTED	Samuel He
Upload a set of 15 different files that are not .jpg or .jpeg.	UNTESTED	Samuel He
Create an invalid prediction response on the backend, and attempt to upload an image to the model.	UNTESTED	Samuel He
Create 264 test cases with corresponding images that cover all of the different categories. Compare results with pre-determined day of growth inputs.	UNTESTED	Mary Hughes
Create 66 test cases in Python with corresponding images that cover all of the different categories. Compare results with pre-determined nutrient solution inputs.	UNTESTED	Mary Hughes
Test the API using POSTMAN. Verify with AWS Consoles that the API was used.	UNTESTED	Mary Hughes
Test using POSTMAN. Verify in AWS Lambda Console that the Lambda Function has been used at the time the POSTMAN request was sent.	UNTESTED	Mary Hughes
Test using POSTMAN. Verify in AWS SageMaker Console that the model endpoint has been accessed and returned a prediction at the time the POSTMAN request was sent.	UNTESTED	Mary Hughes
Upload 20 different images to POSTMAN, one from each day of the growth cycle represented in the dataset, and verify the output shown in the POSTMAN console is the correct format for the UI to receive and interpret properly.	UNTESTED	Mary Hughes
Upload a set of 20 images, and compare their individual predictions with the model to the UI output.	UNTESTED	Shared
Firstly, validate the Frontend communcation with the Backend. Secondly, send 30 images to the model using both Postman and the UI. Compare response results.	UNTESTED	Shared
Compare readability on at least 5 different monitor display/brightness settings.	UNTESTED	Shared

Questions?