Oregon State University
College of Engineering

CS Capstone Design Document

November 27, 2018

Pure Grass Seed Sorter

Prepared for

Oregon State University Seed Lab

Dan Curry _____ Signature _____ Date

Prepared by

Group 32
The Seed Team

Bharath Padmaraju _____ Signature _____ Date

Kevin Deming _____ Signature _____ Date

Haoxuan Zhan _____ Signature _____ Date

Cong Yang _____ Signature _____ Date

Christopher Wohlwend _____ Signature _____ Date

Abstract

The primary objective of the project is to automate grass seed sorting. The members of the group will be building software to be able to discriminate between pure grass seeds from all other plant seeds including but not limited to weeds, and crop seeds. The method we will utilize will be a combination of implementing computer vision and deep learning algorithms to accurately identify off type seeds under a high definition camera. This will vastly reduce the stress and workload imposed upon seed analysts, and likely speed up the sorting process. Not only does this project offer a opportunity to improve seed research, but also creates possibilities in other fields where our technology can automate menial and repetitive tasks.

CONTENTS

1 Introduction

1.1 Scope

This document describes the design and creation of the system developed to autom ate the seperation of pure and off-type seed from samples. This document will describe the necessary procedures, technologies and algorithms required to assemple a prototype, and how those technologies should be used together.

1.2 Intended Audience

This document is intended to be read by individuals interested in the development and design of our seed seperator prototype. This document will also serve to allow those invested in our progress to understand our design, and compare it to our progress.

2 Definitions

Neural network: The neural network is a matrix that is trained using a deep learning algorithm.
Test sample: Our test sample contains 25,000 seeds.
Target seed: It is the seed we want from test sample.
Purity: The authenticity of the target seed.
Off-type seed: The seeds we wish to distinguish from the target seed.
Webcam: A plug and play small scale high definition camera that is lightweight portable.
Raspberry Pi: A lightweight single board computer.
Raspbian OS: A linux based distribution operating system used by default by most Raspberry pi's
External HDD: A usb powered hard drive, with extended storage.
GPU: Graphics processing unit, optimal for tensor operations.
Movidius Compute Stick - VPU: Vision processing unit, used for storing neural networks.
OpenCV: Open source computer vision library.
SQLite: Database schema.
Convolution: Refers to the process of transferring data from one node to another.
NCSDK: Neural compute stick software development kit.

3 Conceptual Model for SDD

3.1 Software Design Context

Our project will be composed of various software pipelines that need to be interfaced together in order to have a contiguous reliable model. The way we will accomplish this is by we laying out all the required components and exploring how they can interface between each other. Furthermore, if a workaround is necessary then we will outline it and warn the user of potential pitfalls and drawbacks.

At the heart of the project we will be utilizing a raspberry PI which will coordinate the various software components required for the build. The raspberry pi will be running on the raspbian OS. Since we will need to train the neural network, data will need to be collected and stored and because the raspberry pi has limited storage we will be mounting

an external hard drive to it. Moreover, the external hard drive will be connected to a database. For our database we will be using SQLite to manage our data.

For collecting images we will be using a USB webcam to take pictures and store them on the external hard drive attached to the raspberry pi. The external hard drive will also be used to store the processed training data that will also be attached to an external computer with the necessary resources needed to train the network, namely a powerful Nvidia GPU running CUDA drivers. Training will be done through the Keras deep learning framework and the resulting neural network model will be stored into a movidius compute stick as a .h5 file.

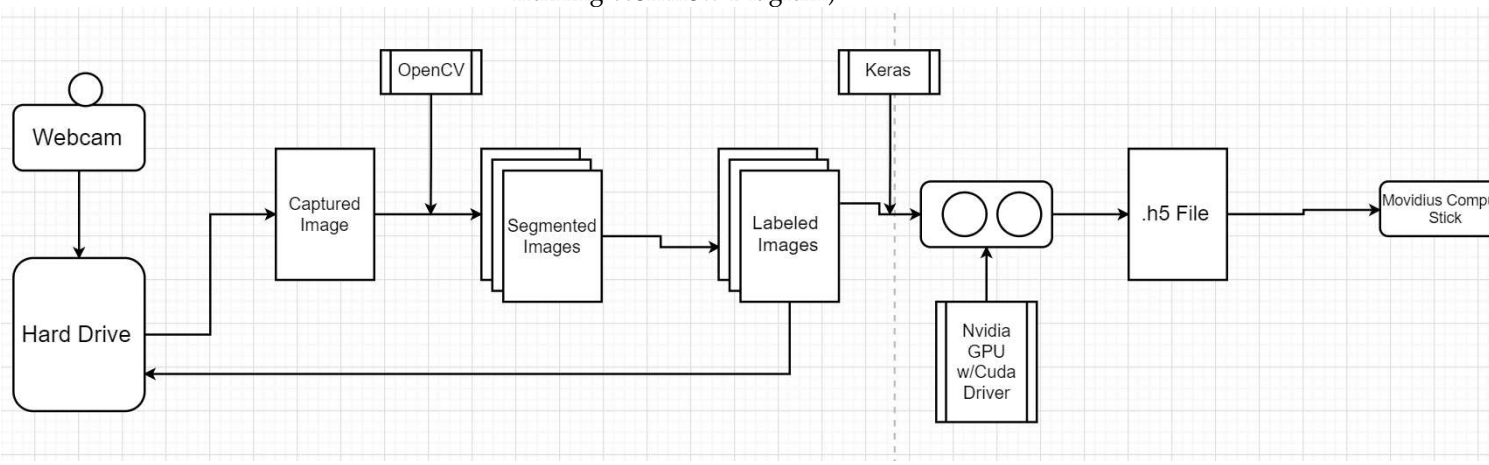
Finally, we will be utilizing the aforementioned Intel Movidius Neural Compute Stick running NCSDK to classify the real time images taken by the webcam, and the images will be stored in the external hard drive and referenced to by the database with the right classification.

3.2 SDD Life Cycle

Our first step will be to manually capture images through the raspberry pi and a usb webcam to store them on the external hard drive. Once the images are taken, we will use a python script with the OpenCV library to segment the image into individual seeds and manually classify/label each one. After segmentation we will store them once again into the library into labeled training/validation directories while deleting the original image. When we have the properly labeled data, we will write another piece of software utilizing the Keras library to develop the Convolutional Neural Network architecture and train it on a GPU. The training of the GPU will be with the previously collected image sets in the database. The resulting trained matrix will be stored as a .h5 file on the Movidius VPU computer stick. This process is just to train the network for future product integration.

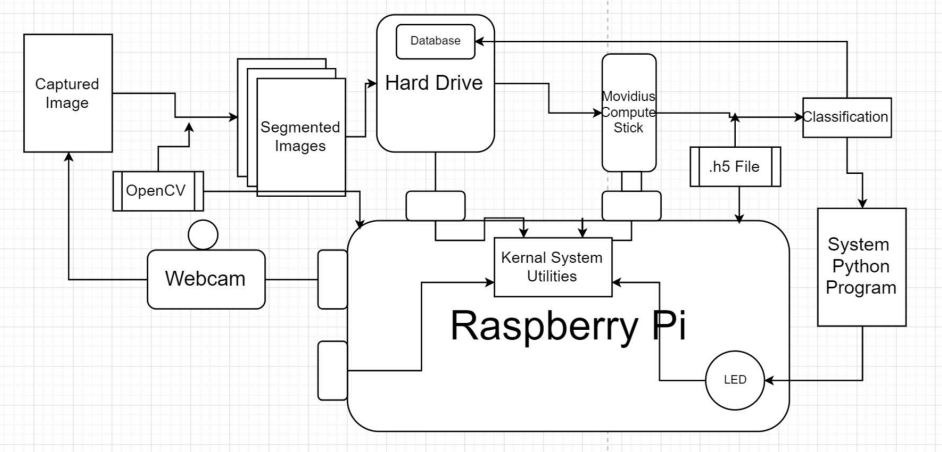
Fig. 1. (

Training Workflow Diagram)



The initial steps in our final product is similar to the training process. We start with the capturing of images and segment them in the same python script using opencv. The partitioned images with individual seeds are stored into a specified folder for classification. A separate python program will be developed with the .h5 file from the compute stick loaded to classify images fed through the hard drive, the output of the classification will be sent to two external sources. One, back to the hard drive with references to the images to be stored onto the database, two the output will be sent to another python program, that will invoke a system call to light up an LED if a bad seed is detected.

Fig. 2. (



Raspberry Pi Diagram)

4 DATABASE

4.1 Context

The database exists to store the images used by the neural network, including training data, and data collected at runtime for future training and testing. Images will generally be added one at a time during runtime, but during testing and development the capability to add several, potentially hundreds of images per transaction.

4.2 Dependancies

The database will not depend on any other subsystem for operation. It is needed for training the neural network computational stick, however, once the neural network is trained the database will be used to store results and review the effectiveness of the system. Thus it will have little impact on the operation of the final device.

4.3 Structure

Making use of SQLite, the database itself will consist of a single file, alongside a folder containing the named image files. The database will consist of several tables, one for the training, one for verification, and one for images collected during runtime. The training table will consist of the following columns: index (integer), image address (String), actual type(boolean).

Fig. 3. testing database example

Index	Image Address	Type
1	images/image34	TRUE
2	images/image36	TRUE
3	images/image12	FALSE
4	images/image15	TRUE
...		
750	images/image750	TRUE

Fig. 4. verification database example

Index	Image Address	Type	Flag
1	images/image34	TRUE	FALSE
2	images/image36	TRUE	FALSE
3	images/image12	FALSE	TRUE
4	images/image15	TRUE	FALSE
…			
750	images/image750	TRUE	FALSE

The verification table will consist of the same columns with the addition of a flag column to indicate that during verification, the neural net missclassified the image in the entry.

Finally the runtime table will consist of an index (integer), image address (String), and guess(boolean).

Fig. 5. runtime database example

Index	Image Address	Guess
1	images/image34	TRUE
2	images/image36	TRUE
3	images/image12	FALSE
4	images/image15	TRUE
…		
750	images/image750	TRUE

Each of these tables will share a database file, and a folder to store images in. While not in use, this foder could be compressed into a tar archive, but would be uncompressed before runtime or during startup.

4.4 Logical

The database will be coded to support the following functions:

```
#returns the image address associated with the given the db index.
entry getImage(index, table);

#returns the image addresses associated with the given index range.
getImages(start, end, table);

#adds [num] entries to the db table.
void addImages(entries, int num, table);

#performs the given query, and returns an array containing the results.
query(query);
```

The getImage, and getImages functions are designed to allow easy querying of images iteratively during neural network training. In such cases, we only care that each data point gets used, and filtering results based on type or guess

is unlikely to be necessary. The addImages function allows the system to add the seed images collected during runtime to the database. The query function allows us to move entrys to different table in order to increase the size of the training or verification data sets.

4.5 Interaction

The database will interact with other subsystems by receiveing and servicing requests to access data in the database. During training, this will be returning entries from the training and verification data sets. At runtime, the system will present images to save via the addimages function. Each iteration, developers will utilize the query function to perform less typical operations, including migrating runtime data into the training and verification tables.

4.6 Resources

The database will require little RAM, but will require its own drive in order to support storing a large number of images. We estimate that drive size will be close to 10 GB, but depending on the size of the training dataset, the amount of memory needed may change slighly.

4.7 Rational

The database is needed to efficiently organize data critical to training the neural network, but our use case is fairly lightweight. SQLite provides an incredibly lightweight solution that can be easily used both during training, and during data collection, as well as giving developers the ability to quickly query data during testing and development. Using image pointers as opposed to storing the images makes the database slightly less portable, but should make passing images from subsystem to subsystem more efficient.

5 Neural Net

5.1 Context

Our neural network is the heart of the project, it allows us to process quickly and accurately classify images that are fed through it. We will be feeding training images from the hard drive to train it, but it will spend most of its life in deployment on the Movidius Neural Compute stick. Which will be plugged into the raspberry pi.

5.2 Composition

We will build the neural network using Keras which is a wrapper library for the popular tensorflow library. This will allow us to quickly test and alter code.

5.3 Structure

The structure of the neural network will be a convolutional neural network optimized for image processing.

5.4 Logistics and Dependencies

Because the neural network will interface our other devices and programs through the raspberry pi we need to make sure that the pi has Tensorflow and Keras installed and all the required dependencies including Numpy, Pillow, Scipy, Jupyter, matplotlib and scikit-learn.

5.5 Algorithms

There are a myriad of algorithms internally within CNNs. We can pick and choose between different convergent optimizing functions however we will use Adam optimizer. Furthermore, there is pooling, convolutions, different activation functions, ReLu, loss functions, dropout, and decays. All of these factors will play a role into how effective our algorithm will be as a whole, and it will be up to us to fine tune the overall architecture.

5.6 Rational

A CNN is necessary because of its ability to apply filters and classify images even if the subject is awkwardly oriented. This is valuable for our project because the seeds being fed through the conveyor belt will be in different locations and orientations. Furthermore, a CNN ensures the ability to distinguish between noise and positive signals continuously due to its lack of a memory module.

6 CAMERA

6.1 Context

The camera subsystem is responisble for capturing images of seeds on the conveyor belt on a timed interval and sending the images to the rasberry pi for image processing. The camera subsystem needs to provide clear and quick images to the vision processing subsystem, so that the seeds captured by each image can be identified and classified by the image processing subsystem.

6.2 Composition

This subsytem will make use of a camera, usb connector, image control program, and a constant light. The camera needs to take high quality images fast enough to capture 15 or more small grass seeds per second.

6.3 Structure

The image control program will need to send a signal repeatedly with a set interval in order to capture the images at a speed that is effective and useful for our system. The usb connector will be used to send signals and data in and out of both the camera as well as the raspberry pi. These images will then be processed by a different subsystem. The constant light will need to provide a source of light to the conveyor belt so that the images used dont change even with outside sources of light.

6.4 Rational

In order to get high quality images the team decided that a resolution of 1920 x 1080 would be sufficient for high definition quality to determine good and bad seeds. For image speed the camera would need to run more than 10 frames per second and 30 frames per second will allow us to be well above what we need. The camera we decided to use for our project is the Logitech C920 Pro Webcam which is able to take images at 1920 x 1080 resolution as well as doing so at 30 frames per second. This camera also comes with mounts attached to it and the mount allows the camera lens to look up or down for adjusting.

The next part of this component is the program that sends a take image signal to the camera and saves the image in a specified way. The program will use a timer to take an image, wait, then take another image, wait, repeat. This will

allow for a controlled amount of images sent as well as capturing all the seeds that go by. In order to do this OpenCV is an open source library for computer vision in python that is capable of taking images and processing them. Since Python is the code that we are planning on using for connecting our system OpenCV is perfect for doing image taking and processing.

The light that we have will need to be strong enough to reduce the effect of outside light sources that change throughout the day. Since the program could be running at any time, any outside variables need to be removed in order to get consistant and accurate results. The light would need to be mounted above the seeds near or next to the camera in order to reduce shadows. For this we will use a generic desk lamp since it is able to be moved and adjusted and can give consistent lighting.

6.5 Logical

For the camera subsystem we will be using OpenCV, an open source vision software, in order to capture and analyse initial images. The program writen with OpenCV will send a call to the camera to take an image every interval of time. The interval that we use to take images will be faster than the processing subsystem that way it doesn't have to wait for a return signal to take the next image. This will streamline the image taking and updating process so that it is not dependent on the speed of the next subsystem.

6.6 Interaction

The camera control program will interact with the camera, the hard disk, and the image processing system. The way it interacts is by sending a signal to the camera, then saving the image back to the hard disk, and then the image processing system will take the image from the hard disk and process it. The control program will then continue to to send signals on the timed intervals.

6.7 Resources

The camera system requires only a small amount of processing and storage. It needs enough storage to save a single high definition image that the camera takes and processing power to conisitently send a signal to keep taking images. It also needs to use one usb port on the Raspberry pi to connect the camera.

7 Vision Processing

7.1 Context

The goal of the vision processing subsystem is to transform each image received by the camera, and tranform the result into several individual images, one for each identified seed, so that the neural net can classify each of them. This system will be responsible for both identifying each seed present on the received image, and creating a smaller images containing individual seeds.

7.2 Composition

This system will be composed of a single algorithm, making use of the OpenCV image processing library. By making use of OpenCV, the algorithm should be reduced to initial filtering, noise removal, and thresholding.

7.3 Algorithm

Filtering will take the initial image and create a boolean array to establish whether each pixel made it through the filter. OpenCV supplies a threshold() function which can easily apply such a filter to the images, however testing will be required to identify the exact parameters required to seperate the seeds from other seeds and the background. Since the background will be quite consistant, this filter should be easy to create, but will be more formally established during testing.

After filtering we remove noise removal, which can be accomplished easily using the openCV morphologyEx() MORPH_OPEN operation.

Finally, utilizing OpenCV's distanceTransform() function and another filter, we can ensure each identified block is seperate, and use OpenCV's connectedComponents() function to identify each block and its coordinates. With those coordinates we can create many smaller images containing single seeds based on the initial image.

7.4 Logical

The vision processing subsystem will have a single function facing the remainder of the project. This function will take a single image as a parameter, and return an array containing a set of images, each with one of the seeds identified.

7.5 Dependancies

The vision processing subsystem will require image data recieved from the camera subsystem. The neural network is dependent on this subsystem because it takes in the individual images and determines good or bad seed based on it.

7.6 Interactions

The vision processing subsystem will interact with both the camera, and neural network subsystems. The vision processing system serves as the in between code necessary to translate the image received from the camera into images that the neural network is capable of identifying.

7.7 Resources

The vision processing subsystem will run on the rasberri pi, use processing power to both identify and segment the images, and requires no additionaly memory.

7.8 Rational

It is not feasable for the neural network to classify the entire image at once, so we need a method to seperate each seed, and feed images to the neural network individually. OpenCV provides one of the most usable code libraries for image processing, and was identified as the best technology to use by our team. The vision processing algorithm has the potential to be quite complicated, so keeping it seperate from the other systems is important for maintainability.

8 User Interface

8.1 Context

The user interace subsystem needs to be capable of starting and stopping the execution of the entire system, and will be responsible for displaying any error messages that may arrise during execution. Because the scope of the interface is fairly minimal, we plan to implement it via the command line.

8.2 Composition

The user interface will operate in its own thread, with the ability to start the main program. Then the user interface can wait for further input, and set a flag telling the main thread to stop upon receiving a shutdown command.

8.3 Interaction and Dependancies

The user interface will primarily interact with the user, and the remainder of the program as a whole. Users will be supplied error messages, and give commands, while the UI starts the system, and sets flags for the system to respond to. The user interface will be necessary to start the system, and doesn't depend on any of the other subsystems.

8.4 Rational

A command line interface was choosen to minimize the amount of work required to get a functional prototype built. Since so little user interaction is necessary, this should have minimal impact on the usability of the system as a whole. By keeping user interactions in a seperate thread, we reduce the amount of time the main thread spends checking for input, which allows the system to be more efficient overall.

9 Conclusion

9.1 Rational

Each subsystem was isolated in the design so as to enable each part to be developed independantly. This will make it easy for us to identify when a task is complete, and makes testing each part of our code easy. Knowing that the task is unlikely to be practicle without machine learning, we have essentially built a pipeline that will enable a neural network to process the necessary data. Without this pipeline it would be impracticle to train a neural network to process an entire image.